KIPS Tr. Software and Data Eng.
Vol.4, No.9 pp.347~356 pISSN: 2287-5905



http://dx.doi.org/10.3745/KTSDE.2015.4.9.347

# Finding Frequent Route of Taxi Trip Events Based on MapReduce and MongoDB

Fadhilah Kurnia Putri† · Seonga An† · Magdalena Trie Purnaningtyas†† · Han-You Jeong††† · Joonho Kwon††††

## ABSTRACT

Due to the rapid development of IoT(Internet of Things) technology, traditional taxis are connected through dispatchers and location systems. Typically, modern taxis have embedded with GPS(Global Positioning System), which aims for obtaining the route information. By analyzing the frequency of taxi trip events, we can find the frequent route for a given query time. However, a scalability problem would occur when we convert the raw location data of taxi trip events into the analyzed frequency information due to the volume of location data. For this problem, we propose a NoSQL based top-$K$ query system for taxi trip events. First, we analyze raw taxi trip events and extract frequencies of all routes. Then, we store the frequency information into hash-based index structure of MongoDB which is a document-oriented NoSQL database. Efficient top-$K$ query processing for frequent route is done with the top of the MongoDB. We validate the efficiency of our algorithms by using real taxi trip events of New York City.

**Keywords : Taxi Trip Data, Top-*K* Frequent Query Processing, NoSQL Database, MapReduce, MongoDB**

# 택시 데이터에 대한 효율적인 Top-*K* 빈도 검색

Fadhilah Kurnia Putri† · 안 성 아† · Magdalena Trie Purnaningtyas†† · 정 한 유††† · 권 준 호††††

## 요 약

IoT(사물인터넷) 기술의 빠른 개발로 인하여 기존의 택시들은 디스패처와 위치 시스템을 통해 서로 연결되고 있다. 일반적으로 현대의 택시들은 경로 정보를 획득하기 위한 목적으로 GPS(Global Positioning System)를 탑재하고 있다. 택시 운행 데이터들의 경로 빈도를 분석하여, 주어진 질의 시간에 해당하는 빈번한 경로를 찾을 수 있다. 그러나 위치 데이터의 용량이 매우 크고 복잡하기 때문에 택시의 운행 이벤트의 위치 데이터를 분석된 빈도 정보로 변환할 때에 확장성 문제가 발생한다. 이 문제를 해결하기 위하여, NoSQL 데이터베이스에 기반한 택시 운행 데이터에 대한 Top-$K$ 질의 시스템을 제안한다. 첫째, 원시 택시 운행 이벤트를 분석하고 모든 경로들의 빈도 정보를 추출한다. 추출한 경로 정보는 NoSQL 문서-지향 데이터베이스인 MongoDB에 해시 기반의 인덱스 구조로 저장한다. 주로 발생하는 경로에 대한 효율적인 Top-$K$ 질의 처리는 몽고DB의 상에서 이루어진다. 미국 뉴욕시의 실제 택시 운행 데이터를 이용한 실험을 통하여 알고리즘의 효율성을 검증하였다.

**키워드 : 택시 운행 데이터, Top-*K* 질의 처리, NoSQL 데이터베이스, MapReduce, MongoDB**

## 1. Introduction

The transportation system is one of the fundamental needs for people in many big cities since millions of people move from one place to another day to day. In these transportations, the taxi service has been an important role due to its proximity, personalized service and convenience to access anywhere. In the case of New York City, metropolitan which we target in our study, 50,000 taxi drivers take charges of the 485,000 trips each day, totaling 175 million trips per year in 2014[1].

Global Position System(GPS) devices, one of the widely adopted IoT(Internet of Things) devices, already have been equipped to most taxis to record the location of the pickup, the location of the adropoff, and other states.

※ 이 논문은 2015년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2014R1A1A2055639).
† 비 회 원 : 부산대학교 빅데이터협동과정 석사과정
†† 비 회 원 : 부산대학교 전기컴퓨터공학과 석사과정
††† 비 회 원 : 부산대학교 전기공학과 교수
†††† 정 회 원 : 부산대학교 빅데이터협동과정 교수
Manuscript Received : July 28, 2015
First Revision : September 3, 2015
Accepted : September 3, 2015
* Corresponding Author : Joonho Kwon(jhkwon@pusan.ac.kr)

