

# Analyzing traffic and identify a specific vehicle using advanced Map Reduce Technique

Amulya N<sup>1</sup>, Dr.Prakasha S<sup>2</sup>

<sup>1</sup> M. Tech Student, Department of Computer Network & Engineering, RNS institute of Technology, Bangalore

<sup>2</sup> Professor and Head, Department of Computer Network & Engineering, RNS institute of Technology, Bangalore

\*\*\*

**Abstract** – Huge information has developed as another period of data era and handling. Enormous information applications are required to give a great deal of advantages and comfort to our lives. Distributed computing is a famous framework that has the assets for enormous information preparing. As the quantity of cell phones is quick expanding, portable distributed computing is turning into a vital piece of numerous huge information applications. In this article, we propose a novel Map Reduce-based structure to handle geo-scattered huge information in portable cloud design. The proposed system bolsters basic and also complex operations on geo-scattered huge information, and utilizations different information collection plans to fulfill distinctive application prerequisites.

**Key Words:** Map Reduce, geo scattered, distributed computing.

## 1.INTRODUCTION

Enormous information takes many structures, incorporating messages in interpersonal organizations, information gathered from different sensors, caught recordings, et cetera. Huge information applications intend to gather and break down a lot of information, what's more, proficiently concentrate profitable data from the information. A current report demonstrates that the measure of information on the Web is around 500 billion GB. With the quick increment of cell phones that can perform detecting and get to the Web, a lot of information are created day by day. All in all, huge information has three components: extensive volume, high speed and huge assortment [1]. The Universal Information Organization (IDC) anticipated that the aggregate sum of information created in 2020 all inclusive will be around 35 ZB. Facebook necessities to prepare around 1.3 million TB of information every month. Numerous new information are created at high speed. For instance, more than 2 million messages are sent over the Web each second.

Portability administrations, for example, Google Maps and Route Seer-bad habit give advantages and

accommodation to individuals. These applications are enormous information applications in light of the fact that the informational collection size is huge and the information refresh rate is quick [2]. A lot of new portability related information are created each day, for example, video observation information gathered by top notch cameras at roadsides and intersections. Normally, the quickly generated enormous information are not transferred to a server farm on the double. Rather, the new enormous information is immediately put away in nearby servers incidentally. Past research takes a shot at huge information for the most part study effective handling strategies and scientific techniques for huge information in a bunched domain, and don't consider a geo-scattered huge information situation. The above transportation benefit in view of new and verifiable enormous information has a place with a geo-dispersed huge information situation. In this circumstance, it is a test to effectively deal with a demand for geo-scattered huge information application. Moreover, unique administration targets require distinctive complexities of operations on huge information. When all is said in done, operations on enormous information can be separated into two classifications: straightforward musical show tions and complex operations. For instance, recovery has a place with straightforward operations, while examination of video substance (in light of information mining) is a mind boggling operation. A system for efficiently handling geo-scattered huge information ought to bolster both straightforward and complex operations.

Portable distributed computing [3–5] is a rising cloud benefit display in light of versatile registering and distributed computing. As the processing capacity of cell phones expands, portable distributed computing can sort out and use calculation assets of disseminated cell phones. Another model for versatile distributed computing is known as the cloudlet-based portable cloud show. The cloudlets [6, 7] are sent close Wi-Fi get to focuses (APs) and cell base stations to give cloud benefits productively, and diminish the system fetched between portable clients and a focal cloud. In the

portable cloud architecture, there are a few situations where substantial crisp informational indexes (as a component of huge information) are created quickly and immediately put away in cloudlets, and the new information are relocated to the focal cloud occasionally. In this circumstance, if a few solicitations for enormous information applications come in, the regular strategy for transfer ing a lot of new information to the server farm is not efficient as far as correspondence overhead and reaction time. Most past research takes a shot at versatile distributed computing dis-cuss how to effectively offload errands from cell phones to a cloudlet or a focal cloud keeping in mind the end goal to spare cell phone vitality and lessen undertaking consummation time. In any case, few works examine how to use versatile distributed computing to prepare geo-scattered enormous information and upgrade reaction time to portable clients. To productively prepare geo-scattered enormous information utilizing portable distributed computing, coordinated effort among hubs is important.

