Learning-based Online Query Evaluation for Big Data Analytics in Mobile Edge Clouds

Qiufen Xia†, Luyao Bai§, Zichuan Xu†, Weifa Liang‡, Omer Rana¶, and Guowei Wu†
† Dalian University of Technology, Dalian, Liaoning, 116620, China
‡ Australian National University, Canberra, ACT 0200, Australia
§ University of Tsukuba, Ibaraki, Japan ¶ Cardiff University, Cardiff, United Kingdom
qiufenxia@dlut.edu.cn, bailuyao1997@outlook.com, z.xu@dlut.edu.cn,
wliang@cs.anu.edu.au, RanaOF@cardiff.ac.uk, wgwdut@dlut.edu.cn

***Abstract*—The rise of big data brings extraordinary benefits and opportunities to businesses and governments. Enterprise users can analyze their consumers' data and infer the business value obtained, such as purchasing goods correlations, customer preferences, and hidden patterns. Meanwhile, with the emerge of big data processing frameworks, such as Hadoop and Tensor-flow, more and more mobile users are embracing big data analytics by issuing queries to analyze their data. In this paper, we investigate the problem of Quality-of-Service (QoS) aware query evaluation for big data analytics in a mobile edge cloud to maximize the system throughput while minimizing the query evaluation time of each admitted query, by exploring the materialization of intermediate query results. We consider dynamic big-data query evaluations where user queries arrive one by one without the knowledge of future arrivals, and the system needs to respond to each query by accepting or rejecting the query immediately. We propose an online algorithm for query admissions within a finite time horizon, the proposed algorithm can intelligently determine whether some immediate results during a query evaluation need to be materialized for later use of other queries, by making use of the Reinforcement Learning (RL) method with predictions. We finally investigate the performance of the proposed algorithm by simulations, and results show that the performance of the proposed algorithm is promising, by achieving a higher system throughput while reducing the average evaluation cost per query by from 20% to 52% compared to the comparison benchmarks.**

I. Introduction

In the era of big data, big data analytics is of paramount importance in many applications such as financial analysis, social interaction web sites, astronomical telescope service, etc. For example, Facebook-like social media sites can uncover usage patterns by analyzing the web site history records, such as click records and activity records, to facilitate their marketing decision. Such applications for big data analytics generally require the data being processed as quickly as possible to help users' decision-making; otherwise, the quite late query result may not have any business value at all, due to the dynamic nature of marketing.

Mobile edge computing has been emerged as a promising technology to enable real-time big data analytics in the network edge, by deploying servers or server clusters (*cloudlets*) in the proximity of users. Since cloudlets are very closely located with mobile users, big data generated by users can be uploaded to and processed at cloudlets in extremely low latency. In this paper we aim to enable time-sensitive big data analytics in mobile edge clouds (MEC) by providing efficient evaluation approaches for big data query evaluation. Specifically, we study the problem of QoS-aware query evaluation in a MEC, where queries arrive one by one without the knowledge of future arrivals.

The motivation of this paper is that many big data analytic queries usually share common sub-expressions, and thus share common intermediate results if they are evaluated on the same collection of datasets. If we materialize the intermediate results of some common sub-expressions as *materialized views* in memory, the evaluation of future queries can make use of the materialized results and thus its evaluation time can be significantly shortened, thereby improving the system throughput. Consider the following two queries as an example:

- *Query 1: Which students have registered to the university since January 1, 2018 and majored in* **Software Engineering***?*
- *Query 2: Which students have registered to the university since January 1, 2018 and majored in* **Computer Science***?*

Both queries have a common *sub-expression* and select the same range of dates from a base dataset. To optimize query evaluation performance, intermediate result of Query 1 can be materialized in main memory of a cloudlet which can be utilized when evaluating Query 2. The mentioned example happens in typical data warehouses where hundreds of queries need to be evaluated, these queries are often generated from a few patterns and thus have many sharing possibilities. Materializing the intermediate result is preferred, if the cost of materializing is much less than the cost of re-evaluating the queries. However, in traditional warehouses, it is commonly assumed that characteristics of successive queries are already given as a priori. In contrast, the future arrivals of queries typically are not known in advance, it is unlikely to provide such sharing among such queries. Intermediate results of the queries thus have to be materialized dynamicallyaccording to historical query information and the current system status.

Query evaluation for big data analytics with intermediate result materialization in a MEC is a new topic and poses many challenges. One challenge is to decide whether an intermediate result of a query should be materialized for the evaluation of

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

future queries, considering that if the materialized intermediate result will not be used again in future, the memory resource for it will be wasted. Another challenge is in which cloudlet(s) to store the materialized intermediate results, given the capacitated cloudlets. If the intermediate result is materialized in a cloudlet far from the locations of other queries using it, it may hardly be used due to the long evaluation response delay. The last challenge is how to meet the stringent QoS in terms of delay requirement of each query while minimizing its evaluation cost, by leveraging a fine-grained trade-off between the evaluation cost and the execution time of each query.