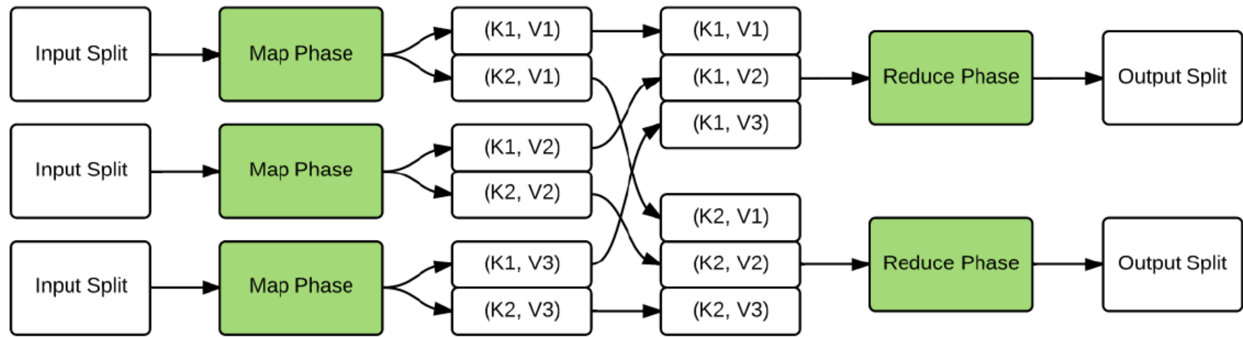



Fig. 1. A MapReduce Framework

These positioning techniques allowed us to collect and accumulate location data of the movement histories of taxi trips. Various meaningful information could be extracted by analyzing this data. For example, in [2], the researchers try to convert urban taxi data into information with insights into many different aspects of city life. To detect hot spots in urban cities[3], other applications of traffic managements trace and track the taxi trip events. In this regard, we are also interested in finding frequent routes of taxi trip events. This is done by designing a top-$K$ query processing system for taxi trip events.

We will face a scalability problem with a single machine's storage and local computation power due to the huge volumes of taxi location data. For example, daily GPS trace data of taxicab in a Chinese city is about 450GB for 6 months[4]. The MapReduce framework becomes the de facto standard for distributed processing of huge datasets. Thus, we will propose distributed analytic algorithms executed across multiple machines.

Since extracting the frequent information needs to scan all dataset, the MapReduce framework is suitable for this process. However, it needs an efficient query processing to provide answers for various user requests. Thus, we propose to use a MongoDB database, a document-oriented NoSQL database, is suitable for processing users' requests.

In this paper, we propose a top-$K$ query processing system for handling taxi trip events. This system is a combination of MapReduce and NoSQL database technologies. We can extract the frequent routes by analyzing taxi trip events and store them into a hash-like index. We group trip events for each minute during the MapReduce. A user requests his interest by specifying the time and duration at the top of MongoDB. Then, he can obtain the frequent routes corresponding to the query time and duration.

The remainder of this paper will be organized as follows : Section 2 presents related research work. Section 3 explains the background of our study, the data description and geographic assumption. In Section 4, we present how to extract the frequent route information and process the top-$K$ query by combining MapReduce and NoSQL technologies. We shall report the experiment results in Section 5. Section 6 concludes the paper.

## 2. Related Work

### 2.1 Analyzing Taxi Trip Data

Many studies have been made for analyzing taxi service based on the real-time/historical location data. GPS is becoming available widely as one of IoT (Internet of Things) devices. Hence, various research areas use GPS generated data for analyzing the location. In [5], authors used taxi GPS traces for urban land-use classification, especially for recognizing the social features of the region. Lee et al: analyze taxi location data in Jeju Island, Republic of Korea to design a pick-up pattern scheme of a taxi service[6]. They execute a k-means clustering procedure to group the taxi location variables and create its spatio-temporal pick-up frequency. A MapReduce programming model[7] is applied for solving traffic problems since the real GPS traffic data is a large-scale dataset in nature.

### 2.2 MongoDB

The traditional relational database is widely used for storing and querying structured data. Also, another kind of technology is NoSQL database, which has standout scalability and availability[8]. Recently, NoSQL databases get high. There exists a study which attempts to use NoSQL database to replace the relational database[9]. The authors try to compare the two database system : (1) MySQL[10] as RDBMS and (2) MongoDB[11] as NoSQL database. At this work, the researchers enter 100,000 textbooks information

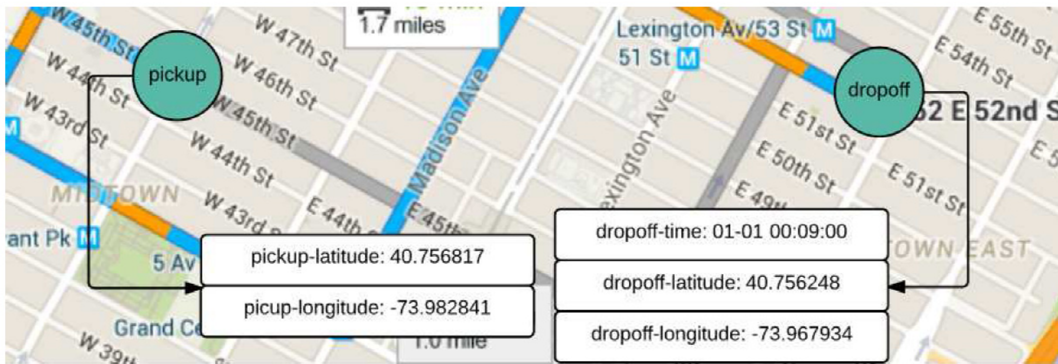



Fig. 2. Taxi Trip Data View

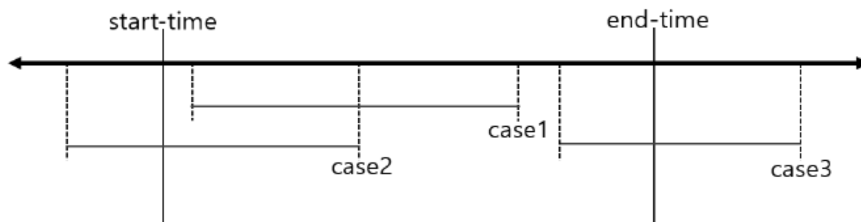



Fig. 3. Cases of Taxi Trip Events

data into both database and check the cost time of MongoDB and MySQL. The query of MySQL is executed by a join operator which requires several tables. The query of MongoDB is executed by reading a document table which contains all necessary information. The query execution time and data insertion time of MongoDB are faster than those of MySQL. This work testifies the efficiency of MongoDB, which we will use in our study. Dede et al: provides the relative advantage and disadvantage of combining MongoDB with Hadoop, the most popular implementation of MapReduce[12].