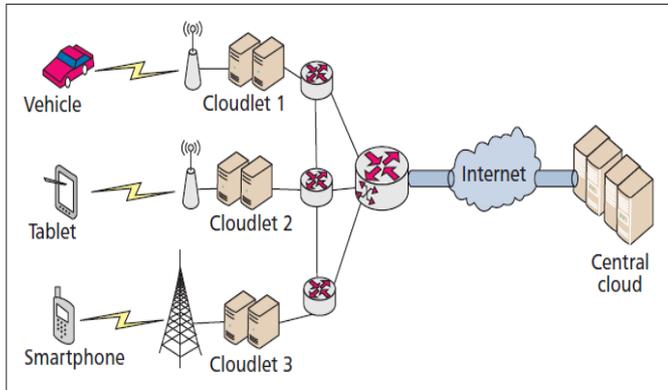
In this article, we propose a novel and adaptable system in view of MapReduce to bolster straightforward and additionally complex operations on geo-scattered huge information. The proposed structure is alluded to as the progressed MapReduce system (AMF). For a demand with straightforward operations, AMF utilizes helpful handling in the versatile cloud and MapReduce to prepare geo-scattered enormous information. To begin with, the proposed system naturally separates a challenging task into a few branch occupations as indicated by the conveyance of info information, and after that each branch employment is performed utilizing helpful preparing in the portable cloud. For a demand with complex operations, AMF extricates the required various contributions from geo dispersed huge information in parallel, and afterward total separated required numerous contributions from various cloud hubs. After the collected information are prepared by performing complex operations, AMF makes the last outcomes and sends them to the client. For complex operations, AMF utilizes diverse information conglomeration plans for various application necessities. For continuous applications, the objective is to limit the reaction time to versatile clients. For non-ongoing applications, AMF makes an exchange off between reaction time and correspondence overhead. Furthermore, the proposed collection plans depend on joint effort among cloud hubs. AMF adaptively uses Map Reduce to perform basic operations on geo-

scattered enormous information by distributed and parallel processing. In addition, AMF moves forward Map Reduce to bolster complex operations on geo-scattered huge information by accumulation plans.

## 2. When Map Reduce Meets Geo-Dispersed Big Data

Map Reduce is a software framework introduced by Google to perform distributed computation on large data sets. Map Reduce perform distributed computation on large data sets. Map Reduce [8] is a promising computing model for big data processing. The MapReduce framework has been used widely by any corporations such as Google, Yahoo, and Amazon to process big data efficiently. The main idea of the MapReduce framework is to split a large job into a number of smaller tasks, including mapping and reducing tasks, and these tasks are performed independently on different worker nodes. Before starting map tasks, input data need to be partitioned into several small data blocks of the same size ranging from 16 to 64 MB. Each data block is then assigned by a master to a worker along with a map operation. The mapper (i.e., the worker assigned to a map task) applies a map operation to compute intermediate key-value pairs. A master is in charge of assigning map and reduce tasks to workers. A map operation consists of three functions: map function, sort function and combine function. The map function can be obtained from the specific operation corresponding to a request. The sort function is responsible for sorting the intermediate values computed by mappers in order to group key-value pairs corresponding to the same key. The combine function is utilized to integrate all the intermediate values sharing the same key so that the size of the intermediate values is reduced. Then the intermediate values are partitioned into  $R$  blocks by a hash function and stored in local disks. In addition, a reduce operation includes three functions: shuffle, merge, and reduce. The shuffle function enables each reducer to pull its intermediate values from local disks. The merge function groups all intermediate values sharing the same key. The reduce function implements the requested simple operation on input data. Typical work environments for MapReduce are clustered environments in which many machines have stable connectivity, high bandwidth, and a shared file system. When all of the big data is stored in a single data center, the MapReduce framework is simple, flexible, and efficient. However, the huge amount of distributed real-time information introduces a new scenario. In the new scenario, fresh data as a part of big data stored in cloudlets are geographically separated from data in the central cloud, and migrating a large amount of fresh data to the central cloud

may cause a large delay to users. Hence, the conventional MapReduce framework for clustered environments is not



