Peer Data Caching Algorithms in Large-Scale High-Mobility Pervasive Edge Computing Environments

Yaodong Huang, Fan Ye, and Yuanyuan Yang
Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA
{yaodong.huang, fan.ye, yuanyuan.yang}@stonybook.edu

***Abstract*—Emerging innovative edge devices like drones, self-driving cars, phones/tablets and IoT nodes are revolutionizing our daily lives. Caching data among peer edge devices enables data sharing needed in many applications. In such applications, network scalability and node mobility bring many challenges. They change the topology and the resources in the network and make the network less robust. In this paper, we propose peer data caching strategies that consider the scale and mobility of these increasingly popular edge devices. We propose a grouping method creating a layered design to reduce the number of entities in each layer. We propose inter-group and intra-group optimization problems which proactively cache data onto best places to support robust and fast data access. We develop a 7-approximation algorithm for inter-group optimization and use uncapacitated facility location problems to solve intra-group optimization. We also transform the mobility of nodes into node behaviors to reduce the impact of mobility on the network. Our extensive simulation results show that our proposed strategies can apply to large-size and high-mobility networks, while achieving satisfactory results for data access.**

I. Introduction

Smart *edge devices* possessing powerful computational capabilities, rich sensing modalities, and fast data transmission radios are becoming pervasive in our daily lives. They are expected to accomplish unprecedented tasks such as intelligent drone cruise, self-driving, and personalized health management. Many such applications produce massive amounts of sensing data and require the sharing of data among peer devices.

Consider some situations where the peer data sharing provides rich information and the foundation of various applications. On a campus, smartphones carried by students and faculty generate sensing data; cars that drive on campus collect road and traffic information. When shared with peer devices, such data can inform people about activities and emergencies, help drivers avoid traffic and find parking spaces. In a large mall, shoppers can share their locations with merchants to obtain the latest coupons for discounts. In peer data sharing, caching helps nodes to access high demand valuable data. For example, cached traffic accident photos and video clips help keep nearby drivers aware of the traffic conditions, and cached activity information may help authorities to manage the population flow to avoid overcrowded public spaces.

Previous work [1], [2] have studied data caching in edge environments, usually confined at a relatively small scale and largely static topology (e.g., people staying in a corner of a square or an atrium). Peer caching under the large number and the high mobility of nodes is needed for many applications (e.g., pedestrians, vehicles across multiple city blocks). Meanwhile, the scale and mobility in edge scenarios bring many challenges. First, when the number of nodes grows, the latency to obtain input information from all these nodes becomes too high. By the time all the information is collected, topology changes may have made the information obsolete. Second, the computational power of the network may not be sufficient to analyze all the information accurately. Computational overhead often grows more than linearly with the number of nodes. If the network size is large, it takes too long to compute the caching decisions of which nodes should cache which data. Naively collecting the information and conducting computation more frequently will only lead to much increased overhead.

To enable data caching under high mobility and large scale in edge computing, we propose a 2-tier hierarchical grouping method, which organizes the network into multiple groups. We first find the best group where a data chunk should be cached (inter-group data caching), and then find the best nodes in this group to cache the data chunks (intra-group data caching). Compared with the traditional "flat" structure, it has several advantages. First, in the multi-layer structure, nodes form a group, and groups form the network. This hierarchy decreases the number of entities that must be managed in each layer, and the operation in each layer can be made transparent to each other, both leading to better scalability. Second, it also reduces the number of operations needed when nodes move and the network topology changes. This helps to transform the mobility issue to certain behaviors of nodes. When such behavior occurs, the corresponding processes will be triggered. Finally, we proactively cache data onto the best places based on the current situation, where the demanding groups can have quick and robust access to the corresponding data.

Specifically, we propose an approximation algorithm for inter-group optimization problems, and utilize an existing algorithm for intra-group optimization. We propose algorithms to transform mobility issue to simple node behaviors to alleviate the impact on nodes. Our simulation shows that our proposed strategies can adapt to different scales of network and different levels of mobility, while keeping the accessing time for data low and successful ratio for delivering data high.

We make the following contributions in this paper.

- We introduce scalable caching in high-mobility edge computing environments. The system is effective in the large scale and frequent mobility scenarios, offering fast data access and fair caching. We use a novel grouping method to reduce the computation and communication overhead to address network scale and node mobility.

978-1-5386-6808-5/18/$31.00

- We formulate the inter-group data caching optimization problem and propose an approximation algorithm to solve it. We prove that the approximation ratio is 7. We also propose solutions to the intra-group data caching problem.
- We propose node behavior problems to transform the mobility of nodes to certain node behaviors. We then design the corresponding algorithms that the node behavior will trigger.
- We implement our design in a network simulator, and extensive simulations show that our design can adapt to different node mobility and network scale, while achieving short latency for data access.

The rest of this paper is organized as follows. In Section II we discuss some related work on edge computing, cooperative caching and facility location problems. In Section III we discuss the model of our designed system. In Section IV we give the problem formulation and algorithms of our design. We evaluate the proposed algorithms and methods in Section V. Finally, we conclude this paper in Section VI.