### 2.3 MapReduce Framework

MapReduce is a programming model with a distributed algorithm for processing large datasets[13]. As shown in Fig. 1, MapReduce has two important phases : (1) a map phase and (2) a reduce phase. The map and reduce phases can be designed for any computation over users input dataset and implemented as a mapper and a reducer class. A key-value pair is a basic processing unit for an input and an output of the MapReduce.

## 3. Preliminaries

### 3.1 Taxi Trip Data Description

The taxi trip data is based on a data set released under the FOIL(The Freedom of Information Law) and made public by Chris Whong[14]. This raw data reports the taxi trip events based on geospatial data streams from New York City. The total size of data is approximately 12GB for the whole year 2013 containing about 173 million events. Each event consists of 17 attributes including a location, timestamps for pickup and drop off, payment information and so on. For our research, we will use 5 attributes as shown in Fig. 2 : dropoff-time, pickup-longitude, pickup-latitude, dropoff-longitude and dropoff-latitude.

### 3.2 Definition of The Route

The problem we would like to challenge is the identification of recent frequent routes. In other words, the goal of our study is finding top-$K$ frequent routes for the given query time and duration. The query time consists of end-time, last $T$ minutes, and the number of frequent routes ($K$). The start-time is automatically computed as the time before $T$ minutes from the end-time.

Taxi trip events during $T$ minutes could be categorized into two cases as shown in Fig. 3 : (1) Case 1 : both pickup-time and dropoff-time are within the interval denoted as [start-time, end-time]. (2) Case 2 : the dropoff-time is before the end-time of the query, however the pickup-time is before the start-time. (3) Case 3 : the pickup time is after the start-time of the query whereas the drop-off time is after the end-time. In our study, we will consider the case 2 equally with case 1 regardless of pickup-time, since the case 1 is finished before the end-time of the query. Unlike this the case 3

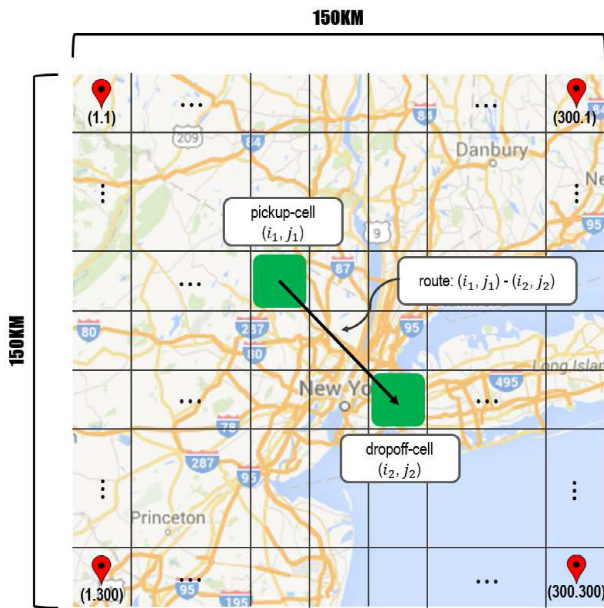



Fig. 4. Definition of The Route

is not in our consideration since the taxi trip is not terminated until the end-time. In other words, if the dropoff-time is in [start-time, end-time] of the query, we consider this event is within the interval of taxi trip events during $T$ minutes.

For defining a route of a taxi trip, we first need the cell identifiers of pickup/drop-off locations from coordinates. We assigned coordinates of pickup/dropoff locations to grid cells for New York City due to the length and complexity of geospatial data. We assume that a simplified flat earth for mapping coordinates. The number of the grid cell is 300*300 and the size of each cell is 500 by 500 meter. The total size of the grid cell is the square, the length of one side is 150KM. Each cell has an identifier of a pair $(i, j)$ where i and j represents the row number and column number respectively. The left top of initial cell, which of identifier is (1, 1), is located at (41.46691978, -74.911343) in geographic coordinates, longitude and latitude[15]. The identifier for the cell grid increase towards the east and south bound. The shift to east could be an increment in i, and the shift to south could be an increment in j. Due to the flat earth assumption, the i and j value of an identifier pair will be increased/decreased as 1 when 0.005986 degrees in longitude and 0.004491556 degrees in latitude is occurred respectively. In this regard, we map coordinates to grid cells for New York City, we can compute the identifier for coordinates.

