Distributed Data Retrieval for Real-time Decision-making under Freshness Constraints

By

Shuo Feng

Senior Thesis in Computer Engineering

University of Illinois at Urbana-Champaign

Advisor: Professor Tarek F. Abdelzaher

December 2015

Abstract

This paper describes a distributed data retrieval algorithm for crowd-sensing application, which aims to collect data with minimized bandwidth cost while satisfying data freshness constraints. In a resource-limited setting, data loses freshness very fast. For instance, the condition of a road during a rush hour may be dynamic due to the rapid change of the traffic. In order to schedule an optimized route to a destination from a given location, we have to know its real-time condition. The protocol we design is to exploit logic dependencies among data by using and-or tree to reduce the overhead of the network and handle concurrent requests at the same time. Meanwhile, we further modify the centralized system into a distributed form so that each node in this network is able to calculate the best retrieval order locally. Furthermore, we integrate some ideas of other literature to let each node store the retrieved data locally due to the fact that the price of storage is lower and lower these days. Finally, we implement part of the algorithms and test the efficiency of using varying probabilities and earliest deadline first to sort queries.

Subject Keywords: distributed systems; crowd-sensing

Acknowledgments

I would like to thank Professor Tarek F. Abdelzaher and Shaohan Hu for providing me great support.

Contents

1. Introduction 1
2. Literature Review 2
3. Description of Research Results 3
    3.1 Data retrieval algorithm 3
    3.2 Conditional probability calculation 5
4. Conclusion 9
References 10

1. Introduction

In many real-world scenarios, the network bandwidth is very limited. For instance, some less developed areas around the world may not have money to build broadband networks. In other settings such as post-earthquake, the infrastructure is severely damaged. Therefore, there is a need to find a way to save the network resources. Moreover, some emergencies need to be handled as quickly as possible. For instance, a rescue team needs to save citizens in a building after an earthquake. If there are other less urgent requests happening at the same time, we still need to perform the rescue task first since saving people's lives is of the first priority. Once the task is fixed, we need to evaluate the conditions of different routes we might need to go through. Each route consists of several roads, which may or may not be destroyed. Once we know one road is unusable, we will decide not to go along that path. We can model that into a Boolean expression (shelterOccupied) ∧ (path1IsGood ∨ path2isGood ^ path3isGood). If we know path2 is in bad condition, the retrieval of information of path 3 would become meaningless. Since the size of a piece of information can be very large, avoiding fetching useless data can make a huge difference. Thus, the key to minimizing the network bandwidth cost is to fetch the data in a proper order. Another thing we need to take into consideration is the freshness deadline of each piece of data. In other words, the information contained in each of the data can expire within a short amount of time. The information that indicates that the path was available one hour ago might not imply that the path is available now. Therefore, the more perishable the data, the later should we execute the actual retrieval. Therefore, we integrate the above factors and come up with an efficient algorithm.

The rest of the paper is organized as follows. We discuss some relevant work in Section 2. We then discuss the algorithms we design and some calculation processes in Section 3. In the end, Section 4 summarizes the paper.

2. Literature Review

Crowd-sensing is one of the hottest areas in distributed systems nowadays. Many scientists put efforts into finding various scenarios in which the crowd-sensing can be applied. For instance, in order to find legal parking lots more efficiently, a crowd-sensing model was given by Coric et al. [1]. As far as the basic underlying mathematic model is concerned, the optimal Boolean predicate evaluation order has also been studied in computer theory. Casanova et al. [2] analyze the query tree structure and suggest some efficient evaluation orders. Other researchers also give some potential heuristics to optimize the result. In distributed systems community, Hu et al. [3] [4] give an algorithm to handle the concurrent request scenarios and deal with the query deadline and object freshness constraints, respectively. In contrast to the previous work, our model satisfies all the constraints above simultaneously in a distributed fashion. Moreover, we add a pre-fetch feature to further its performance.