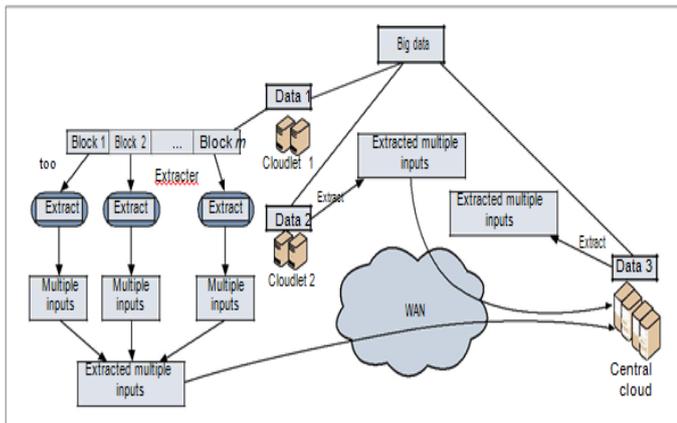
**Figure 1. Mobile cloud architecture.**

suitable for the above scenario in terms of network delay. Moreover, operations on large data sets supported by MapReduce are usually simple mathematical operations such as count, sort, and selection. At present MapReduce does not support complex operations (e.g., data mining and data analysis) on big data very well. Hence, the issue of efficiently performing complex operations on geo-dispersed big data in the mobile cloud model needs to be solved.

### 3. Mobile Cloud Architecture for Geo-Dispersed Big Data Applications

Figure 1 shows the mobile cloud architecture that is used to provide better support for geo-dispersed big data applications. The architecture consists of several cloudlets and a central cloud. The central cloud stores part of the big data, and the cloudlets have large amounts of fresh data (part of the big data), which are uploaded to the central cloud periodically to update the data set. The central cloud has sufficient computation resources to process all of the big data. However, migrating large amounts of fresh data from cloudlets to the central cloud may cause long delays. On the other hand, a cloudlet has less computation resources than the central cloud but very short communication delays to mobile users. For some geo-dispersed big data applications that require complex mathematical operations, the corresponding multiple inputs cannot be partitioned and processed by distributed and parallel computing. Currently, complex mathematical operations are not well supported by conventional MapReduce [9]. In this situation, when a request for a geo-dispersed big data application happens, migration of a large amount of data is not efficient in terms of response time. Hence, the cloudlets should be utilized to assist in performing complex operations on geo-dispersed big data and reduce response time and the size of a data

block is bigger than the size of the minimal input unit. In the mobile cloud environment, mobile devices play a key role in generating big data and requesting big data applications. Mobile devices discussed in this article include vehicles and small smart mobile terminals, including tablets, smartphones, and so on. For vehicles, there are some cases in which data collection rates can outperform the Internet, such as video surveillance in buses. At present, many buses have installed high-definition camera systems to monitor in-bus conditions. Traditionally, every bus needs a very large-volume hard drive to store video content for a few days. Then the video content is checked in an offline manner [5]. In mobile cloud architecture, cloudlets can be used to store the rapidly generated video surveillance content to implement timely video content processing and save the high cost of large-volume hard drives. In this case, preprocessing using cloudlets can reduce the communication delay for delivering a large amount of data to a central cloud for complex operations. In addition, for simple operations, distributed and parallel computing using cloudlets and a central cloud can decrease the response time of a request. This mobile cloud architecture is flexible and advantageous in supporting geo-dispersed big data applications for mobile users. First, the architecture contains two layers of clouds that are in different locations. The central cloud is usually distant from mobile users. Cloudlets are deployed near mobile users to provide services quickly. For instance, a mobile user can collect and upload surrounding real-time information to a nearby cloudlet, and then other mobile users arriving at the same or neighboring cloudlet are able to retrieve and download recent data fast. Second, cloudlets reduce the workload of the central cloud so that tasks in the central cloud can be performed faster. Third, a seamless connection between mobile devices and cloudlets can be accomplished by hybrid wireless communication technologies such as Wi-Fi and cellular 3G/4G. For example, a mobile device equipped with standard wireless interface can access to a nearby cloudlet through Wi-Fi AP or cellular base stations. complex algorithm with complete inputs. Second, a cloud performing the complex algorithm on multiple inputs should guarantee that the entire operating procedure of the complex algorithm runs correctly. Due to some inherent attributes of complex mathematical operations, partitioning multiple inputs or the algorithm into several parts for parallel computing usually does not work. Hence, given all inputs, a complex algorithm that can process the inputs and create results correctly should be performed in one machine rather than a cluster of machines.



**Figure 2. Extraction module and aggregation operations in AMF.**

#### 4. The Advanced MapReduce Framework

In this article, we propose a novel and flexible framework, AMF, based on MapReduce to process geo-dispersed big data. Different from the conventional MapReduce framework in a clustered environment, AMF focuses on supporting complex mathematical operations and efficiently performing simple operations on geo-dispersed big data. AMF combines cooperative processing in a mobile cloud with MapReduce to efficiently process geo-dispersed big data. In this article, we mainly consider the case in which partial or complete large volumes of input data are promptly stored in cloudlets. In addition, we assume that both cloudlets and the central cloud have sufficient computation resources to perform simple and complex operations. The specific method of AMF is laid out below.