Then, we define the route of a taxi trip event as the pair of identifiers of pickup position and drop-off position as illustrated in Fig. 4. If there is an event which of the

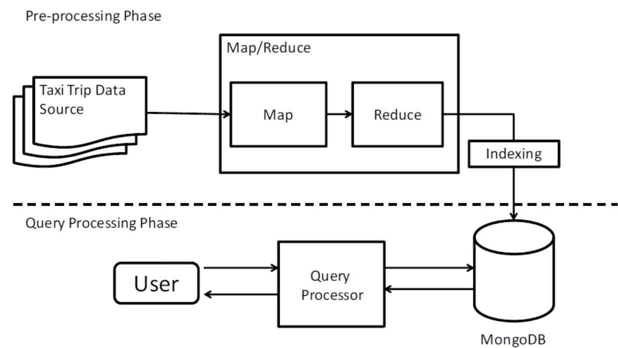



Fig. 5. System Architecture

pickup or drop-off locations is out of grid cell range, we will treat this event as an outlier and removed from the results.

Let us consider an example route. Assume that we obtain a coordinate (40.756817, -73.982841) as a pickup-position. This coordinate that has corresponds the cell having identifiers (158, 157). If we have a drop-off position coordinate such as (40.756248, -73.967934), then the cell identifiers will be (157, 158). Thus, the route of this taxi trip event could be represented as (155, 158)-(157, 158).

## 4. The Proposed System

In this section, we shall describe the architecture of the proposed system and distributed approaches for finding top-$K$ frequent routes.

### 4.1 Overall Architecture

Fig. 5 shows the overall system architecture. We use two phases for obtaining top-$K$ frequent routes from taxi trip events. The pre-processing phase groups taxi trip events for each minute based on MapReduce algorithms. The output of MapReduce algorithms is a hash-based index structure and stored into a MongoDB document store. At the query processing phase, a user submit his interest to the system by specifying parameters and obtains the top-$K$ frequents routes.

For implementing a distributed top-$K$ query system, there might be two strategies. The first approach is to implement all algorithms using only the MapReduce framework. The result of this approach is all outputs for top-$K$ frequent route with range time $T$. One of the limitations of this approach is too much duplicate data generated by map/reduce functions as shown in Fig. 6. For example, a user wants to get top-10 frequent routes during 10 minutes from any specified time. The map function duplicates all input data several times

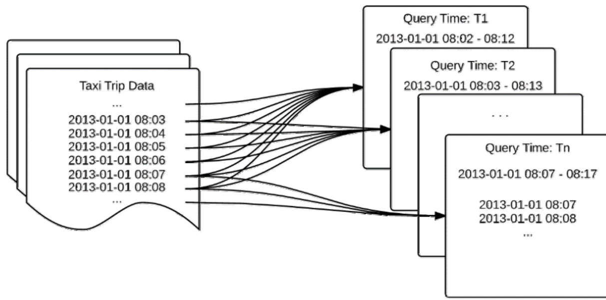



Fig. 6. Duplicate Data

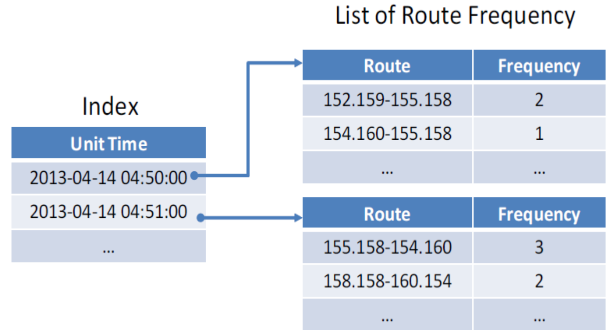



Fig. 7. Data Structure

which corresponds to the range of 10 minutes interval. Since the taxi trip data in one minute can be included several time ranges, the same data can be copied $T$ times. The reduce function aggregates the duplicated data for the same query time.

Another disadvantage is that we need to implement map/reduce functions for all user queries. User can request different time ranges $T$ and/or the number of frequent routes $K$. We need to implement map/reduce functions for all ad-hoc queries. However, this is very hard and not feasible in the MapReduce framework.

The second approach, which we adopt in this paper, is to use the MapReduce for the pre-processing step and the MongoDB document store for the query step. This approach overcomes the drawbacks of the previous one. The advantage using this approach is we only need to run MapReduce once for generating hash-based index structures to group taxi trip events by unit time. This output of MapReduce would be stored into MongoDB. The MongoDB will work for the query processing. Users request queries with various parameters in fast time with efficiency of MongoDB.

### 4.2 Data Structure

We will maintain a hash-based index structure for retrieving top-$K$ frequent routes efficiently. The unit time (one minute interval) is hashed in the index. Each hash bucket maintains a list of (route, frequent) pairs. A pair of (pickup cell id, drop-off cell id) represents the route as explained before. How to use this index is described when we explain the query processing step.

### 4.3 The Pre-processing Step

We will process taxi trip data with the MapReduce framework due to the huge volume of the data. The map phase takes the taxi trip events in New York City for the whole year 2013 and generates key-value pairs where the key is a unit time meaning for dropoff-time and the value would be the whole route for the unit time. For easier computation of the frequent route, we store the frequency counter to the route information.