Although materialization has been studied extensively in relational databases [4], query evaluation for big data analytics in MEC is still in an early stage and has different characteristics to explore. There are several studies on query intermediate result materialization for big data in distributed cloud environments, such as Hadoop. Many of these studies however either assumed ideal distributed environments with abundant computing and storage resources, or did not consider query evaluation costs and delays. These methods thus cannot be directly applied to MEC. Most recently, the optimization of query evaluation for big data analytics has attracted the attention of many researchers, e.g., there are studies focusing on data placement and query evaluation for big data analytics [6], [9], [10], [12]–[14], [16], [18]. However, none of them considered the intermediate result materialization. To the best of our knowledge, we are the first to consider intermediate result materialization for big data analytic queries in mobile edge clouds. We aim to admit as many queries as possible, subject to resource constraints on cloudlets.

The major contributions of this paper are as follows. We first formulate the QoS-aware query evaluation for big data analytics with intermediate result materialization in a mobile edge cloud, by giving detailed models for intermediate result sharing, query evaluation costs, and query QoS requirements. We then propose an online algorithm that dynamically admits queries and materializes intermediate results, via an efficient online algorithm for reinforcement learning and a prediction mechanism. We finally evaluate the performance of the proposed algorithm against some benchmarks. Experimental results show that the performance of the proposed algorithm is promising, by achieving a higher system throughput while reducing the average evaluation cost per query by from 20% to 52% compared to the comparison benchmarks.

The remainder of the paper is arranged as follows. Section II will introduce the system model, notations and problem definition. Section III will propose an efficient online algorithm that dynamically admits queries. Section IV will provide some experimental results on the performance of the proposed algorithm. Section V will survey state-of-the-arts on this topic, and Section VI will conclude the paper.

II. Preliminary

A. System model

We consider a mobile edge cloud $G = (V, E)$ deployed as a Wireless Metropolitan Area Network (WMAN) within the proximity of mobile users, where V represents a set of *cloudlets* and E is a set of links that interconnect the cloudlets in V. Let v_i be a cloudlet in V. Each cloudlet v_i has a limited computing resource to evaluate queries. Denote by C_i the computing capacity of cloudlet v_i. Let e be a link in G. There is a delay for transmitting each unit data through link e. Users submit their big data queries to their nearby cloudlets. The queries however may be forwarded to the other cloudlets for processing, since their nearby cloudlets may not have sufficient computing resource for query processing or the processing cost at the local cloudlets may be too expensive. Fig. 1 gives an example of a mobile edge cloud.

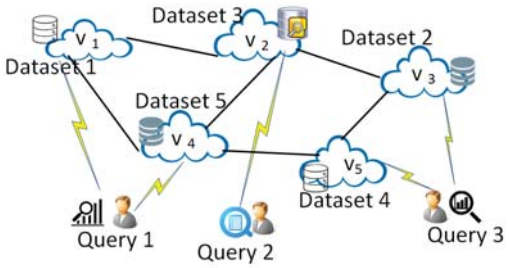


Fig. 1. The system model

B. Big data analytics and intermediate result sharing

Users frequently generate large volume of data from their mobile phones, web logs, click streams, Internet-of-Things (IoT) sensors, and many other sources. Due to the capacity and energy limitations of mobile devices, such large volume of data needs to be transferred to the edge cloudlets for continuous analysis. We refer to the data generated by a user as the *dataset* of the user, and the dataset is then stored in a nearby cloudlet of the user that is termed as its *home cloudlet*. Assuming that time is divided into equal time slots, let $\mathcal{S}_t$ be the set of datasets generated by all users in time slot t, denote by S_j a dataset in $\mathcal{S}_t$, where $1 \leq j \leq J$ with J representing the number of datasets in $\mathcal{S}_t$, i.e., $J = |\mathcal{S}_t|$. Let $v(S_j)$ be the cloudlet at which dataset S_j is stored.

Datasets stored in the edge cloudlet will be consumed by other users continuously. Each dataset S_j may be demanded by multiple queries for big data analytics. Denote by Q_t the set of queries that arrive in the system in time slot t. Each $q_m \in Q_t$ may demand multiple datasets. Its evaluation may need the intermediate results of the other queries if they share common sub-expressions. For example, $\mathcal{S}(q_m)$ is the set of datasets demanded by q_m. Let $I(S_j, q_{m'})$ be the intermediate result of evaluating query $q_{m'}$ on dataset S_j. If two queries q_m and $q_{m'}$ share a common sub-expression and the intermediate result $I(S_j, q_{m'})$ is materialized in a cloudlet already, then query q_m can be evaluated based on the materialized $I(S_j, q_{m'})$.

C. Query's Quality of Service

We consider big data applications with stringent timeliness requirements, considering that big data applications usually require in-time result delivery. If the query is not evaluated on time, the obtained results may no longer applicable. We refer

to *quality of service* (QoS) in terms of *delay requirement* of a query as the duration from the time point the query is issued to the time point the query result has been obtained at its home cloudlet. The delay experienced by query q_m thus is dominated by replicating each dataset S_j to a cloudlet for processing and transmitting the intermediate result from the cloudlet to its home cloudlet. Since the replication of the required multiple datasets of query q_m can proceed in parallel, we consider the delay experienced by q_m as the maximum delay incurred in replicating its original datasets intermediate results. For the datasets that have materialized intermediate results that q_m can leverage, the replication of those datasets will be saved. Denote by d_m the maximum tolerable delay of query q_m.