3. Description of Research Results

3.1 High level distributed data retrieval algorithm

This paper adapts a centralized data retrieval algorithm to distributed mode for crowd-sensing scenarios under freshness constraints. It is known that the internet resources are very limited in a crowd-sensing setting, such as the post-disaster environment. In that case, we have to use an optimized algorithm to retrieve data to reach our goal. We model it into a tree structure, where leaf nodes stand for the data objects and non-leaf nodes represent AND-OR relationships. In order for the whole tree to be true, we have to ensure that each AND branch is true and at least one OR branch is true. As long as we know one OR branch is true, we can shortcircuit other OR branches. Similarly, we can shortcircuit other AND branches as long as we know there is an AND branch is evaluated to be true. Therefore, the tree model gives us good evaluation guidelines.

Another issue we need to consider is the freshness of each data. It could be the case that the data we retrieve expire when the decision is made since it takes some time before we gather all the data we need. Therefore, we have to add parallel level to meet the freshness deadline or we need to reorder the data retrieval. Moreover, we also take into consideration the deadline of a request since each requester might need to respond to an emergency as soon as possible.

Overall, we need to perform three evaluations based on different factors. First, we need to figure out the query needed to be processed with the earliest deadline. Second, in order to minimize the total expected bandwidth cost as much as possible, we focus on one query tree and evaluate which data object needs to be retrieved first. Here we use t_{i_j} to denote the data object on i's query tree. As the equation (1) shows, the higher the value of $\frac{1-p_{i_u}}{c_{i_u}}$, the earlier the retrieval of t_{i_v} should be.

$$t_{i_u} > t_{i_v} \leftrightarrow \frac{1 - p_{i_u}}{c_{i_u}} > \frac{1 - p_{i_v}}{c_{i_v}} \quad (1)$$

Third, in order to meet the freshness deadline, we also need to order data objects. We apply the Latest Deadline First policy to order all the data first. If we find that some piece of data still expires under the best optimized order, we have to add the parallel retrieval level. After we combine the above three order rules, we can achieve a balanced retrieval order.

Input: The set M of retrieved data in previous round, the set Q of all unresolved queries and set O of object requests, current time T

1 Initialize R ← ∅

2 Order all its queries according to their deadlines

3 Fetch one query following the earliest deadline first policy

4 Repeat

5 use values in M to update cache

6 If current node is the root node

7 prune unnecessary leaves in Qc ← ∅, Qd ← LDF order of all objects in the query, Sp ← ∅

8 L ← {tij } sorted in descending order of (1− pi)/cij

9 while |Qd| > 0

10 te ← end element of Qd

11 Qd ← Qd \ te, Sp ← Sp ∪ {te}

12 if Qd + Sp meets freshness deadlines

13 L<-L\Sp

14 break

15 for each tl in L

16 QH ← Qc + t1 + Qd \ t1

17 if QH meets freshness constraints

18 Qd ← Qd \ t1

```
                    Qc ← Qc + t1
            for each t2 in L   such that t2 is  less optimized than t1 but are in the same branch
                    QH ← Qc + t2 + Qd \ t2
                    if QH  meets freshness constraints
                                        Qd ← Qd \  t2 , Qc ← Qc +   t2
                                        break
        bind t1 and t2 together
        append the rest of Qd to QH
    for each te  in QH
        if te  is being processed by checking whether they are marked in the cache
                check whether resolved by without violating the freshness
                    if not violate
                            remove it from QH
                    else
                    Perform the sequential retrieval
```

In addition, we modify the algorithm running on a single node to run on different nodes. Each node can compute its own optimized order itself, and can function as a forward node and storage node at the same time.

3.2 Conditional probability calculation

In order to update the probability for each object data, we need to compute the conditional probability according to the elapse of the time after the data exists in the local cache. To make it close to the real world scenarios, we search the data from the California Department of Transportation. We first plot the average traffic in a day. Then we set a threshold by heuristics, which is the red line in the left graph in figure 1 to indicate the level of traffic. If the curve is above the threshold, it is believed to be congested. Then we designed our own algorithms to calculate whether the traffic is still heavy after

different periods of time and plot the result as a graph on the right. After analyzing, we get an approximate piece wise function to fit the discrete points.