II. Related Work

Edge computing [3] brings computing, storage services in close proximity to mobile users. It saves resource (battery, storage, data traffic) consumptions of mobile devices and may provide services at a quality similar to cloud-based ones. Peer data sharing among devices enables peer devices to share sensing, control data to create many novel applications. Our previous work propose peer data discovery [2] focusing on how to discover and retrieve data, and fair caching in peer data sharing [4] addressing how to ensure fair and efficient caching among peer devices of limited resources. These previous work establish the foundation of peer data sharing. To adapt to new application scenarios, we further study how to adapt caching decisions to network scale and node mobility.

The mobility of nodes in edge computing scenarios has already drawn much attention. Some work focuses on finding the optimal placement of resources to adapt to mobility patterns. Shiqiang Wang et al. [5] study the service placement for Mobile Micro-Clouds to support the coexistence of users and service instances. The solutions are based on the predictions in a certain look-ahead window, and they perform data migration dynamically on edge clouds to support service delivery. M. Reza Rahimi et al. [6] propose a framework to model node mobility and propose an optimal service allocation algorithm called *MuSIC* to support large numbers of nodes. Lin Wang et al. [7] study the online resource allocation for edge clouds where the user mobility is arbitrary. They propose online algorithms to put resources where user mobility and resources are not known. Some work using location-based data storage so that the node can have access to data based on their locations. Patrick Stuedi et al. [8] propose *WhereStore* to store data based on locations. It finds the pattern of device locations to distribute resources among smartphones and clouds.

Network scale has also attracted attention in edge computing. Multi-tier and hierarchical approaches provide easy network expansion and improve resource utilization. Liang Tong et al. [9] propose a hierarchical approach that places the load at different tiers of the network to minimize the delay for mobile devices. Jose Oscar Fajardo et al. [10] combine client-driven adaption and network-assisted adaption to achieve multimedia delivery. The client-driven adaption places the content close to the user, and the network-assisted adaption improves the quality of media in highly dynamic channels.

The optimal resource placement problem can often be mapped to the classical Facility Location (FL) problem. Most work maps their problems into different kinds of FL problems or modifies related FL problems to solve resource placement. Usually, either Uncapacitated Facility Location (UFL) problem [11] or rent-or-buy problem [12] is used. The more general case for these two problems are Connected Facility Location (ConFL) problems [13]. Among them, UFL does not consider the content dissemination costs, while rent-or-buy problem does not consider the facility building costs in ConFL problem. In this paper, we will reference mostly to the UFL problem and its solutions. The current best solution that we find is proposed by Shi Li [14], where they obtain a 1.488-approximation algorithm.

III. System Design

We now discuss the model, methodology and decision process of our proposed system.

A. System Model

Let graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected undirected graph representing the network topology for a multi-hop wireless network, where vertex set ($\mathcal{V}$) represents the nodes in the network, and edge set ($\mathcal{E}$) represents the connection links between nodes. Nodes will move inside the network, and connection links will break and reconnect between different nodes. Data will be generated continuously, and some nodes will request specific data. To achieve fast and robust access, we will first select some nodes to proactively cache data. These nodes will obtain data from the data producer. This phase is called *dissemination phase*. Then the requesting nodes will access data from these caching nodes, which is called *accessing phase*. We present the detailed design next.

B. Grouping Method

Maintaining network over different scales and nodes mobility is crucial in peer data sharing applications. Networks with smaller numbers of nodes are easier to coordinate because limited amounts of data need to be transmitted to compute caching decisions. For a large flat network, the data transmitted would require huge computation and transmission overheads, which a peer-based network cannot afford. Thus, we propose a hierarchical grouping method, which divides the network into multiple, connected and smaller groups each consisting of multiple nodes. The number of groups and the number of nodes in each group are both limited for easier management. The design adapts to scalability and mobility by forming and dissolving groups.

When proactively cache data into the network, the network will first decide which data chunk [1] to distribute onto which groups. Then the group that receives the data distribution decides which nodes in the group will cache the data chunk. These two decisions are made independently.

[1]One data item consists of multiple data chunks of the same size; this uniform size makes the design simpler [4].

1) Optimal data placement: Caching data in optimal places can provide fast and reliable data access, critical for satisfactory user experience. First, the network decides the optimal groups to cache the data chunk. This problem is **inter-group (data caching) optimization problem**. The goal is to find groups both near data requesters and low cost for caching, such that every data request can be satisfied, and each group does not store more data than its capacity.

After such a group is chosen, this group will decide the optimal nodes to cache the chunk, which is the **intra-group (data caching) optimization problem**. The goal is to ensure both fast data access and fair caching loads among mostly individually owned nodes, which are crucial problems in edge computing [4].

2) Node behavior under mobility: Node mobility can cause many problems such as transmission losses and topology changes. Ensuring service quality while network properties are changing is challenging. Grouping method brings a solution to minimize the impact of mobility on the network. In our design, every node must belong to one and only one group, and the mobility of a node may change the group which it belongs to. The unpredicted mobility of nodes is classified into two simple node behaviors: leaving a group or joining a group. Upon nodes leaving/joining a group, such node behaviors trigger the corresponding processes to adjust data caching, so as to maintain inter-group optimality despite minor violations. **Node behavior problem** discusses conditions and the corresponding triggered processes under the mobility of nodes.