#### 5. Features of Complex Mathematical Operations on Geo-Dispersed Big Data

Complex mathematical operations on normal data can be abstracted as a complex algorithm that can process multiple inputs. We discuss the features of complex mathematical operations on geo-dispersed big data in the following. First, finding multiple required inputs is critical for performing complex mathematical operations efficiently on big data because many irrelevant data are included in the original input data. In addition, the original input data that contain the multiple required inputs are very likely located in different clouds due to geo-dispersed big data. Therefore, aggregation of geo-dispersed multiple inputs needs to be accomplished in order to obtain the complete inputs. Aggregation in this article means transferring geo-dispersed multiple required inputs to a cloud that can perform the requested complex algorithm with complete inputs. Second, a cloud performing the complex algorithm on multiple

inputs should guarantee that the entire operating procedure of the complex algorithm runs correctly. Due to some inherent attributes of complex mathematical operations, partitioning multiple inputs or the algorithm into several parts for parallel computing usually does not work. Hence, given all inputs, a complex algorithm that can process the inputs and create results correctly should be performed in one machine rather than a cluster of machines.

#### 6. Performing Complex Operations in AMF

AMF mainly utilizes distributed extracting to decrease the size of data that needs to be aggregated and processed. First, multiple required inputs need to be extracted from large amounts of original input data so that the size of input data is reduced. In other words, extracting multiple required inputs means refining original input data. Due to the reduced size, the time to transfer the refined input data is less than that of the original input data. AMF employs distributed and parallel computing to extract multiple required inputs from the original input data (geo-dispersed big data). Second, the multiple inputs are aggregated to guarantee the complete inputs. Third, the requested complex mathematical operations are performed in a cloud node.

As shown in Fig. 2, AMF automatically divides the entire extraction job into several branch jobs by distributed extracting in each cloud. A branch job consists of extract operations and the original input data. Before starting the extract task,

the original input data needs to be partitioned into several small data blocks of the same size and the size of a data block is bigger than the size of the minimal input unit. Each data block is then assigned to an extractor along with an extraction operation. The extractor applies the extraction function to obtain some of the required inputs. The extraction function can be obtained from the specific request for a big data application. After the entire extraction job is done, an aggregation of the multiple required inputs needs to be performed to create the complete inputs. Once the aggregation of the geo-dispersed multiple inputs is done in a cloud, the corresponding cloud starts to perform complex mathematical operations on the inputs. As shown in Fig. 3, the complete inputs cannot be partitioned and are sent to an analyzer directly. The analyzer applies an analysis function to obtain the final results. The analysis function is an algorithm with complex mathematical operations. As shown in Fig. 4, the proposed framework employs cooperative processing in mobile cloud and MapReduce to perform simple operations on geo-dispersed big data. First, the proposed framework automatically divides a big job into

several branch jobs according to the distribution of input data, and then each branch job is performed using cooperative processing in the mobile cloud. The main idea of cooperative processing is to leverage a central cloud to accelerate processing and reduce response time. When a cloud node starts to perform simple operations on input data, MapReduce is utilized to perform computing in parallel. We divide simple operations into two categories. For simple operations in the first category, the distributed intermediate results do not need to be aggregated for further processing such as search operation. For the second category, the distributed intermediate results need to be aggregated and processed further for instance sort operation. First, we demonstrate our method of performing simple operations in the first category. Considering the fact that the data sizes of the processed results are normally small, the communication delay for delivering the final results is negligible compared to the computing time of the large amount of input data. As the intermediate results in the first category are part of the final results, the corresponding communication delay is also negligible. The proposed cooperative processing makes cloudlets allocate a number of chunks of input data to the central cloud. In other words, the total tasks are divided and processed in parallel by cloudlets and the central cloud. While a cloudlet is