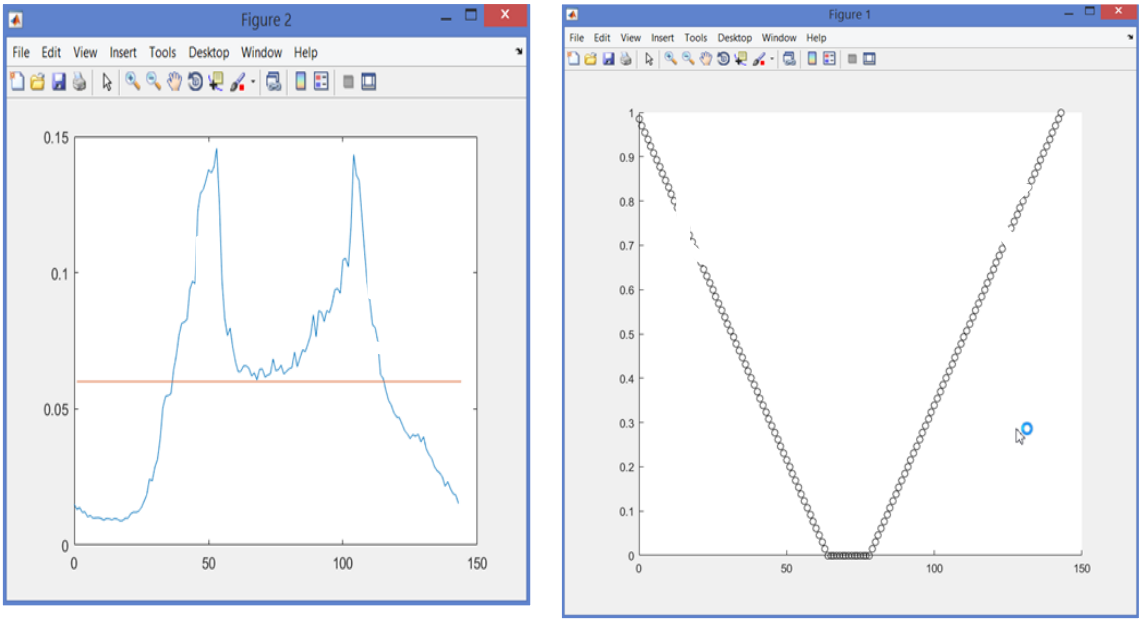


Figure 1 Left: The traffic condition in a day with x axis representing time in minutes and y axis representing level of traffic. Right: The conditional probability changing with time