D. Cost model

The cost of evaluating a query consists of the storage cost, materialization cost, process cost and transmission cost, and each of these component costs is defined in the following.

Storage cost: The *storage cost* is for storing origin datasets in the system. Let $c_{s,i}$ be the storage cost of a unit volume of data in cloudlet $v_i \in V$. The storage cost for each dataset S_j in v_i thus is $|S_j| \cdot c_{s,i}$. Recall that $\mathcal{S}(q_m)$ is the set of datasets demanded by query q_m. The storage cost of q_m thus is

$$C_s(q_m) = \sum_{S_j \in \mathcal{S}(q_m)} \sum_{v_i \in V} x_{i,j,m} \cdot |S_j| \cdot c_{s,i}, \quad (1)$$

where $x_{i,j,m}$ is an indicator variable showing whether dataset S_j is replicated to cloudlet v_i for the evaluation of q_m.

Materialization cost: The cost of materializing intermediate results is due to the consumption of memory resource of cloudlets. Let $c_{m,i}$ be the cost of materializing a unit volume of intermediate result in the memory of cloudlet v_i. Assuming that α is the ratio between the volume of the intermediate result and the volume of original dataset S_j with α being a constant with $0 < \beta \leq 1$ [11], the materialization cost of query q_m is

$$C_m(q_m) = \sum_{S_j \in \mathcal{S}(q_m)} \sum_{v_i \in V} y_{i,j,m} \cdot |S_j| \cdot c_{m,i} \cdot \alpha, \quad (2)$$

where $y_{i,j,m}$ is an indicator variable that indicates whether its intermediate result $I(S_j, q_m)$ is materialized in the memory of cloudlet $v_i \in V$.

Process cost: The *process cost* is the cost for evaluating queries based on its required datasets, which is incurred by computing resource usage during query evaluation. Let $c_{p,i}$ be the cost of processing a unit volume of data in cloudlet v_i. Notice that there may be intermediate results that have been materialized in the system, the process cost of query q_m for the datasets with the intermediate results can be saved. Let $C_p(q_m)$ be the process cost of query q_m, which can be calculated by

$$C_p(q_m) = \sum_{S_j \in \mathcal{S}(q_m)} \sum_{v_i \in V} x_{i,j,m} \cdot |S_j| \cdot c_{p,i}. \quad (3)$$

Transmission cost: Denote by $h(q_m)$ the home cloudlet of query q_m. The *transmission cost* is the cost of data transfer within the mobile cloud network by transferring the intermediate results of each query q_m from the cloudlets where the queries are evaluated to the home cloudlet $h(q_m)$ of q_m. Denote by c_e be the cost of transmitting a unit volume of data along link $e \in E$. The cost of transmission for the evaluation of query q_m is

$$C_t(q_m) = \sum_{S_j \in \mathcal{S}(q_m)} \sum_{v_i \in V} \alpha \cdot |S_j| \cdot x_{i,j,m} \sum_{e \in p_{v_i,h(q_m)}} c_e, \quad (4)$$

where $p_{v_i,h(q_m)}$ is the shortest path between cloudlet v_i to the home cloudlet $h(q_m)$ of q_m in the mobile edge network G.

The **total cost** $C(q_m)$ of evaluating query q_m thus is

$$C(q_m) = C_s(q_m) + C_m(q_m) + C_p(q_m) + C_t(q_m). \quad (5)$$

E. Problem definition

Consider a finite time horizon consisting of T equal time slots, let Q_t be a set of big data analytic queries arrived at the beginning of each time slot t, the *dynamic QoS-aware query evaluation of big data analytics with intermediate result materialization* in a mobile edge cloud is to replicate the datasets of each query in Q_t at strategic cloudlets and decide whether to materialize the intermediate results of query evaluation, such that as many as queries in $\cup_{t=1}^{T} Q_t$ can be admitted while their total evaluation cost is minimized, subject to the capacity on each cloudlet v_i in V, while meeting the specified delay requirement d_m of each admitted query $q_m \in \cup_{t=1}^{T} Q_t$.

III. Online Algorithm

In this section, we first formulate the procedure of intermediate result materialization as a **Reinforcement Learning (RL)** process. We then propose an online algorithm for the dynamic QoS-aware query evaluation of big data analytics with intermediate result materialization. To ensure the online algorithm resilient to the dynamic evolution of the mobile edge cloud, we also propose an effective mechanism that can predict the future actions for further cost savings in query evaluation due to intermediate result materialization.

A. RL formulation

In a reinforcement learning process, a component called '*agent*' is designated to interact with the environment of the system. At each time slot t, the agent observes some state s_t of the system and makes the decision of what action a_t to take. Following action a_t, the state of the environment transitions from state s_t to a new state s_{t+1} and the agent receives a reward r_t. To model whether an intermediate result needs to be materialized, we here describe the objective (the reward function), state space, action space, and detailed procedure of the proposed RL formulation as follows.