3) Group maintenance: Over time, nodes may join or leave a group, varying group sizes and resources. Some groups may grow too large, making it hard to compute efficient data caching decisions. Other groups may shrink to only several nodes with little resources. Neither produces sufficient benefits to offset the cost of maintaining them. Thus a group may need to be split or dissolved under such conditions. When to split or dissolve groups is the **group maintenance problem**. We have some basic design and preliminary results on conditions and operations [2] for groups to dissolve and split. Due to space limitations, we will not elaborate on the detail design in this paper.

C. Decision Making Procedure

The procedure of making caching decisions is as follows. Periodically, the node mobility and the connection information are collected. Then, whether group maintenance actions are needed is decided and conducted if so. Next, if a new data request for a data chunk appears, the inter-group optimization problem is solved to find the optimal group. If there is no new request, this process will be skipped. Then intra-group optimization is invoked to find the optimal nodes in each chosen group. Finally, the mobility of nodes is detected, and the corresponding process will be triggered. For easy presentation, we first consider the centralized algorithm and assume that there is a centralized server, which holds and processes all the information, responsible for sending the decisions to corresponding groups/nodes.

[2]We consider communication and computational costs for a group to maintain itself. If they exceed certain thresholds, the group will split or dissolve. Splitting makes two new groups. Dissolving removes a group and nodes in the group will join other groups.

We make the following settings and assumptions for the application scenarios.

1) Routing and accessing cost: We use the expected transmission count (ETX) model as the routing strategies and accessing cost metric [15]. It finds the least contentious route in multi-hop wireless networks to ensure reliable and fast data transmission.

2) Data dissemination: Once groups, and eventually nodes receive instructions to cache data, they will fetch data using information (e.g., producers or previous cached nodes of the data chunk) in instructions.

3) Network scale: This work focuses on peer sharing situations much beyond a local wireless transmission neighborhood, e.g., at campus and mall scale. We do not consider scales beyond such sizes, such as a whole city, which will most likely require the combination of peer and cloud assisted sharing.

IV. Problem Formulations and Algorithms

A. Inter-group Data Caching Optimization Problem

Inter-group optimization problem focuses on how to distribute data onto groups to make sure that 1) data are placed near to requesting nodes for quick access, and 2) the cost for data caching is low. The goal of the optimization is to minimize both the cost of data access and cost for fetching data. The cost of data access κ_{ijn} is the cost of a demand interest j to get data from group i which will cache data n. For the cost of data access κ_{ijn}, we use the obtained expected transmission count (ETX) between i and j. The cost of fetching data ϕ_{in} is the cost of group i to get data n from the original data source, which is the ETX between i and the data source node. The κ_{ijn} and ϕ_{in} are known after analyzing the information of ETX probes. We formulate this problem as an integer linear programming problem.

$$\min \quad \sum_i \sum_j \sum_n \kappa_{ijn} a_{ijn} + \sum_i \sum_n \phi_{in} g_{in} \tag{1}$$

$$\text{s.t.} \quad \sum_n q_n g_{in} \leqslant l_i \ \ (\forall j), \tag{2}$$

$$\sum_i a_{ijn} \geqslant r_{jn} \ \ (\forall i, \forall n), \tag{3}$$

$$a_{ijn} \leqslant g_{in} \ \ (\forall i, \forall j, \forall n), \tag{4}$$

$$a_{ijn}, g_{in} \in \{0, 1\}, \tag{5}$$

$$r_{jn} \in \{0, 1\}, \tag{6}$$

$$l_j, \kappa_{ijn}, \phi_{in} \geqslant 0. \tag{7}$$

In the problem formulation, a_{ijn} and g_{in} are assignment variables. $a_{ijn} = 1$ means group j that demands data n will access it from group i. $g_{in} = 1$ means group i will cache data n. Constraint (2) ensures the cache will not exceed the total storage of group l_i. q_n is the size of data n. In real situations, since the nodes may leave or join a group, new data may be carried into a group before another round of inter-group optimization. Thus, we will leave some vacant storage for caching in case of nodes joining and bring new data. We set l_i smaller than the total

storage of the group. Constraint (3) makes sure that all demands for data will be satisfied. $r_{j,n}$ indicates group j demands data n. If a group j needs to access data n, there must be at least one data access assignment. Constraint (4) ensures that if data n on group i can be accessed from another group j, data n must be cached on group i. Constraints (5) – (7) are the ranges for every variable. Note that if group i caches a data chunk that other groups demand, the data chunk will be marked with a "loan number" equal the number of those groups requesting from group i. The notations are presented in Table I.

This problem has significant differences from the current data or facility placement problems. We introduce the data chunk ID n and place all data chunks simultaneously, rather than placing data items one by one. When the algorithm runs continuously over time, many data chunks already exist in the network. When inter-group optimization problem needs to be reconducted, especially when the number of groups changed due to the group maintenance problem or when new data chunks generated in the network, it is not practical to find a reasonable order to caching data chunk one by one.

This inter-group optimization problem is NP-hard. We present a very simple proof here. First, we make one special case. Assume that there is one only chunk n_0 in the network. In equation (2), we set l_i to be large enough to contain all the possible cache data copy. In equation (3), we set $r_{j0} = 1$. In this special case, it can transform into the actual Uncapacitated Facility Location (UFL) problem. Since UFL problem is NP-Hard, and one special case of our problem is UFL, our problem is harder than UFL problem, thus NP-Hard as well. Next we propose an approximation algorithm.