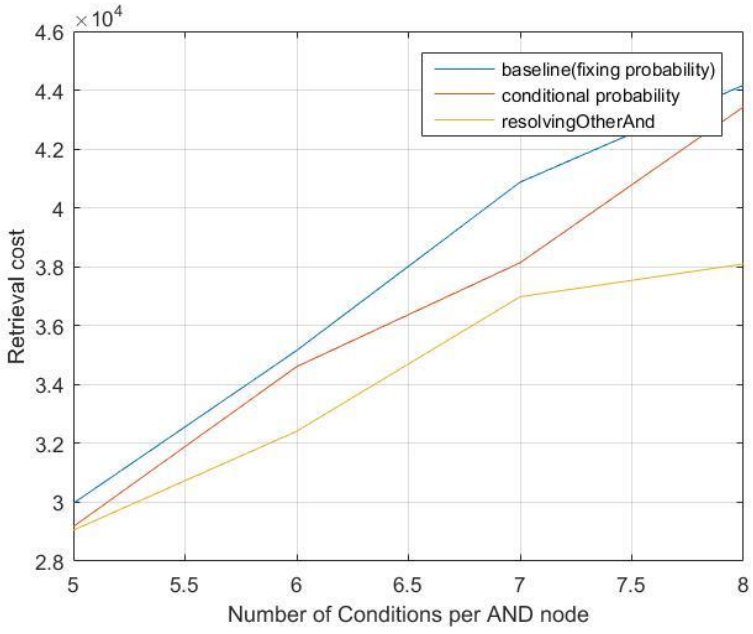
3.3 EDF policy

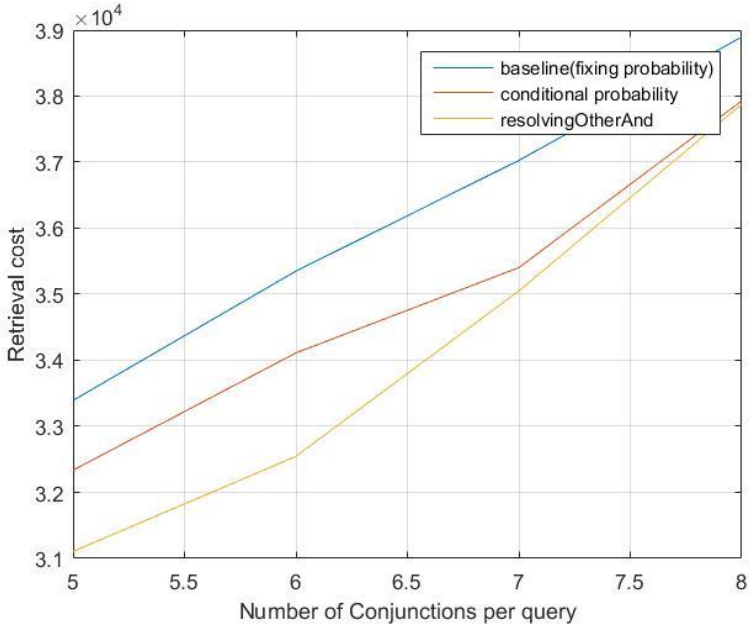
In order to handle multiple requests scenarios, we use Earliest Deadline First policy to sort the queries and resolve each of them sequentially. It is obvious that we need to meet the earliest deadline since these queries are more emergent. We compare it with the random order case.