transmitting input data to the central cloud, the cloudlet is also processing the input data except the data allocated to the central cloud. When both the cloudlets and the central cloud finish their own processing, the complete final results are created. Our method calculates the amount of data chunks allocated to the central cloud in order to reduce the total response time. We give notation definition in our scheme as follows. Let  $a$  denote the time to execute the simple operations on a single chunk of data in a cloudlet,  $b$  denote the time to execute the simple operations on a single chunk of data in the central cloud, and  $c$  denote the time to transfer one single chunk from a cloudlet to the central cloud. If the total amount of chunks of input data in a cloudlet is  $m$ , and the number of chunks allocated to central cloud is  $n$ , the time to process total time to transmit  $n$  chunks through a WAN and process them using the central cloud is denoted as  $T_2^p$ . Thus, for performing simple operations on input data in a cloudlet, the total response time of cooperative processing is:

$$T^c = \min(\max(T_1^p, T_2^p)).$$

In Eq. 1,  $T_1^p = (m - n)a$  and  $T_2^p = nc + nb$ . When

$$n = \frac{ma}{a + b + c},$$

the minimal value of  $\max(T_1^p, T_2^p)$  can be acquired. When big data is quickly stored in distributed cloudlets, each cloudlet independently calculates its own  $T^c$  and performs simple operations by cooperative processing. Once some intermediate results are generated in a node, they are directly transmitted to mobile users as a part of the final results. Second, we demonstrate our method to perform simple operations in the second category. Like the method in the first category, the proposed cooperative processing makes cloudlets allocate a number of chunks of input data to the central cloud to implement parallel computing. When both the cloudlets and the central cloud finish their own processing, the aggregation of intermediate results is started. Our method calculates the amount of chunks of data allocated to the central cloud and the corresponding minimal aggregation time in order to reduce the total response time. We give notation definition in our scheme as follows. Let  $G_1$  denote the data size of intermediate results corresponding to  $(m - n)$  chunks in the cloudlet, and  $G_2$  denote the data size of intermediate results corresponding to  $n$  chunks in the central cloud. Let  $T_1^a$  denote the time to aggregate intermediate results and create the final results in the

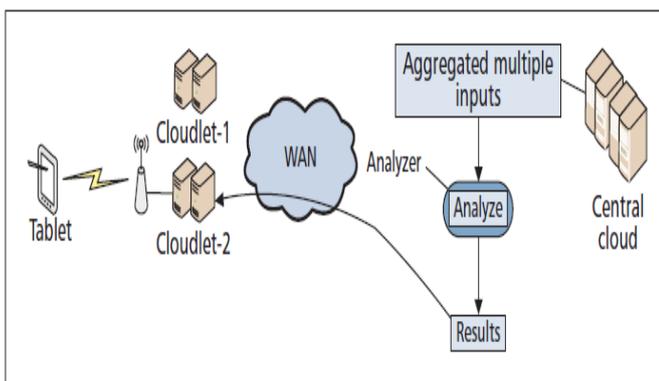


Figure 3. Analysis module in AMF.

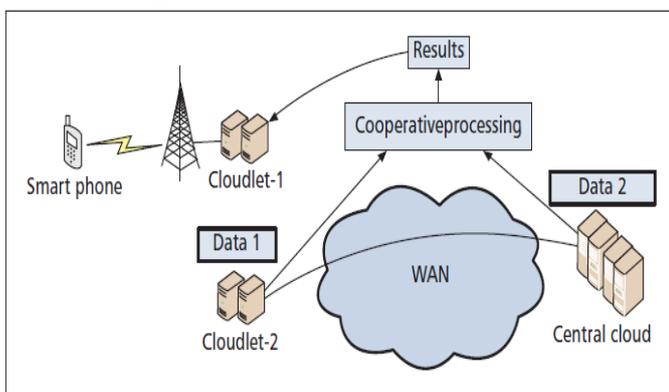


Figure 4. Simple operations in AMF.

cloudlet, and  $T_2^a$  denote the time to aggregate intermediate results and create the final results in the central cloud. For other notations, we still use the above definition such as  $a$ ,  $b$ , and  $c$ .

Considering the fact that the data sizes of the final results are normally small, the communication delay for delivering the  $(m - n)$  chunks using the cloudlet is denoted as  $T_1^p$ , and the final results is negligible.

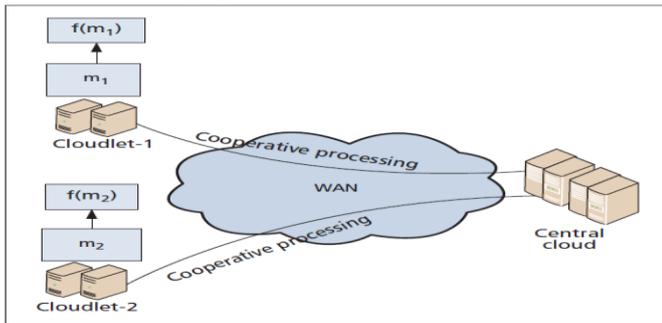


Figure 5. Response time using cooperative processing.