The intuition of the approximation algorithm is to approximate the largest value of dual problem. First, we obtain all the information from the network, including the current groups, connections, demanding nodes, and data chunks. We then obtain the dual problem and increasing the certain dual variables, in order to gradually advance to the largest possible value of dual problem. During the process, if certain criteria are met, the data chunk will be assigned to cache on a certain group. Basically, groups near the demand and with ample resources will cache the data chunk. We have proved that this approximation algorithm has the approximation ratio of 7. There are three types of nodes getting from the cached data, which is 1, 2 and 4 times larger in cost respectively than the previous one. This indicates that the approximation ratio is 7 at most. Due to space limitations, we leave out the detailed proof on the approximation ratio of this algorithm. [3]

$$\max \quad \sum_j \sum_n r_{jn}\alpha_{jn} - \sum_i l_i\beta_i \tag{8}$$

$$\text{s.t.} \quad \alpha_{jn} - \theta_{ijn} \leqslant \kappa_{ijn} \quad (\forall i, \forall j, \forall n), \tag{9}$$

$$\sum_j \theta_{ijn} - q_n\beta_i \leqslant \phi_{in} \quad (\forall i, \forall n), \tag{10}$$

$$\alpha_{jn}, \beta_i, \theta_{ijn} \geqslant 0 \quad (\forall i, \forall j, \forall n). \tag{11}$$

[3]The detail proof of the approximation ration can be viewed at https://mcl.cewit.stonybrook.edu/resources/7-approximation-proof/

Algorithm 1 Inter-group Optimization

Input: $\mathcal{G}, \mathcal{M}_n, \mathcal{N}, \mathcal{E}$
Output: $DF[], IDF[]$

$\forall (j,n) \in (\mathcal{G}_q, \mathcal{N})$
while $\exists (j,n) \notin DF \cup IDF$ **do**
  for all $j \in \mathcal{G}_q$, $n \in \mathcal{N}$ **do**
    if $(j,n) \notin DF$ **then**
      $\alpha_{jn} += U_\alpha$
    end if
  end for
  for all $i \in \mathcal{G}$,$j \in \mathcal{G}_q$, $n \in \mathcal{N}$ **do**
    if $\alpha_{jn} > \kappa_{ijn}$ **then**
      $\theta_{ijn} = \alpha_{jn} - \kappa_{ijn}$
    end if
    if $j' \in \mathcal{G}_q$ and $\kappa_{j'jn} > \alpha_{j'n}$ **then**
      $IDF[(j,n)] \leftarrow DF[(i,n)]$
    end if
  end for
  for all $i \in \mathcal{G}$,$j \in \mathcal{G}_q$, $n \in \mathcal{N}$ **do**
    if $\sum_{j \in \mathcal{G}_q} \theta_{ijn} \geqslant \phi_{in}$ **then**
      if $\exists (j,n) \in IDF$ **then**
        for all $j \in \mathcal{G}_q$ **do**
          if $(j,n) \in IDF$ and $\theta_{ijn} > 0$ **then**
            $(tj, tn) \leftarrow IDF[(j,n)]$
            $DF[(tj,tn)] \leftarrow (i,n)$
          else if $(j,n) \notin IDF$ and $\theta_{ijn} > 0$ **then**
            $DF[(j,n)] \leftarrow (i,n)$
          end if
        end for
      else
        $DF[(j,n)] \leftarrow (i,n)$
      end if
    end if
  end for
end while

TABLE I
NOTATIONS USED IN THE FORMULATION AND THE ALGORITHM

$e \in \mathcal{E}$	Connections in the network
$n \in \mathcal{N}$	Data chunks in the network
$m_n \in \mathcal{M}_n$	Data sources of data chunk n
$g \in \mathcal{G}$	Groups in the network
$\mathcal{G}_q \subset \mathcal{G}$	Set of groups which demand data chunk n
$DF[]$	Direct freeze pair set
$IDF[]$	Indirect freeze pair set
U_x	Unit increase value of var x
κ_{ijn}	The cost for demand j access data n from group i
ϕ_{in}	The fairness degree cost for group i cache data n
q_n	The size of data n
l_i	The available storage capacity of group i
r_{ijn}	The indication of group j request data n from group i
a_{ijn}, g_{in}	The assignment variable for caching
α, β, θ	The assignment variable in dual problem

The detailed approximation algorithm is described in Algorithm 1, and the notations used are given in Table I. The dual form of the optimization problem is given at (8) to (11).

The algorithm works as follows. For each pair of demand and chunk jn (i.e., a node in group j demanding certain chunk n), we need to find a group to store chunk n for group j. A demand-chunk pair jn "freeze" to a group if it finds a group that will cache data n (direct freeze), or a group also requires data n and has found another group will cache data n (indirect freeze). Every demand-chunk pair must freeze to a group in which the chunk will be stored. If the demand-chunk pair does

not have a group to store, the corresponding cost parameter on the α_{jn} is increased by a set unit (lines 3-7). Lines 9-11 ensure the feasibility of the problem, which is to satisfy (9). Lines 12-14 are to indirectly freeze the demand which has already been frozen to a group. Then, lines 16-31 are to decide which group to cache the corresponding chunks. Lines 17-26 are used to select groups supporting the demand-chunk pair that can freeze to an indirect frozen pair. Some of the groups will be selected to cache corresponding chunks. Lines 27-29 make the groups caching the data in which the demand-chunk pairs do not freeze to any frozen groups. When all the demand-chunk pairs are frozen, the algorithm terminates. The indices of direct freeze (i, n) indicates that group i will cache data chunk n. This information will then be forwarded to the corresponding group. Then the group will decide which nodes will cache the data chunk n next.