Objective: If there is a materialized intermediate result for a query q_m, the query can make use of the intermediate result in the memory instead of re-evaluating it from its source datasets. Therefore, the objective of intermediate result materialization aims to save the processing cost of future queries. We use *the average saved evaluation cost* as the

objective of intermediate result materialization. Specifically, let $C_{mtl}(q_m, t)$ be the evaluation cost of query q_m whose intermediate results are materialized in time slot t. Denote by $C_{void}(q_m, t)$ the evaluation cost of q_m if no intermediate results will be materialized or de-materialized in time slot t. The objective of the RL procedure for intermediate result materialization at each time slot t thus is to maximize the cost savings due to intermediate result materialization, i.e.,

$$\max \sum\nolimits_{q_m \in Q_t} \big(C_{void}(q_m, t) - C_{mtl}(q_m, t)\big). \quad (6)$$

State space: The state of the system consists of currently materialized intermediate results and queries to be assigned to cloudlets. Let $\mathcal{I}_t$ be the set of existing materialized intermediate results. Recall that $\mathcal{S}(q_m)$ is the set of datasets required by each query $q_m \in Q_t$, $\mathcal{S}(q_m)$ may have newly generated datasets in $\mathcal{S}_t$ or existing datasets in the system. For simplicity, we use $\mathcal{S}_t(Q_t)$ to denote a set of 'distinct' datasets that are demanded by all queries in Q_t. Notice that there can be many datasets in the system, making the state representation indefinite over a long time. Intuitively, the most recent replicated datasets are more attractive to users. We thus maintain datasets in previous $\mathcal{T}$ time slots.

Action space: At each time slot, for each dataset S_j in $\mathcal{S}(q_m)$ and the set of queries requiring S_j, the agent needs to make decisions of (1) whether to keep the materialization of an existing intermediate result; and (2) whether to materialize a newly generated intermediate result. Thus the action taken can be modeled as $\{\emptyset, 0, 1\}$, where 0 indicates that the agent does not wish to materialize the corresponding intermediate result, 1 implies the materialization for the intermediate result generated from S_j and a common-subsequence of the queries demanding S_j, and $\emptyset$ means that no action is taken.

Having given the objective, state space, and action space of the RL procedure for intermediate result materialization, we now describe the detailed steps of the procedure. We maintain the *actual number* $MV_j(t)$ of materialized views for each dataset S_j at each time slot t. Denote by $\hat{MV}_j(t)$ the *predicted number* of materialized views for dataset S_j in time slot t. If the accumulative reward decreases more than a predefined threshold ϑ, we predict the action that should be taken in the current time slot. To this end, we first predict the number $\hat{MV}_j(t)$ of materialized views that should be created in the next time slot t, by making use of historic traces of the actual numbers of materialized views for each dataset in previous p time slots, based on an auto-regression moving average model. As the information about dataset S_j in previous p time slots is given, i.e., $MV_j(t-1)$, $MV_j(t-2)$, ..., $MV_j(t-p)$ are given, $\hat{MV}_j(t)$ can be predicted by

$$\begin{aligned} \hat{MV}_j(t) \;=\; & a_1 \cdot MV_j(t-1) + a_2 \cdot MV_j(t-2) \\ & + \ldots + a_p \cdot MV_j(t-p), \end{aligned} \quad (7)$$

where $a_{p'}$ is a constant with $0 \le a_{p'} \le 1$, $\sum_{p'=1}^{p} a_{p'} = 1$, and $a_{p_1} \ge a_{p_2}$ if $p_1 < p_2$.

Denote by $Q_t(S_j)$ the set of queries that demand dataset S_j, and let $N_t(S_j)$ be the number of common sub-expressions of the queries in $Q_t(S_j)$. Obviously, it is likely to create a materialized view for each common sub-expression of all the queries that share the common sub-expression. We thus determine the number $MV_j(t)$ of materialized views for each dataset S_j after time slot t as

$$MV_j(t) = \min\{\hat{MV}_j(t), N_t(S_j)\}. \quad (8)$$

Given the number $MV_j(t)$ of materialized views that should be created for dataset S_j in time slot t, we rank the set of common sub-expressions in a non-increasing order according to the number of queries sharing the common sub-expression. We then create a materialized view for each of the common-subexpressions until $MV_j(t)$ materialized views for dataset S_j are created. The RL procedure with predictions is shown in algorithm 1.

Algorithm 1 RL_Proc_Predict

Input: The set Q of queries; the set $Q_t(S_j)$ of queries demanding dataset S_j; the number $N_t(S_j)$ of common sub-expressions of queries in $Q_t(S_j)$; the maintained actual number $MV_j(t-p)$ of materialized views for each dataset S_j for the previous p time slots with $p \ge 2$.

Output: The materialization decisions in memory for the existing and newly generated intermediate results of each dataset S_j.

1: $\Delta \leftarrow C_{void}(q_m, t-2) - C_{mtl}(q_m, t-1)$; /* the reward gain in time slot $t-1$/*
2: **if** $\Delta > \vartheta$ **then**
3: Predict the number $\hat{MV}_j(t)$ of materialized views that should be created for each dataset S_j in time slot t by Eq. (7);
4: **if** $\hat{MV}_j(t) > N_t(S_j)$ **then**
5: Create a materialized view for the intermediate result of each of the common sub-expressions;
6: **else**
7: Rank the set of common sub-expressions in a non-increasing order according to the number of queries sharing the common sub-expression;
8: Create a materialized view for each of the common-subexpression until there are $\hat{MV}_j(t)$ materialized views for dataset S_j;
9: **return** The materialization decisions in memory for both existing and newly created intermediate results in the current time slot t of each dataset S_j;

B. Query admission procedure

Given the materialization decisions for both existing and new intermediate results of each dataset S_j, we admit queries in each time slot t. Specifically, to allow the materialized intermediate results being fully utilized by evaluating queries, the query admission procedure proceeds iteratively.