**Algorithm 1:** Map function

```
input  : docid id,doc d
output: Unit time dt, pair(route, 1)
foreach Line l in Document d do
    Initialize a pickup position pp;
    Initialize a drop-off Position dp;
    // split data and get dropoff-time and pickup/dropoff positions
    triple tr(dropoff_dt dt, pp, dp) ← split(l);
    route r ← pair(getCell (tr.pp), getCell (tr.dp));
    roundUp (dt); // round up dt to nearest minute
    EMIT(dt, pair(r,1)) ;
end foreach
```

Algorithm 1 describes the map function. Each line of the input document d has 17 attributes; however we deal with only 5 attributes: dropoff-time, pickup-longitude, pickup-latitude, dropoff-longitude, dropoff-latitude. After read a line, we initialize $pp$ as pickup position and $dp$ as drop-off position. The position consists of values of longitude and latitude. We will generate a triple $tr$ which consists of dropoff-dt $dt$, $pp$, $dp$ from the input event line. The $dt$ is dropoff-time of the event. Then, we obtain the route $r$ by getting the grid cell identifiers for pickup/drop-off locations. We round up the time $dt$ to the nearest minutes for grouping each event to one minute unit time. Finally, the key of the output becomes dropoff-dt $dt$ which is rounded up and the value of output becomes a pair of route $r$ and 1 as frequency.

**Algorithm 2:** Reduce function

```
input  : Unit time dt, pair(route,frequency) p
output: Unit time dt, hashmap(route, frequency)
Initialize hashmap hm as new Association Array;
foreach pair(route,frequency) rf in p do
    hm[rf.route] ← hm[rf.route] + rf.frequency;
end foreach
EMIT(dt, hm);
```

Algorithm 2 explains the reduce function which takes the ordered and grouped data. We initialize a hashmap *hm* whose key is the route and the value is the frequency of that. The elements of *hm* is the aggregation of the frequency for each route. After aggregation, we emit a pair (*dt*, *hm*) as final output of the pre-processing step.

### 4.4 The query processing step

**Algorithm 3:** Top-k Query processing

**input** : end-time $Q$, a number of routes $K$, duration $T$
**output**: top $K$ frequent routes

```
Initialize hashmap hm as new Association Array;
Start time st ← Q - T;
Range time rt ← range(st,Q);
foreach datetime dt in rt do
    List of Route Frequency hm[dt] ← findValue(dt);
end foreach
List of Route Frequency lrf ← Aggregate(hm); // Aggregate all route frequency to lrf
Initialize priority queue(route,frequency) pq;
foreach pair(route,frequency) dt in lrf do
    pq.add(lrf);
end foreach
return pq.limit(K); // return the first K results of pq
```

The top-$K$ frequent query processing is explained in Algorithm 3. A user query includes three parameters : (1) an end time $Q$ (2) a number of frequent routes K (3) a duration T. The start time st of the query is automatically calculated extract $T$ minutes from $Q$. Then, MongoDB searches the hash-based index is done by using start time and end time. Each bucket of index contains the list of pairs. When we find the pair, we insert it into the candidate priority queue. The queue automatically maintains K elements. Finally, the elements of the queue are returned to the user.

Fig. 8 shows an example of the query processing. First, a user specifies three parameters $Q$ as “2013-04-14 04:56:00”, $K$ as 10 and $T$ as 5 minutes to the query processor for obtaining top-$K$ frequent routes (①) Then, the start time of this query automatically computed as “2013-04-14 04:52:00”, before 5 minutes from $Q$ (②). Next, MongoDB finds the index with the range of start time and end time (③) and gets the list of route