We now discuss the complexity. We assume G and N represent the cardinality of $\mathcal{G}$ and $\mathcal{N}$. Apparently, inside the while loop (between line 2 and line 32), the most time consuming part is the loop start at line 16. The complexity is $\mathcal{O}(G^2N^2)$ from the number of loops inside. Since $\mathcal{G}_q \subset \mathcal{G}$, the maximum complexity for getting all the demand-chunk pair is no more than $\mathcal{O}(N^2)$, and this applies for all the groups both caching and demanding ones. Lines 3-7 cost less than $\mathcal{O}(GN)$, and the lines 8-15 cost no more than $\mathcal{O}(G^2N)$. The outer while loop will execute for a longer time. However, it will not exceed $\max \kappa_{ijn}/U_\alpha$. If we increase α to $\max \kappa_{ijn}$, all demand-chunk pairs will be directly frozen. So the iteration time will be no more than $C = \max \kappa_{ijn}/U_\alpha$. Thus, in total, the complexity will be no more than $\mathcal{O}(CG^2N^2)$. For a certain network, C and G are determined, thus the computational complexity is related to the number of data chunks.

B. Intra-group Data Caching Optimization Problem

Intra-group optimization problem focuses on how to cache data chunks on certain nodes in a group to ensure fairness and low latency. The goal is to find the optimal way to put data chunks onto certain nodes, in which the nodes can get the demanded data chunks quickly, while keeping the caching fair among all the nodes in the group. Fair caching is a crucial concept in peer-based scenarios. It spreads the caching workload evenly among nodes and avoids concentration on small fractions of nodes. It is crucial since nodes will run out of resources too quickly. The idea that lets most nodes participate in caching based on the current situation is called fair caching.

The problem for determining caching locations can be mapped to an extension of the classical Facility Location problem. In this case, the Uncapacitated Facility Location (UFL) problem best suits our problem [16]. The UFL problem considers the accessing cost from nodes to the facility and the cost to build the facility. To use the UFL problem, we set the accessing cost as our own accessing cost, and the facility build cost can be mapped to the fairness degree cost. We use ETX as the latency metric for the accessing cost. The cost is the ETX between demanding nodes to data caching nodes. Fairness degree cost measures the resources usage condition of a node. Fair caching has been studied in [4], which we leverage the fairness degree cost in this study as

$$f_i = \frac{S(v)}{S_{\text{tol}}(v) - S(v)},$$

where $S(v)$ is the used storage of node v, and $S_{\text{tol}}(v)$ is the total storage of that node.

After getting the accessing cost and fairness degree cost, we plug them into the UFL problem and use the solutions (the optimal location of a facility) of the UFL problem to determine where to cache data chunk. Since the UFL problem is NP-hard, an approximation algorithm will be used. The best approximation ratio, for now, is 1.488. We implement one of the approximation algorithms with the approximation ratio of 4 to distributed data chunk inside a group [16].

C. Node Behavior Problem

The node behavior problem maps the unpredictable movement of nodes into simple algorithms that can be categorized into two major behaviors on the node-group level: node leaving a group or node joining a group. Each behavior will trigger the corresponding algorithm. These algorithms will try to preserve the optimal result of the inter-group caching.

1) Nodes leaving behaviors: The movement a node may change the connection thus topology. If a node loses all connections to every node in the group to which it now belongs, it *leaves* this group. This triggers the node leaving behavior since the resource of the group is changed. Algorithm 2 describes the detail process of the node leaving behavior algorithm. The algorithm redistributes data chunks optimally based on the resources a group now has.

Algorithm 2 Node leaving algorithm

Input: node v that will leave group g

```
1: g ← g − v
2: if v.DemandData = ∅ then
3:     Intra-group optimization (g)
4: else if v.DemandData = n then
5:     Intra-group optimization (g)
6:     v_0 ← FindStore(v, n)
7:     v_0.loanNo[v, n] ← v_0.loanNo[v, n] − 1
8: end if
```

The algorithm considers two situations whether the leaving node demands any data. If the leaving node does not demand any data, the group will redistribute the already cached chunks among nodes that are still in the group. In the leaving node demands some data, the data storage will not change immediately. Since the node is likely moving to a nearby group, the data can still be accessed quickly.

Note that the data will be managed by a "loan number", which represents the expected number of demand for the data this group cached. The original loan number equals the x_{ij}. The leaving node will change the "loan number" between groups, minus one from the original group from which it gets data. [4] If the loan number is 0, which means no groups thus nodes need the data, and the group will stop maintaining the data (e.g., not

[4]If the group, which the node leaves from, has some other nodes requesting the same data, then the group demanding for the data will not change, and the corresponding "loan number" will not change.

getting the new version of the data). Data will not be deleted until expiration in case there will be other demands later.

2) Nodes joining behaviors: After a node leaves a group, it will *join* a new group where it can benefit most to the network. To prevent the node from moving in and out of groups frequently, the node will become a member of a group that it has most connections to. Algorithm 3 describes the detail process for a node to join a group. This algorithm evaluates whether to get data for the joining node and keeps the inter-group optimization result.