Equation 2 demonstrates our method of calculating the response time in the case where the complete input data is only stored in a cloudlet. So the corresponding total response time of cooperative processing is

$$T^c = \min(\max(T_1^p, T_2^p)) + \min(T_1^a, T_2^a). \quad (2)$$

In Eq. 2, the minimal value of  $\max(T_1^p, T_2^p)$  can be acquired as in Eq. 1. Let  $s$  denote the size of a chunk. Hence,

$$T_1^a = \frac{G_2 c}{s} + \frac{(G_1 + G_2) a}{s}$$

and  $T_2^a$  can also be obtained. We use a function  $f(x)$  to describe the relationship between the size of the processed results and the size of the input data, which is denoted as  $x$ . For different applications, the corresponding  $f(x)$  is different. Machine learning can be used to establish the function  $f(x)$ . In this article, we do not specifically discuss how to establish  $f(x)$ , but give a method to calculate the sizes of the processed results of input chunks  $(m - n)$  in the cloudlet and input chunk  $n$  in the central cloud (i.e.,  $G_1$  and  $G_2$ ). Therefore, through the initial values  $m$ ,  $s$ ,  $f(x)$ , and  $n$ , we can calculate the values of  $G_1$  and  $G_2$ . Then,  $\min(T_1^a, T_2^a)$  can be obtained. Thus, the total response time  $T^c$  of cooperative processing can be calculated when a request with simple operations happens. At last, we compare the response time corresponding to only using the cloudlet  $ma$  with the  $T^c$ , the processing method corresponding to the lower response time is selected and performed. For distributed input data stored in selected and performed. For distributed input data stored in several cloudlets, we demonstrate our methods to

perform simple operations in the second category as follows. We consider the case in which the complete input data is stored in two cloudlets, and the cloudlets have different input data corresponding to  $m_1$  chunks and  $m_2$  chunks. Let  $a_i$  denote the time to execute the simple operations on a single chunk of data in cloudlet  $i$ ,  $T_i^c$  the total response time calculated by Eq. 2 corresponding to cloudlet  $i$ , and  $d$  denote the time to transfer one single chunk from one cloudlet to another. For other notations, we still use the above definitions such as  $b$  and  $c$ . First, each cloudlet independently calculates its own  $T_i^c$  and  $m_i a_i$  by our above method and determines the processing method according to the respective response time. After the first step, intermediate results corresponding to input data from different cloudlets are created, and they may be distributed at different nodes due to cooperative processing. Second, according to the distribution of the intermediate results, calculate the aggregation time in each candidate aggregation node  $i$  which has partial intermediate results or is the central cloud. The calculation of total response time can be done in the central cloud or cloudlets. Before the calculation, the initial values including the function  $f(x)$  need to be shared among these computing nodes. Based on various aggregation times and processing times in different aggregation nodes, the respective total response time  $T_i^b$  can be obtained. Taking the case in Fig. 5 as an example, where the intermediate results  $f(m_1)$  and  $f(m_2)$  are created in cloudlets 1 and 2, respectively, using Eq. 2, the total response time corresponding to aggregation node cloudlet 1 is

$$T_1^b = \max(T_2^c + f(m_2)d, T_1^c) + (f(m_1) + f(m_2))a_1$$

For brevity, the total response times,  $T_2^b$  and  $T_3^b$ , corresponding to aggregation node cloudlet 2 and the central cloud, respectively, are not demonstrated in detail. At last, the processing method and the aggregation node corresponding to the response time  $\min(T_1^b, T_2^b, T_3^b)$  are performed and selected. For other cases where distributed input data is stored in several cloudlets, the computing method is the same, and the processing method corresponding to the lower response time can be determined when a request with simple operations in the second category happens. Next, we consider the case in which both cloudlets and the central cloud have a part of large volumes of input data. We assume that the complete input data is stored in two cloudlets and the central cloud, and the data size is  $m_1$ ,  $m_2$ , and  $m_3$ , respectively. First, each cloudlet independently calculates its own  $T_i^c$  and  $m_i a_i$  by Eq. 2 and determines the processing method according to the respective response time. Because the input data  $m_3$  is

pro-cessed by the central cloud, the corresponding processing time is  $m_3b$ . Second, according to the distribution of the intermediate results, calculate the total response time  $T^b$  in each candidate aggregation node  $i$  that has partial intermediate results or is the central cloud. At last, the processing method and the aggregation node corresponding to the minimal response time are performed and selected.