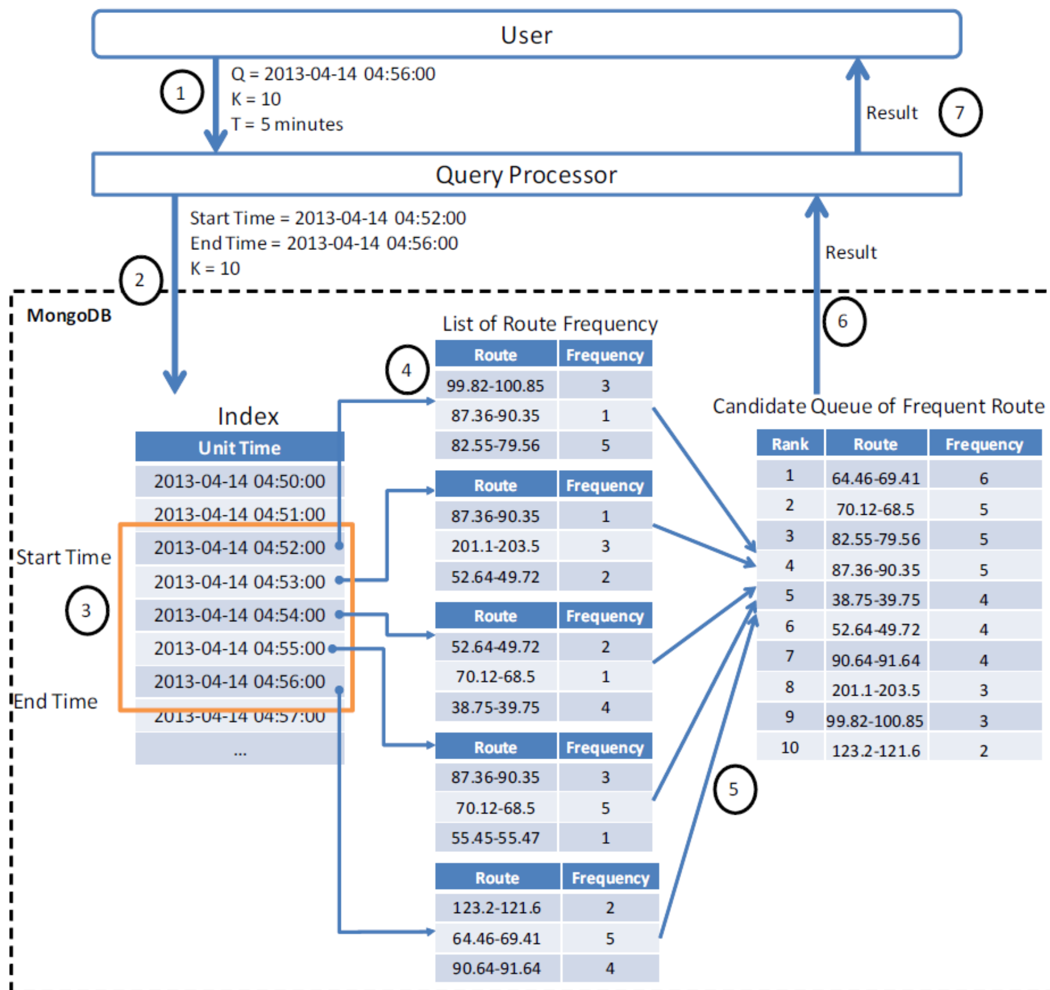
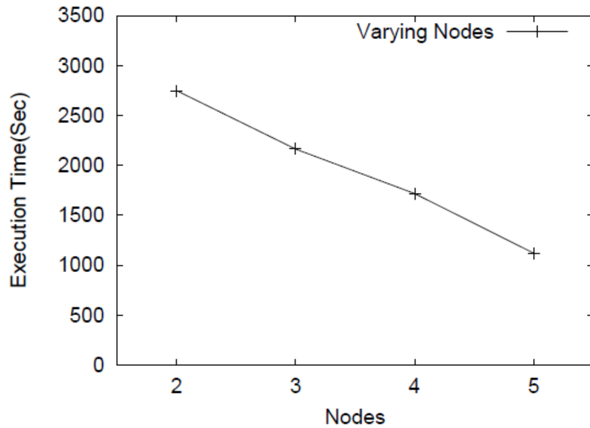
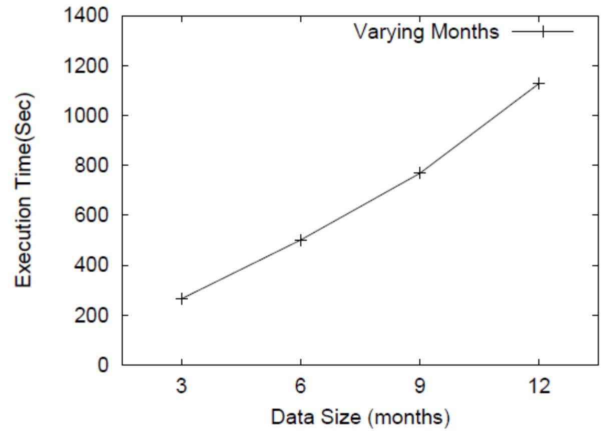



Fig. 8. Query Processing Flow




(A) Varying the Nodes




(B) Varying Data Size

Fig. 9. Extracting the Frequent Routes

frequency from selected index (④). The list has the route and frequency of each route in index. The lists are aggregated as candidate queue of frequent route (⑤). The query processor gets results from the candidate priority queue and returns Top-10 frequent route during last 5 minutes from 2013-04-14 04:56:00 to the users (⑥), (⑦).

As an exceptional case, a user specified the query time as non-rush hour time such as midnight. In this case, the total number of frequent routes is less than the value of K. We decide to add "NULL" to the last frequent route if the number of frequent is less than "K".

## 5. Experimental Evaluation

In this section, we will present an experimental evaluation of the performance of the proposed system. Experimental results with real world datasets of New York City demonstrate the feasibility and efficiency of our system.

### 5.1 Experimental Setup

The experiments are conducted on the top of a Hadoop cluster which consists of 5 machines : one master node and four slave nodes. Each machine runs a 64-bit Ubuntu 12.04 as their operating system, equipped with Intel Core 2 Quad CPU @2.66Ghz, 2 GB Memory. The analysis algorithm is implemented in Hadoop 1.2.1 using JAVA version 1.7 and the query processing is implemented in MongoDB 2.6.10[1]. We used the 2013 data of New York City's taxi trip data from FOIL[14]. The size of each month and total size is about 1 GB and 12GB respectively.

Our target query is finding the top-$K$ frequent routes during the last $T$ minutes. Thus, a user query includes two parameters : K and T. The value of K corresponds the number of routes which will be returned to the user and the value of $T$ means the duration from the given query time to search the routes. We randomly choose 10 the dates for the queries 10 and execute them for the experiment. The reported execution time is the average value.