Algorithm 3 Node joining algorithm

Input: node v that will join group g, $k > 1$

$g \leftarrow g \cup \{v\}$
if v.DemandData $= \emptyset$ **then**
    Intra-group optimization (g)
else if v.DemandData $= n$ **then**
    $v_0 \leftarrow$ FindData(n)
    $c \leftarrow \kappa_{gv_0n}$
    if $c > k \times \phi_{gn}$ **then**
        g.StoreData(n)
        Intra-group optimization (g)
    else
        v_0.loanNo$[v, n] = 1$
    end if
end if

The algorithm also considers two situations whether the leaving node demands any data. If a node joins a group without demanding any data, the group will redistribute the cached data chunks among nodes in this group, including this newly joined node. Since the resources are changed, data chunks must be redistributed by applying intra-group optimization. Meanwhile, if a node joins and it demands some data, the group needs to find the data to fulfill the demand. It first sends a request to find out where the data are cached or originated, and estimates the accessing cost, which is also the cost of fetching. If accessing cost is k times larger than the cost to get the data chunk from within the group, then the group can proactively cache data for this demand. Intra-group optimization will be applied. Otherwise, the group will add a request from groups which cached the data chunk. Then, the groups receive the request will adjust the "loan number". Since accessing costs will always be paid for accessing the data, k should be larger than 1. k is set based on node mobility to prevent a node from leaving a group quickly after joining. If a node moves quickly (like a car), k should be relatively larger. Due to space limitations, we will not elaborate on the determination of k.

In all situations listed, we only require intra-group optimization. It avoids frequently applying inter-group optimization, which costs much more computation and communication resources than intra-group optimization. It keeps the previous inter-group optimization results with minor variations. The deviation will be reset whenever another round if inter-group optimization applies.

V. Performance Evaluation

In this section, we evaluate proposed algorithms and strategies. We focus on answering the following questions: 1) How do the algorithms and strategies adapt to node mobility and network scale? 2) How do they perform under different amounts of data? 3) How do routing strategies and request patterns affect their performance?

To answer these questions, we implement our algorithms using ns-3 simulator [17]. We generate different mobility traces on all different number of nodes. Nodes are randomly placed in the field, with the average density 0.02km^2 per node. In average, each node has direct connections to 4 other nodes (neighbors). Node connections are set to ad-hoc Wi-Fi network, using physical layer IEEE 802.11g, with the practical connection range to be 152 meters found from ns-3. The packets for ETX probes use UDP broadcast at a data rate of 1 Mbps. The transmission of interest and data packets uses the route calculated from the ETX metric and transmitted through UDP at a data rate of 54 Mbps. The routing table is refreshed every second (in the simulation time) by the smallest ETX value. Nodes will send packets to the destination using routes computed from ETX metric.

In order to get a data chunk from the caching groups or nodes, the demanding nodes will send an interest packet to the nodes which cache the data chunk, and this node will send the data chunk packets back to the demanding nodes. The mobility model for nodes is the random waypoint in our evaluation. Unless otherwise specified, new data chunks are generated every 15 seconds, and for each data chunk, there are 5 nodes which will demand this data. The size of each data chunk is 5 Kbytes. We also leave 25% vacant storage when apply the inter-group optimization constraints (2). We conduct our simulations on computers with Intel Core i5-4590 and 8GB RAM. All results are the average of 5 simulations.

A. Performance under Different Data Amounts

We first evaluate the performance under different data amounts. In this case, there are 5 to 15 data chunks generated and then cached on selected nodes every 15 seconds. The total number of nodes in the network is between 100 to 500 and initially divided into around 10 to 20 groups for different networks. In the network, there are 30% moving nodes at human walking speed, and the rest of the nodes remain static.

Fig. 1 shows the average data access time (a), successful data delivery ratio (b) and message overhead (c). The successful data delivery ratio is the ratio between successfully delivered data chunks and the corresponding interest from the demanding groups. The average data access time is the time between sending the interest packet and receiving all corresponding chunks. The message overhead is the overall data packets transmitted in the network, not including interest packets. Our proposed strategy can obtain good delivery ratio under different amount of data, over 89%. Meanwhile, the average accessing time for a data chunk is less than 3 seconds, which shows our algorithms can achieve fast data access. As for the message overhead, the more data inside the network, the more data need to be transmitted, thus the more message overhead.

Fig. 1(d) shows the average number of hops for data transmitted to the demanding nodes. If there are more data in the network that is requested, the contention will be higher. The routing will not always go through direct paths since their ETX metrics around congested areas will be high. This can help