### 7. Efficient Aggregation Schemes

The aggregation operation is a key step in performing complex mathematical operations because the required multiple inputs are geo-dispersed. Aggregating large amounts of multiple inputs may cause long delay for users. If the size of aggregated multiple inputs is big, the transmission time and cost can be reduced by an efficient aggregation scheme, which can improve the performance of AMF. We propose two types of aggregation scheme to support real-time and non-real-time geo-dispersed big data applications, respectively. Since real-time applications require smaller response time, the corresponding aggregation scheme is referred to as a dynamic aggregation scheme, which aims to reduce response time. On the other hand, for non-real-time applications, our proposed aggregation schemes aim to finish the requested tasks in a relatively longer time but with low cost, and the corresponding aggregation scheme is referred to as planned aggregation.

In the dynamic aggregation scheme, whenever a cloud node finishes extraction operation, it transfers the extracted multiple inputs and the corresponding size to other cloud nodes involved in the entire job. When all required multiple inputs are aggregated in a cloud node, the cloud node starts to per-form analysis operation. After a cloud node finishes an analysis operation in the shortest time, it informs other cloud nodes with a state message including the original user request and final results. When a cloud node receives a state message, if it is working on the same user request with the received state message, the cloud node ends all the operations caused by the same user request immediately. Due to the different computing capability of each cloud node, the respective time to extract required multiple inputs from a large amount of data is different. The dynamic aggregation scheme reactively carries the next task when the extraction of required multiple inputs in a cloud node is finished. Hence, the minimal aggregation time can be achieved by the dynamic aggregation scheme.

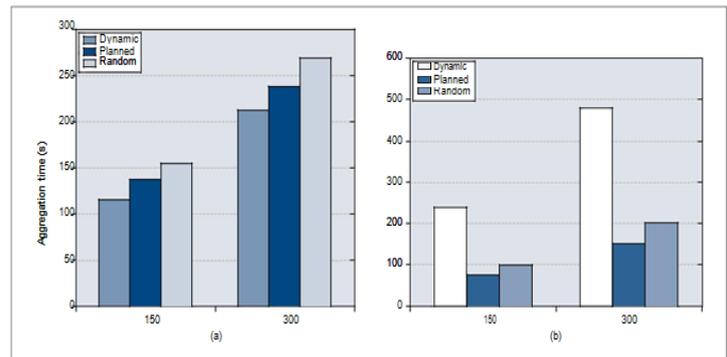


Figure 6. Simulation results of aggregation schemes: a) aggregation time; b) communication overhead.

The state message can save some communication and computation cost when the final results are obtained in the shortest time. In the planned aggregation scheme, when all cloud nodes involved in a big job finish their own extraction operation, they start the planned aggregation scheme. In other words, after the slowest extraction operation is finished, the aggregation scheme starts. First, each cloud node transfers the data size of the extracted multiple inputs to other cloud nodes. Second, each cloud node finds the maximal size of extracted multiple inputs among all cloud nodes, and the cloud node that contains the maximal-size extracted multiple inputs is selected as the aggregation node. Third, each cloud node excluding the aggregation node transfers extracted inputs to the aggregation node. The planned aggregation scheme per-forms aggregation operation with minimal communication overhead. For non-real-time geo-dispersed big data applications, we assume that the analysis time and extraction time at each cloud node are acceptable to users.

We use NS-2 for simulation. Input data is distributed in two cloudlets and a central cloud. For evaluating performance of our aggregation schemes, we simulate two cases where the size of total extracted input data is 150 MB and 300 MB, respectively. In each case, we consider two types of distribution of extracted input data. For the first distribution of case 1, cloudlet 1 finishes its extract operations in 15 s, and the extracted input data is 25 MB. Cloudlet 2 finishes its extract operations in 25 s, and the extracted input data is 50 MB. The central cloud finishes its extract operations in 35 s, and the extracted input data is 75 MB. For the second distribution of case 1, the time to extract input data is the same with the first distribution. The extracted input data in cloudlet 1, cloudlet 2, and the central cloud is 75 MB, 50 MB, and 25 MB, respectively. In case 2, the time to extract input