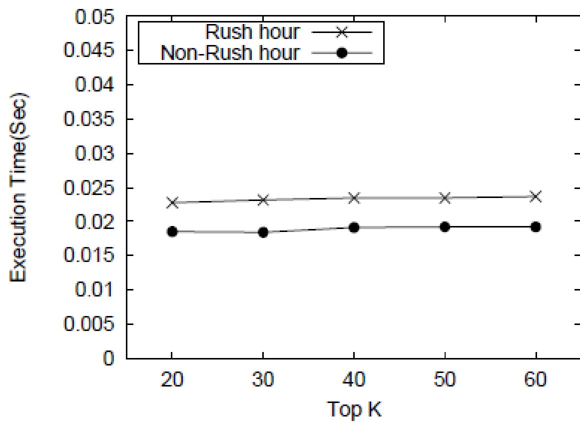
### 5.2 Experimental Results

In this subsection, we describe the experimental results of our system which is a combination of a MapReduce framework and MongoDB NoSQL database. The experiments are conducted in two steps : a pre-processing step and a query processing step.

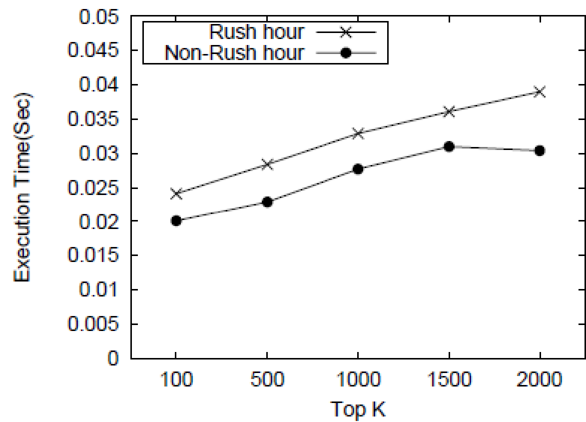
1) Pre-Processing Phase : In this experiment, we measured the execution time for pre-processing phase, which extracts frequent routes information from raw taxi trip events.

Since our approach is implemented in a distributed way, we first fixed the size of data to 12 months and varied the number of nodes. Fig. 9(A) shows the result. As expected, we observe that the distributed approach in multiple nodes can achieve a scalable ability to handle huge volumes of data efficiently. The running time of 5 nodes is almost half of that of 2 nodes.

Next, we fixed the number of node to 4 and varied the data size by increasing months in steps of 3 months. As shown in Fig. 9(B), the execution time increased with the data size. The months of data related with the size of data and are one of key factors for performance.




(A) small K 〈 100




(B) large K 〉 100

Fig. 10. Query Processing Time by Varying K

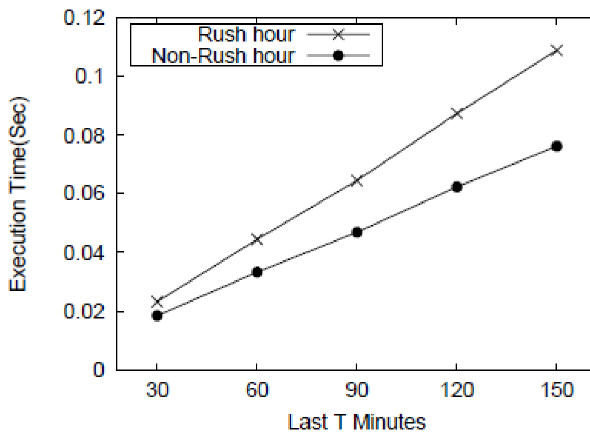



Fig. 11. Query Processing Time by Varying T

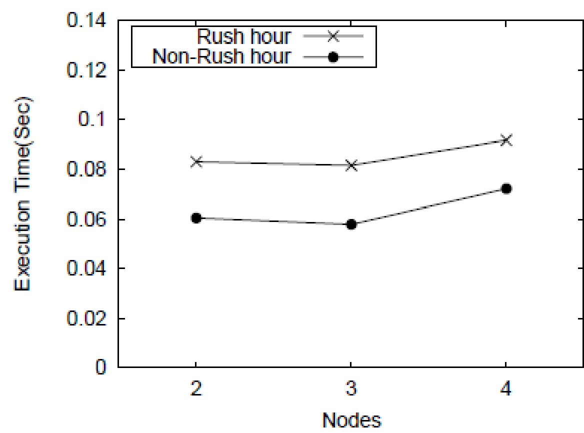



Fig. 12. Distributed Query Processing Time

2) Query Processing Phase : In this experiment, we validate the performance of top-$K$ query processing for computing frequent routes. And we want to compare with the query execution time of rush hour and non-rush hour in New York City. Generally the rush hour is corresponding to 6-10 AM and 4-8 PM[16]. The remaining times of date could be regarded as a non-rush hour.

a) Varying K : We measured the execution time by varying the number of frequent routes denoted by K. We fixed the range of query time as last 30 minutes. The experimental result is an average running time of 10 query times. Fig. 10(A) shows the results when the values of K varied from 20, 30, 40, 50, 60. First, we observed that the query is executed quickly. Another observation is that the execution time very slightly increased with the values of K. This is mainly due to the efficiency of the hash-based index structure. Also, we observed the execution time of the rush hour is more than non-rush hour.

b) Last $T$ Minutes : In this experiment, we measure the execution time by varying the length of duration specified by $T$ minutes. We fixed the value of K to 30, randomly chose a query time 10 times and compute the average running time. The duration $T$ varied from 30 to 150 in steps of 30. Fig. 11 shows the result. As expected, the execution time linearly increased with the value of $T$ since we need to check more hash buckets for computing frequent routes.

c) Distributed query processing : In this experiment, we also measure the execution time by varying the number of nodes to see the effects of the number of nodes. We created 2, 3, and 4 nodes for sharding clusters of MongoDB and set the value of K to 100 and the duration is 60 minutes. The hash-based index is

separated based on drop-off time. Fig. 8 shows the result of experiment. Due to the efficiency of the hash-based index, the running time does not have much correlation with the number of nodes.