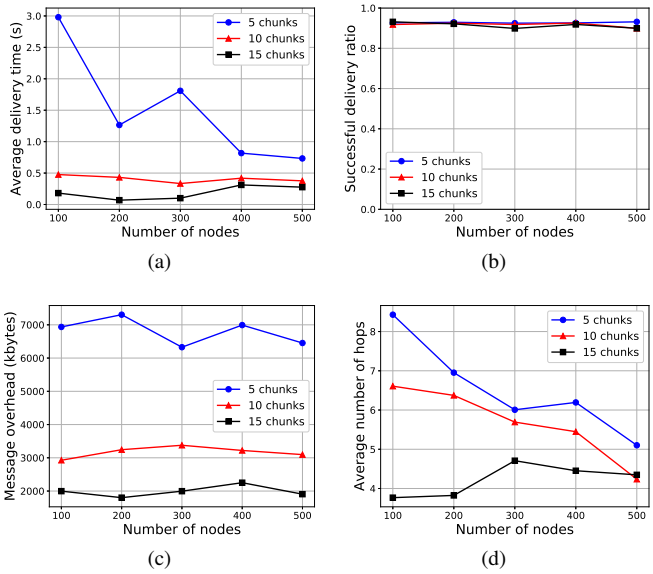


Fig. 1. The performance of the proposed strategy under different total amounts of data chunks in different sizes of networks. (a) The average data delivery time for the successfully delivered data. (b) The successful data delivery ratio. (c) The overall message overhead of all transmitted data chunk packets in the network. (d) The average number of hops data chunks being transmitted before they reach the destination.

explain why it takes longer to get the data when the amount data is larger, and why the overhead is not proportional to the number of data chunks. Meanwhile, we observe that the number of hops decreases when the number of nodes is larger. With more nodes but the same amount of data, less contention will appear in the network. Although the caching node might be farther from demanding node in hop count, the average data transmitted is smaller because of less contention. This helps to explain why the data access time drops when the number of nodes is larger.

B. Performance under Different Mobility

As we mentioned before, we map node mobility into two kinds of node behavior. We test our algorithms and strategies under different mobilities, from 30% to 50% of nodes moving like pedestrian walk (1.5m/s). The rest of the nodes are still immobile. We use the baseline settings as 5 chunks per 15 seconds with the total number of nodes between 100 to 500, initially 10 to 20 groups. This shows that our proposed strategies adapt to the scale of the network.

Fig. 2 shows the average data access time (a) and successful data delivery ratio in this scenario (b). In general, our proposed solutions still achieves high data delivery ratio, at least 89%. Meanwhile, the average data access time remains less than 3.3 seconds. We also plot the average number of hops data travel in Fig. 2(c). It shows that the more mobility, the more data packets will be transmitted. Since our strategies compute the routing table frequently (once per second), mobility could incur more data transmissions. It also explains the decreasing time when the number of nodes is larger since data packets are transmitted fewer times.

We also test our algorithms and strategies under more intense mobility. In some real scenarios, cars serve as important edge

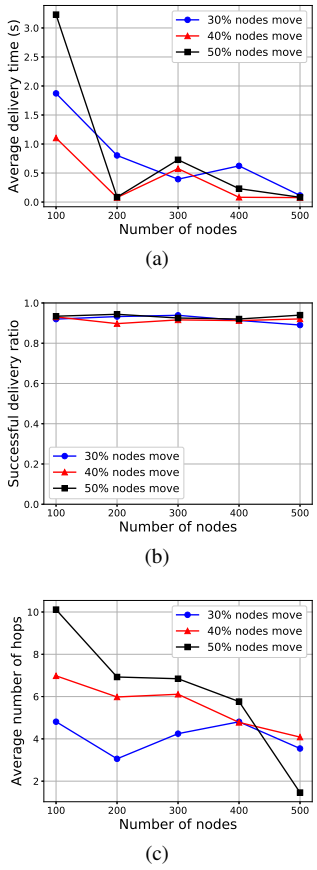


Fig. 2. The performance of our proposed strategy under different percentage of nodes move at **pedestrian** speed. The strategy can adapt to different situation of mobility of the nodes. (a) The average data delivery time. (b) The successful data delivery ratio. (c) The average number of hops of all transmitted data chunks.

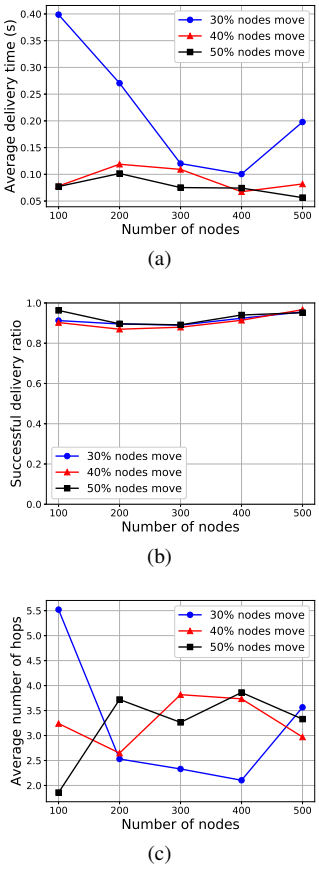


Fig. 3. The performance of our proposed strategy under different percentage of nodes move at **car** speed. The strategy can adapt to intense situation of mobility of the nodes. (a) The average data delivery time. (b) The successful data delivery ratio. (c) The average number of hops of all transmitted data chunks.

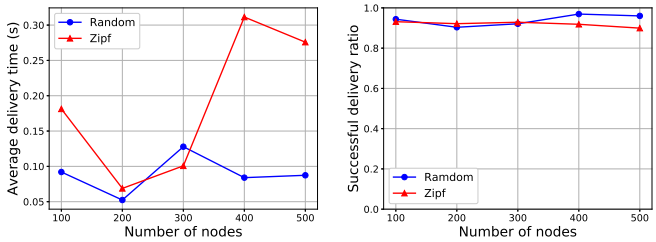
devices with more computational capabilities, power, and storages. Thus, we set 30% to 50% of all nodes moves like a car (11-15m/s, about 25-35 mph) in these scenarios. The rest of the nodes are still immobile.