Within each iteration k, we find a subset of *distinct queries* demanding some datasets, these distinct queries are different from each other, and the demanded datasets by the distinct queries are different from each other, too. Let $\hat{q}_m$ and $\hat{S}_j$ be a distinct query and a dataset demanded by the query. Each pair of a distinct query $\hat{q}_m$ in the found subset and a different dataset $\hat{S}_j$ is considered together as a compound item during the query evaluation. Namely, within each iteration, we reduce the problem of assigning queries to cloudlets to a minimum weight maximum matching problem in an auxiliary bipartite graph $G'_k = (V'_{a,k} \cup V'_{b,k}, E'_k)$, where the vertex set of the bipartite graph is divided into two disjoint subsets: $V'_{a,k}$ consists of distinct queries in Q_t and their demanded datasets,

and $V'_{b,k}$ is the cloudlets of the mobile edge cloud. If the delay requirement of query $\hat{q}_m$ in $V'_{a,k}$ can be met by processing the dataset demanded by $\hat{q}_m$ at the cloudlet node in $V'_{b,k}$, and the computing capacity constraint of the cloudlet is not violated, there would be an edge between each node in $V'_{a,k}$ and each node in $V'_{b,k}$. The weight of the edge is the total evaluation cost if there is no any materialized views for a dataset; otherwise the weight is set as the sum of storage cost, materialization cost and the transmission cost.

Fig. 2 is an illustrative example of reducing a query admission to the minimum-weight maximum matching problem in the defined auxiliary bipartite graph, where $Q_t = \{q_1, q_2, q_3\}$, query q_1 demands datasets S_1, S_2, S_3 and S_4; query q_2 demands datasets S_2, S_3 and S_4; query q_3 demands datasets S_2 and S_6; i.e., $\mathcal{S}(q_1) = \{S_1, S_2, S_3, S_4\}$, $\mathcal{S}(q_2) = \{S_2, S_3, S_4\}$, $\mathcal{S}(q_3) = \{S_2, S_6\}$. The mobile edge cloud G consisting of 5 cloudlets, v_1, v_2, v_3, v_4 and v_5.

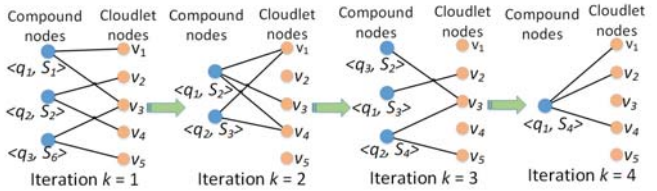


Fig. 2. The query admission by minimum weight maximum matching in an auxiliary bipartite graph

C. Online algorithm

Given the RL procedure for intermediate result materialization and query admission in each time slot, we now propose an online algorithm for the dynamic QoS-aware query evaluation of big data analytics with intermediate result materialization.

The idea of the algorithm is to make the materialization decision at the beginning of each time slot t, according to the current system status and historical information by applying algorithm 1. A weighted bipartite graph $G'_k = (V'_{a,k} \cup V'_{b,k}, E'_k)$ then is constructed based on the materialization decision. A minimum weight maximum matching in G'_k finally is found, which corresponds an assignment of queries. The steps of the proposed online algorithm is elaborated in algorithm 2. For simplicity, we refer to the algorithm 2 as `RL-Online`.

IV. Simulations

A. Parameter settings

We consider a mobile edge cloud consisting of 50 cloudlets in different GT-ITM generated network topologies [1], and there is an edge between each pair of cloudlets with a probability of 0.2. The computing capacities of each cloudlet are randomly drawn from $[8, 16]$ GHz [15]. The processing cost, the storage cost, the materialization cost and the transmission for a gigabyte of data are values in ranges of $[15, 25]$ cents, $[0.4, 0.6]$ cents, $[15, 25]$cents, and $[0.1, 1.8]$cents, respectively. The delay for transferring a unit of data is from 0.01 to 0.06 ms, and the QoS in terms of delay requirement of each query bounds from 0.1 to 1 ms.

Algorithm 2 `RL-Online`

Input: A mobile edge cloud $G = (V, E)$, a finite horizon T, the set $\mathcal{S}_t$ of datasets at time slot $t \in T$, the set Q_t of queries at each time slot t, the set $\mathcal{S}(q_m)$ of datasets demanded by query $q_m \in Q_t$ at each time slot t, the set $Q_t(S_j)$ of queries that demand dataset $S_j \in \mathcal{S}_t$ at each time slot t, the delay requirement d_{q_m} of each query q_m.