## 6. Conclusion

In this paper, we designed and implemented a top-$K$ query processing system for IoT generated taxi trip events. We adapt the MapReduce for extracting frequent routes from raw taxi trip data and the MongoDB NoSQL database for processing top-$K$ queries. Experimental results show that our system supports the top-$K$ queries efficiently with real taxi data of New York City. We believe that the proposed system could also track a large number of vehicles in modern urban areas due to its distributed processing.

## References

[1] 2014 Taxicab Fact Book [Internet], http://www.nyc.gov/html/tlc/html/about/about.shtml/, 2014.

[2] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva, "Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips," *Visualization and Computer Graphics, IEEE Transactions on*, Vol.19, No.12, pp.2149-2158, 2013.

[3] Z. Gui, H. Yu, and Y. Tang, "Locating traffic hot routes from massive taxi tracks in clusters," 2014.

[4] D. Zhang, T. He, S. Lin, S. Munir, J. Stankovic, et al., "Dmodel: Online taxicab demand model from big sensor data in a roving sensor network," in *Big Data (BigData Congress), 2014 IEEE International Congress on,* IEEE, pp.152-159, 2014.

[5] G. Pan, G. Qi, Z. Wu, D. Zhang, and S. Li, "Land-use classification using taxi gps traces," *Intelligent Transportation Systems,* IEEE, 2013.

[6] J. Lee, I. Shin, and G.-L. Park, "Analysis of the passenger pick-up pattern for taxi location recommendation," in *Networked Computing and Advanced Information Management, 2008. NCM'08. Fourth International Conference on*, Vol.1. IEEE, pp.199-204, 2008.

[7] Y. Wang, P. Chen, L. Cheng, and H. Tong, "A solution of traffic problems based on mapreduce," in *Proceedings of the International Conference on Information Engineering and Applications (IEA) 2012*. Springer, pp.749-757, 2013.

[8] NoSQL [Internet], http://nosql-database.org/, 2015.

[9] Z. Wei-ping, L. Ming-Xin, and C. Huan, "Using MongoDB to implement textbook management system instead of MySQL," in *Communication Software and Networks (ICCSN), 2011. IEEE 3rd International Conference on,* IEEE, pp.303-305, 2011.

[10] ORACLE, MySQL [Internet], https://www.mysql.com/, 2015.

[11] MongoDB [Internet], https://www.mongodb.com/, 2015.

[12] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM workshop on Scientific cloud computing*, ACM, pp.13-20, 2013.

[13] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of ACM*, Vol. 51, No.1, pp.107-113, Jan., 2008.

[14] Chris Whong, FOILing NYC's Taxi Trip Data [Internet], http://chriswhong.com/open-data/foil nyc taxi/, 2014.

[15] DEBS2015, grid cell in New York City [Internet], http://www.debs2015.org/call-grand-challenge.html/, 2015.

[16] Rush Hour [Internet], https://en.wikipedia.org/wiki/Rush hour/, 2015.

**Fadhilah Kurnia Putri**

e-mail : fadhilahkp@pusan.ac.kr

2014년 인도네시아 ITS대학 전산학(학사)

2015년～현 재 부산대학교 빅데이터협동과정 석사과정

관심분야 : 빅데이터 처리 및 분석, NoSQL 데이터베이스, 머신러닝

**안 성 아**

e-mail : saan@pusan.ac.kr

2014년 부경대학교 통계학(학사)

2014년～현 재 부산대학교 빅데이터협동과정 석사과정

관심분야 : 빅데이터 처리 및 분석, NoSQL 데이터베이스

**Magdalena Trie Purnaningtyas**

e-mail : magdalena.trie@gmail.com

2014년 인도네시아 Brawijaya대학 정보공학(학사)

2015년～현 재 부산대학교 전기컴퓨터공학과 석사과정

관심분야 : 무선 및 자동차 통신망

정 한 유

e-mail : hyjeong@pusan.ac.kr

2005년 서울대학교 전기컴퓨터공학부(박사)

2005년~2007년 삼성전자 정보통신총괄 책임연구원

2007년~2008년 미네소타대학교 디지털연구소 박사후연구원

2008년~2014년 부산대학교 차세대물류IT기술연구사업단 교수

2014년~현 재 부산대학교 전기공학과 교수

관심분야 : 무선 및 자동차 통신망

권 준 호

e-mail : jhkwon@pusan.ac.kr

2009년 서울대학교 전기컴퓨공학부(박사)

2009년~2010년 차세대융합기술연구원 선임연구원

2010년~현 재 부산대학교 빅데이터 협동과정 교수

관심분야 : 빅데이터 처리 및 분석, 그래프 데이터베이스, XML 문서 필터링 및 인덱싱, IoT 데이터 저장 및 관리