Fig. 3 also shows the average accessing time (a) and successful data delivery ratio (b) in this much more intense mobility scenario. The mobility has some impact on the delivery ratio of data, but it remains over 85% recall rate. The data access is very quick, less than 0.5 seconds. Fig. 3(c) shows that the number of data transmission hops is lower than that in previous work and some fluctuation exists. This is due to the selection of nodes. Since nodes move faster, it is likely to have more fluctuations in hops. The above simulation results show that our design solutions perform well under intense node mobility.

C. Performance under Different Request Patterns

We next evaluate the impact of different data request patterns. Data are often of different popularities, and the each data chunk may be requested in different frequency. We test our algorithms and strategies under two different request patterns, random and Zipf-like distribution [18]. Zipf-like request distribution is a common statistical model for data requests. For different request patterns, different data chunks are requested by different

numbers of nodes. We keep other parameters in the baseline settings.



(a) (b)

Fig. 4. The performance of different request pattern for data demand. Zipf vs random. The difference is not very significant. (a) The average data delivery time. (b) The successful data delivery ratio.

Fig.4 shows the average data delivery time (a) and successful data delivery ratio (b) on different data request patterns. Between these two request patterns, the difference is not significant, presumably the numbers of data chunks and requests are the same. It also achieves very quick data access, in less than 0.3s. For the data delivery rate, the result is also very satisfactory. This also shows that our proposed strategy can adapt to different data request patterns.

VI. Conclusion and Discussion

In this paper, we have introduced the problem of peer data caching under high mobility nodes and varying scale networks in pervasive edge computing environments. We have proposed a grouping method to build a two-layer hierarchy design to reduce the entities being managed in each layer. Then we have formulated inter-group and intra-group optimization problems, and propose a 7-approximation algorithm to solve inter-group optimization. We have also proposed the node behavior problem, to transform node mobility into respective node-group action processes. We have developed algorithms to perform these action processes. We have implemented proposed algorithms and tested it on network simulator. The result shows that our proposed strategies work in different network scales and different node mobilities, while achieving satisfactory results for data accessing.

Running continuously over time is crucial for applying PDS into real new scenarios. We have proposed the group behavior problem and have had some primitive results. In the future work, we will discuss in detail about the design and algorithms for continuous running. Meanwhile, the approximation algorithm we proposed is a centralized one. Centralized algorithm needs a centralized server to collect and control data. Centralized algorithms are sometimes not practical enough as detailed information of each node is hard to collect timely. We will design distributed algorithms in the future. Privacy is also one of the crucial issues in edge scenarios. Some new technologies such as blockchain can help on this topic. We will address how to apply such technologies in edge scenarios in our future work.

For the dissemination part, the basic idea is the cache nodes will first fetch the data from the data source, and when the interest comes to the node, it can send the data to the demanding node. The data requesting and transmission patterns are totally different. In our future work, we will design the corresponding evaluation on this phase. We will also consider real traces and terrains for the node mobility and the methods to improve data access robustness (e.g., data retransmission when lost) in our future work.

Acknowledgment

This work is supported in part by US National Science Foundation under grant numbers 1513719 and 1730291.

References

[1] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5g wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.

[2] X. Song, Y. Huang, Q. Zhou, F. Ye, Y. Yang, and X. Li, "Content centric peer data sharing in pervasive edge computing environments," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 287–297.

[3] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Intelligent Systems and Control (ISCO), 2016 10th International Conference on*. IEEE, 2016, pp. 1–8.

[4] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair caching algorithms for peer data sharing in pervasive edge computing environments," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 605–614.

[5] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2017.

[6] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "Music: Mobility-aware optimal service allocation in mobile cloud computing," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 75–82.

[7] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1281–1290.

[8] P. Stuedi, I. Mohomed, and D. Terry, "Wherestore: Location-based data storage for mobile devices interacting with the cloud," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010, p. 1.

[9] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[10] J. O. Fajardo, F. Liberal, I. Giannoulakis, E. Kafetzakis, V. Pii, I. Trajkovska, T. M. Bohnert, L. Goratti, R. Riggio, J. G. Lloreda *et al.*, "Introducing mobile edge computing capabilities through distributed 5g cloud enabled small cells," *Mobile networks and applications*, vol. 21, no. 4, pp. 564–574, 2016.

[11] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey, "The uncapacitated facility location problem," DTIC Document, Tech. Rep., 1983.

[12] A. Gupta, A. Kumar, and T. Roughgarden, "Simpler and better approximation algorithms for network design," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. ACM, 2003, pp. 365–372.

[13] C. Swamy and A. Kumar, "Primal-dual algorithms for connected facility location problems," in *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 2002, pp. 256–270.

[14] S. Li, "A 1.488 approximation algorithm for the uncapacitated facility location problem," *Information and Computation*, vol. 222, pp. 45–58, 2013.

[15] D. S. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wireless networks*, vol. 11, no. 4, pp. 419–434, 2005.

[16] F. A. Chudak and D. B. Shmoys, "Improved approximation algorithms for the uncapacitated facility location problem," *SIAM Journal on Computing*, vol. 33, no. 1, pp. 1–25, 2003.

[17] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[18] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 1999, pp. 126–134.