Output: The replication locations of datasets in $\mathcal{S}_t$, the assignment of queries in Q_t to cloudlets for evaluation.

for $t \leftarrow 1, \cdots, T$ **do**
  Revoke algorithm 1 to obtain the materialization decisions for datasets;
  $k \leftarrow 0$; /* iteration index*/;
  while true **do**
    $k \leftarrow k + 1$;
    $Num^k_{ad} \leftarrow 0$; /* The number of admitted queries in iteration k*/;
    Find a subset $\cup_{q_m \in Q_t} \mathcal{S}(q_m)$ of datasets that are demanded by queries in Q_t;
    Within $\cup_{q_m \in Q_t} \mathcal{S}(q_m)$, find the maximum subset $\mathcal{S}'_{t,k}$ consisting of datasets $\hat{S}_j$ that are demanded by 'distinct' queries $\hat{S}_j \in \mathcal{S}'_{t,k}$, i.e., $\cap_{\hat{S}_j \in \mathcal{S}'_{t,k}} \hat{S}_j = \emptyset$ *or* $\hat{S}_j$ and $\cap_{\hat{q}_m \in Q_t} \hat{q}_m = \emptyset$ *or* $\hat{q}_m$;
    if $\mathcal{S}'_{t,k} = \emptyset$ **then**
      Return;
    Construct the bipartite graph $G'_k = (V'_{a,k} \cup V'_{b,k}, E'_k)$, by considering each dataset $\hat{S}_j \in \mathcal{S}'_{t,k}$ and the distinct query $\hat{q}_m$ demanding it as a compound node, this compound node is added into G'_k as $V'_{a,k}$, and all cloudlets are added into $V'_{b,k}$;
    if $|V'_{a,k}| > |V|$ **then**
      Move $|V'_{a,k}| - |V|$ compound nodes to next iteration to assign;
    Connect each node in $V'_{a,k}$ and each node in $V'_{b,k}$, if the delay requirement of query $\hat{q}_m$ in $V'_{a,k}$ can be met by processing the dataset demanded by $\hat{q}_m$ at the cloudlet node in $V'_{b,k}$, and the capacity of cloudlet is not violated;
    Set the weights of edges in G'_k, i.e., the weight of an edge is the total evaluation cost if there is no any materialized views for a dataset; otherwise the weight is set as the sum of storage cost, materialization cost and the transmission cost;
    Find a minimum weight maximum matching M in G'_k;
    for each matched edge $e' \in M$ **do**
      Replicate the dataset S_j of the compound node of e' to the cloudlet node of e', if S_j or its materialized view is not at the cloudlet;
      if All the datasets demanded by a query q_m have been processed **then**
        Assign query q_m to cloudlet v_i;
        $Num^k_{ad} \leftarrow Num^k_{ad} + 1$; $Q_t \leftarrow Q_t \setminus \{q_m\}$;
      else
        Mark S_j that is demanded by q_m as processed;
    if $Num^k_{ad} = 0$ **then**
      break;
return The replications of datasets, the assignment of queries in Q_t and the materialization decisions;

A monitoring period T is set 50 time slots, and each time slot is 0.5ms. The volume of each dataset generated by a user is randomly drawn from the range of $[1, 6]$ GB [15], and the amount of computing resource assigned to the processing of 1GB data is a value between $[0.75, 1.25]$ GHz [1]. The numbers of datasets and queries in the system are randomly drawn in $[5, 20]$ and $[100, 600]$, respectively. The number of datasets demanded by each query are randomly drawn from interval $[1, 7]$. Unless otherwise specified, we will adopt these default settings in our experiments. Each value in the figures is the mean of the results by applying each mentioned algorithm on 15 different topologies of the mobile edge cloud.

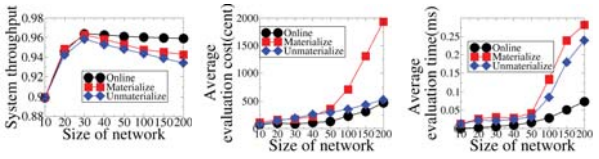
We evaluate the proposed algorithm `RL-Online` against the following benchmarks. The first benchmark admits queries

[1]Amazon EC2 and Amazon S3

one by one without considering the materializations of any intermediate results. Specifically, the queries are firstly ranked in an increasing order of the total volume of their demanded datasets. For each of the queries in the ordered list, the benchmark algorithm finds the cloudlet that achieves the minimum evaluation cost for it. For simplicity, this benchmark algorithm is referred to as algorithm `Unmaterialized`. Furthermore, we consider another variation of the mentioned benchmark algorithm, which always materializes any intermediate result generated during the query evaluation. This variation of the benchmark algorithm is referred to as `Materialized`.

B. Performance evaluation

We first investigate the performance of algorithm `RL-Online` against algorithms `Unmaterialized` and `Materialized`, in a time horizon with 50 time slots, by varying the number of cloudlets, i.e., the size of network, from 10 to 200.

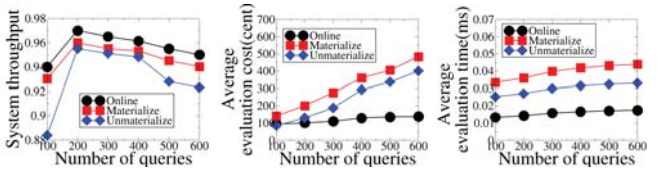


(a) System throughput (b) Average evaluation cost (c) Average evaluation time

Fig. 3. The performance of various algorithms `RL-Online`, `Materialized`, and `Unmaterialized`.