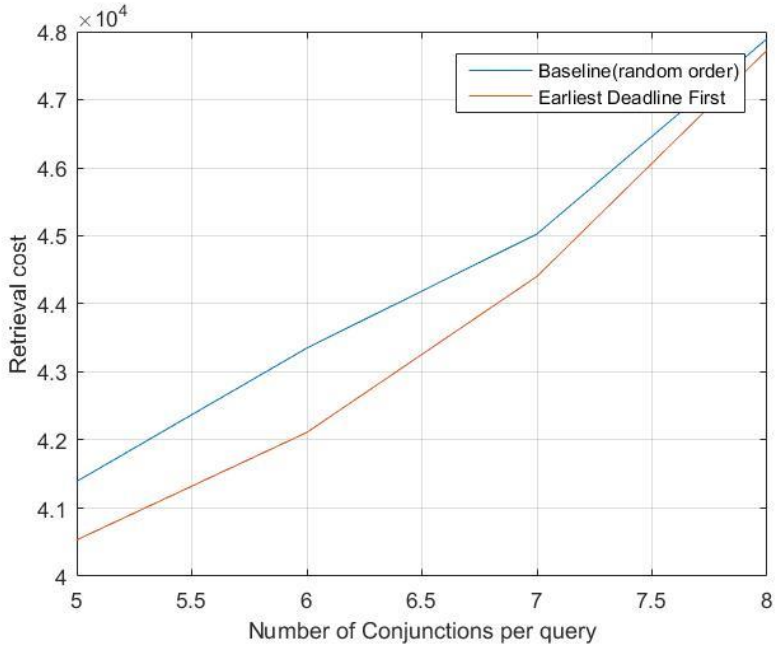
3.4 Simulation result

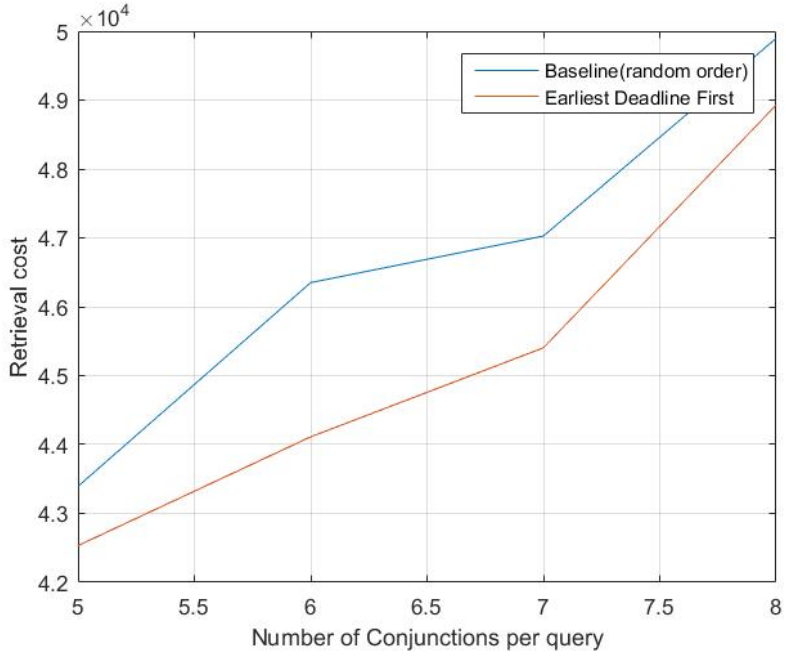
After applying the conditional probability, I write the code and compare the results of three different cases. The base case is that the probability of each condition doesn't change with time and we only use each piece of data to resolve the conjunction where the data is in. In the second case, I vary the probability by time according to the function computed in the above section and still only resolve the current conjunction. In the third case, I resolve conditions in other AND nodes and use conditional probability as well. The result goes as follows. The improvement makes a strikingly difference and

outperforms the baseline. The conditional probability decreases the cost and resolving multiple conjunctions also lower the cost a lot.






×10^4
4.8
4.7
4.6
4.5
4.4
4.3
4.2
4.1
4
Retrieval cost
5
5.5
6
6.5
7
7.5
8
Number of Conjunctions per query
Baseline(random order)
Earliest Deadline First


×10^4
5
4.9
4.8
4.7
4.6
4.5
4.4
4.3
4.2
Retrieval cost
5
5.5
6
6.5
7
7.5
8
Number of Conjunctions per query
Baseline(random order)
Earliest Deadline First

4. Conclusion

During the research, we developed an efficient algorithm for the crowd-sensing scenarios. We have integrated a variety of constraints such as freshness deadline and request deadline. One of the innovative parts of our works is the pre-fetch scheme we applied to reduce the bandwidth cost. Another key breakthrough is the distributed form of calculating the retrieval order, which can be robust to extreme conditions and has the ability to get the globally optimized retrieval order rather than locally optimized order. We use heuristics to tune the value of different parameters such as the bandwidth limit and the parallel retrieval level. We also gather real-world data of traffic of San Francisco to calculate the conditional probability of the degree of congestion. Future work can be done to optimize the parameters of the heuristics to achieve better performance.

References

[1] V. Coric and M. Gruteser, “Crowdsensing maps of on-street parking spaces,” in *IEEE DCOSS*, 2013.

[2] H. Casanova, L. Lim, Y. Robert, F. Vivien, and D. Zaidouni, “Cost optimal execution of Boolean query trees with shared streams,” in *IEEE IPDPS*, 2014.

[3] S. Hu, S. Li, S. Yao, L. Su, R. Govindan, R. Hobbs, and T.F. Abdelzaher, “On exploiting logical dependencies for minimizing additive cost metrics in resource-limited crowdsensing,” in *IEEE DCOSS*, 2015.

[4] S. Hu, S. Yao, H. Jin, Y. Zhao, Y. Hu, X. Liu, N. Naghibolhosseini, S. Li, A. Kapoor, W. Dron, L. Su, Amotz Bar-Noy, P. Szekely, R. Govindan, R. Hobbs, and T.F. Abdelzaher, “Data acquisition for real-time decision-making under freshness constraints,” in *IEEE RTSS*, 2015.