data is the same as in case 1. For the first distribution of case 2, the extracted input data in cloudlet 1, cloudlet 2, and the central cloud is 50 MB, 100 MB, and 150 MB, respectively. For the second distribution of case 2, the extracted input data in cloudlet 1, cloudlet 2, and the central cloud is 150 MB, 100 MB, and 50 MB, respectively. The link speed between cloudlets is 10 Mb/s corresponding to a LAN. The upstream link speed (i.e., the link from each cloudlet to the central cloud) is in the range of [1.3, 3.8] Mb/s, and the downstream link speed is in the range of [3.0, 4.1] Mb/s. We calculate aggregation time using the difference between the time when all extracted input data is aggregated in a node and the time when a request with complex operations happens. Figure 6 compares the aggregation time and communication overhead for three aggregation schemes (dynamic aggregation, planned aggregation, and random aggregation). The random aggregation scheme randomly selects a cloud node as the aggregation node when a request with complex operations arrives. In the random aggregation scheme, when a cloud node finishes its extract operations, it starts to transfer extracted input data to the selected aggregation node. The horizontal axis of Fig. 6 represents the size of the total extracted input data. The simulation results show that the dynamic aggregation scheme achieves the shortest aggregation time of the three schemes, and the planned aggregation scheme incurs less communication overhead than the others.

## 8. A Geo-Dispersed Big Data Application Based on the Proposed Framework

In this section, we consider a geo-dispersed big data application for analyzing vehicles to support an intelligent transportation system. The corresponding input data is a large amount of video surveillance data. The application is able to analyze traffic and identify a specific vehicle according to its color and license number. The video surveillance data is collected by high-definition cameras at roadsides and junctions. When a request happens, first AMF extracts the pictures from the original video surveillance data stored in several cloudlets according to the specified color of the request. Then AMF extracts the images from the extracted vehicle pictures according to the specified license number. At last, the extracted license images and the corresponding position information are aggregated and analyzed to identify the corresponding travel trajectory.

## 9. CONCLUSIONS

In this article, first we talk about the test in using the portable cloud to prepare geo-scattered enormous information. At that point we star represent a novel and

adaptable structure in light of MapReduce to bolster intricate and also straightforward operations on geo-scattered enormous information. The proposed system, AMF, utilizes parallel processing in MapReduce to remove different contributions for complex operations while having the capacity to suitably total and break down geo-scattered huge information. For basic operations, AMF adaptively uses coordinated effort among cloud hubs and MapReduce to productively handle geo-dispersed enormous information. For complex operations on geo-scattered enormous information, AMF utilizes distinctive accumulation plans to meet different application necessities. For constant applications, limiting reaction time to portable clients is accomplished. For non-ongoing applications, AMF makes an exchange off between reaction time and correspondence cost

## REFERENCES

- J. Manyika *et al.*, "Big Data: The Next Frontier for Innovation, Competition, and Productivity," McKinsey Global Inst., May 2011.
- S. Shekhar *et al.*, "Spatial Big-Data Challenges Intersecting Mobility and Cloud Computing," in *Proc. of 11th ACM Int'l. Wksp. Data Engineering for Wireless and Mobile Access*, Scottsdale, AZ, 2012, pp. 1-6.
- D. Huang *et al.*, "Secure Data Processing Framework for Mobile Cloud Computing," *Proc. IEEE INFOCOM Wksp. Cloud Computing*, Shanghai, China, 2011, pp. 614-18.
- B. Chun *et al.*, "Clone cloud: Elastic Execution between Mobile Device and Cloud," *Proc. 6th Conf. Comp. Sys.*, New York, NY, 2011, pp. 301-14.
- R. Yu *et al.*, "Toward Cloud-Based Vehicular Networks with Efficient Resource Management," *IEEE Network*, vol. 27, no. 5, Sept.-Oct. 2013, pp. 48-55.
- M. Felemban, S. Basalamah, and A. Ghafoor, "A Distributed Cloud Architecture for Mobile Multimedia Services," *IEEE Network*, vol. 27, no. 5, Sept.-Oct. 2013, pp. 20-27.
- D. Huang, T. Xing, and H. Wu, "Mobile Cloud Computing Service Mod-els: A User-Centric Approach," *IEEE Network*, vol. 27, no. 5, Sept.-Oct. 2013, pp. 6-11.
- J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, Jan. 2008, pp. 107-13.

K. H. Lee *et al.*, "Parallel Data Processing with MapReduce: A Survey," *ACM SIGMOD Record*, vol. 40, no. 4, Dec. 2011, pp. 11-20.