Fig. 3 shows the results in terms of system throughput, the average evaluation cost and the average evaluation time for each admitted query. From the figures, we can see that our proposed algorithm `RL-Online` admits more queries but consuming less average evaluation cost and evaluation time than its counterparts, with the growth of cloudlet number. For example, although algorithm `RL-Online` admits about 2% and 4% more queries than that of algorithm `Unmaterialized` and `Materialized`, while the average evaluation cost decreases 20% and 52% respectively, when there are 100 cloudlets. The reason is that algorithm `RL-Online` concurrently schedules the queries for processing instead of processing queries one by one as the benchmarks do, by learning the best actions for intermediate result materialization via reinforcement learning. Also, `RL-Online` adopts selective intermediate result materialization, which can maximumly re-use the materialized views while saving processing cost; however, algorithm `Materialized` always materializes any generated intermediate result, which can cause significant resource wastage thereby increasing the evaluation cost and time, while reducing the number of queries that can be admitted, due to the limited cloudlet capacities; algorithm `Unmaterialized` neglects the re-use of all intermediate results by processing all datasets repeatedly, even when some queries share several subexpressions, thereby wasting many resources and resulting in high evaluation cost, as well as high average evaluation time. From Fig. 3 (a), we can also see that the system throughput increases first when the number of cloudlets increases from 10 to 35, and then decreases after 35. The rationale behind is that when the network size is small, its resources are very limited, leading to the queries being rejected in a high probability. However, when its size becomes large, the queries can still be rejected due to long response delay which easily violates the delay requirements of queries. The average evaluation times of the mentioned algorithms is shown in Fig. 3 (c).

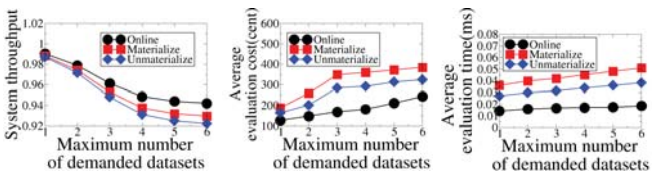
C. Impact of important parameters on algorithmic performance



(a) System throughput (b) Average evaluation cost (c) Average evaluation time

Fig. 4. The impact of number of queries on the performance by various algorithms.

We then investigate the impact of various numbers of queries on the performance of different algorithms by varying the queries' number from 100 to 600. The performance results are shown in Fig. 4. We can see from Fig. 4(a) that the system throughput first increases, and then reaches the peak when the number of queries is 200, followed by a decreases after 200. The rationale behind is that when the queries' number is small, the number of materialized views in the system is small, the system has to process all demanded datasets of the queries separately, which may violate the stringent delay requirements of queries, leading to low system throughput. With the growth of the queries' number, more queries arrive at the system, the system needs to consume more resources to evaluate the arrived queries. However, due to the limited computing resource of cloudlets and the stringent delay requirements of queries, more queries tend to be rejected by the system, which decreases the system throughput. From Fig. 4(b), it can be seen that the average evaluation cost of each query by the three algorithms increases with the growth of query number. This is because the transmission cost, process cost are significantly increasing as more queries are processed in the system. In addition, more views of intermediate results are materialized at memories of cloudlets, leading to higher materialization cost. We also can see from Fig. 4(c) that the average evaluation time of different algorithms grows with the increase on the number of queries, the reason is that it takes more time to evaluate more queries and process the increasing intermediate results.



(a) System throughput (b) Average evaluation cost (c) Average evaluation time

Fig. 5. The impact of maximum number of datasets demanded by each query on performance by various algorithms.

We finally study the impact of the maximum number of datasets demanded by each query on the performance of different algorithms, by varying the number from 1 to 6. The evaluation results are illustrated in Fig. 5. From which it can be seen that with the growth of the maximum number of demanded datasets, the system throughput decreases, the average evaluation cost and the average evaluation time increase. The reason is that when a query demands more datasets at the same time, the QoS in terms of delay requirement of the query is prone to be violated, so the system throughput decreases. Meanwhile, the system needs more time to process all the datasets demanded by each query and the intermediate results generated by the datasets, thereby increasing the average evaluation cost and evaluation time, as illustrated in Fig. 5(b) and Fig. 5(c).

V. Related work

Although view materialization has been studied extensively in relational databases [4], query evaluation for big data analytics in mobile edge clouds is still in an early stage and has different characteristics to explore. There are several studies on query intermediate result materialization for big data in distributed cloud environments, such as Hadoop. For example, ReStore [2] materializes intermediate results of Map-Reduce jobs for reuse in future queries. Perez *et. al.* [7] investigated salient features of SQL-on-Hadoop systems including immutable data, abundant storage to accommodate materialized views, and excessive materialization of intermediate results that enables generating materialized views as a by-product of answering queries. Many of these studies however either assumed ideal distributed environments with abundant computing and storage resources, or did not consider query evaluation costs and delays. Those methods thus cannot be directly applied to the intermediate result materialization in mobile edge clouds. Most recently, the optimization of query evaluation for big data analytics has attracted the attention of many researchers. For example, there are studies focusing on data placement and query evaluation for big data analytics [6], [9], [10], [12]–[14], [16], [18]. However, none of them considered the intermediate result materialization. To the best of our knowledge, we are the first to consider intermediate result materialization for big data analytic queries in mobile edge clouds. We aim to admit as many queries as possible, subject to resource constraints on cloudlets.

VI. Conclusions

In this paper, we studied the problem of dynamic QoS-aware query evaluation for big data analytics with intermediate result materializations in a mobile edge cloud within a finite time horizon. The objective is to maximize the system throughput while minimizing the average evaluation cost of admitted queries, subject to the resource capacity on each cloudlet. To this end, we proposed an online algorithm for the problem by jointly considering query admissions and materializations decisions, through the adoption of Reinforcement Learning with predictions. We evaluated the performance of our proposed algorithm against some benchmark algorithms by simulations. Experimental results show that the performance of the algorithm is promising, which can achieve a higher system throughput while reducing the evaluation cost per query by from 20% to 52% on average, compared to the comparison benchmarks.

Acknowledgments

The work of Qiufen Xia, Zichuan Xu, and Guowei Wu is partially supported by the National Natural Science Foundation of China (Grant No. 61802047, 61802048, 61772113, 61872053), the fundamental research funds for the central universities in China (Grant No. DUT19RC(4)035, DUT19GJ204), and the "Xinghai Scholar Program" in Dalian University of Technology, China.

References

[1] K. Calvert, and E. Zegura. Gt-itm: georgia tech internetwork topology models

[2] I. Elghandour and A. Aboulnaga. ReStore: Reusing Results of MapReduce Jobs. *The Proceedings of the VLDB Endowment* . Vol. 5, No. 6, pp.586597, 2012.

[3] W. Fan, F. Geerts, and F. Neven. Making queries tractable on big data with preprocessing: through the eyes of complexity theory. Proceedings of the VLDB Endowment, Vol.6, No. 9, pp.685-696, 2013.

[4] J. Goldstein and P.-A. Larson. Optimizing queries using materialized views: a practical, scalable solution. SIGMOD Rec., Vol. 30, No. 2, pp.331-342, ACM, 2001.

[5] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, Vol. 10, No. 4, pp. 270-294, 2001.

[6] B. Heintz, A. Chandra, and R. K. Sitaraman. Trading timeliness and accuracy in geo-distributed streaming analytics. *Proc. of SoCC*, ACM, 2016.

[7] L. Perez and C. Jermaine. History-aware query optimization with materialized intermediate views. *Proc. of ICDE*, IEEE, 2014.

[8] P. Li , S. Guo, T, Miyazaki, X. Liao, H. Jin, A. Y. Zomaya, and K. Wang. Traffic-aware geo-distributed big data analytics with predictable job completion time. *IEEE Trans. on Parallel and Distributed Systems*, Vol.28, No.6, pp.1785-1796, 2017.

[9] H. Li, H. Xu, and S. Nutanong. Bohr: similarity aware geo-distributed data analytics. *Open Access Media*, USENIX, 2017.

[10] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica. Low latency analytics of geo-distributed data in the wide area. *Proc. of SIGCOMM*, ACM, 2015.

[11] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves. Sailfish: a framework for large scale data processing. *Proc. of SoCC*, ACM, 2012.

[12] Q. Xia, L. Bai, W. Liang, Z. Xu, L. Yao, and L. Wang. QoS-Aware Proactive Data Replication for Big Data Analytics in Edge Clouds. *Proc. of ICPP*, SRMPDS, 2019.

[13] W. Xiao, W. Bao, X. Zhu, and L. Liu. Cost-aware big data processing across geo-distributed datacenters. *IEEE Trans. on Parallel and Distributed Systems*, Vol.28, No.11, pp.3114-3127, 2017.

[14] Q. Xia, W. Liang, and Z. Xu. Data locality-aware big data query evaluation in distributed clouds *Computer Journal*, Vol.60, No.6, pp.791-809, 2017.

[15] Q. Xia, Z. Xu, W. Liang, and A. Zomaya. Collaboration- and fairness-aware big data management in distributed clouds. *IEEE Trans. on Parallel and Distributed Systems*, Vol.27, No.7, pp.1941-1953, 2016.

[16] Q. Xia, Z. Xu, W. Liang, S. Yu, S. Guo, and A. Y. Zomaya. Efficient Data Placement and Replication for QoS-Aware Approximate Query Evaluation of Big Data Analytics. *Trans. on Parallel and Distributed Systems*, DOI: 10.1109/TPDS.2019.2921337, 2019.

[17] Y. Yan, L. J. Chen, and Z. Zhang. Error-bounded sampling for analytics on big sparse data. *Proc. of the VLDB Endowment*, Vol.7, No.13, pp.1508-1519, 2014.

[18] S. Yu, M. Liu, W. Dou, X. Liu, and S. Zhou. Networking for big data: a survey. *IEEE Communications Surveys & Tutorials*, Vol.19, No.1, pp.531-549, 2017.