Big Data Based Dynamic Flow Aggregation over 5G Network Slicing

Guolin Sun[1], Bruce Mareri[1], Guisong Liu[1], Xiufen Fang[2], Wei Jiang[3,4]
[1] School of Computer Science and Engineering, University of Electronic Science and Technology of China
Chengdu, Sichuan, 611731, P. R. China
[2] School of Mathematical Sciences, University of Electronic Science and Technology of China
Chengdu, Sichuan, 611731, P. R. China
[3] German Research Center for Artificial Intelligence (DFKI GmbH), Kaiserslautern, Germany
[4] Department of Electrical and Information Technology (EIT), Technische University (TU) Kaiserslautern, Germany
[e-mail: guolin.sun@uestc.edu.cn]
*Corresponding author: Guolin Sun

Received February 16, 2017; revised April 27, 2017; accepted June 1, 2017; published October 31, 2017

Abstract

Today, smart grids, smart homes, smart water networks, and intelligent transportation, are infrastructure systems that connect our world more than we ever thought possible and are associated with a single concept, the Internet of Things (IoT). The number of devices connected to the IoT and hence the number of traffic flow increases continuously, as well as the emergence of new applications. Although cutting-edge hardware technology can be employed to achieve a fast implementation to handle this huge data streams, there will always be a limit on size of traffic supported by a given architecture. However, recent cloud-based big data technologies fortunately offer an ideal environment to handle this issue. Moreover, the ever-increasing high volume of traffic created on demand presents great challenges for flow management. As a solution, flow aggregation decreases the number of flows needed to be processed by the network. The previous works in the literature prove that most of aggregation strategies designed for smart grids aim at optimizing system operation performance. They consider a common identifier to aggregate traffic on each device, having its independent static aggregation policy. In this paper, we propose a dynamic approach to aggregate flows based on traffic characteristics and device preferences. Our algorithm runs on a big data platform to provide an end-to-end network visibility of flows, which performs high-speed and high-volume computations to identify the clusters of similar flows and aggregate massive number of mice flows into a few meta-flows. Compared with existing solutions, our approach dynamically aggregates large number of such small flows into fewer flows, based on traffic characteristics and access node preferences. Using this approach, we alleviate the problem of processing a large amount of micro flows, and also significantly improve the accuracy of meeting the access node QoS demands. We conducted experiments, using a dataset of up to 100,000 flows, and studied the performance of our algorithm analytically. The experimental results are presented to show the promising effectiveness and scalability of our proposed approach.

Keywords: Dynamic aggregation; big data; IoT; flow aggregation;

1. Introduction

In smart cities, the main objective of IoT is to have the ability to uniquely recognize, signify and access things anytime and anywhere in the Internet. The applications of interest range from systems supporting urban mobility and its safety such as smart parking, traffic congestion, intelligent transportation systems, monitoring or optimizing assets and critical infrastructures in cities for structural health, smart lighting and smart roads.

With the increasing popularity of wireless sensor networks, the amount of sensory data is increasing at an explosive rate, which up to now exceeds a few petabytes annually. The major challenge for mobile networks is to provide the much needed throughput for the increased traffic demand. The ever increasing speeds and volumes of transmission in 4G and 5G networks, present great challenges for traffic flow management. The first challenge is QoS for the required traffic networks needs to adapt and provide the capacity to handle this rapid growth of data efficiently. The second is that the traffic requires a lot of resources, both in bandwidth for flow transmission and the adaptability to determine the best method to transmit the traffic.

5G networks must support a variety of very diverse use cases with different requirements for latency, throughput, and availability [1]. The dynamic network slicing concept offers a way to optimize 5G networks. The current “one-size-fits-all” approach to wireless networks for all use cases and services to every device everywhere is no longer viable. To meet the future performance expectations, individual network segments, Radio Access Network (RAN), transport, core network and edge cloud, which were formerly treated completely separately, must be re-evaluated. Their performance must be adapted and coordinated to deliver a specific flow, for a specific user, at specific time.

With this in mind that the high amount of traffic from sensors and other IoT devices are generated, special consideration is given to ensure end to end QoS. But in high volumes, a lot of resources are consumed in signaling and processing the flows, thus increasing latency. Solutions need to be implemented in order to ensure efficiency of these small mice flows. Taking into account the versatile attributes of each network slice, traffic requires effective and efficient management. These scalability issues motivate us to use some way or form of traffic reduction in this case flow aggregation. Several aggregation methods have been proposed in [2] [3] in a static way, which is not sufficient enough to guarantee large amounts of IoT mice flows and elephant flows aggregations. Due to this intrinsic relevance, there are many works that propose dynamic and adaptive aggregation schemes [4]. Others propose big data solutions to improve performance of networks analysis [5]. We can put forward the idea of having a dynamic flow aggregation methodology that is using the parallel processing big data applications to manage large scale wireless network data as proposed in [6].

Our main goal of this work is to propose and evaluate a algorithm of dynamic flow aggregation under a MapReduce framework on a Hadoop platform. We propose a Parrallel KNN algorithm in this paper, which is called as ParkNNO, to handle a massive number of mice flows and elephant flows in a dynamic and proactive way, so as to ensure that traffic is sufficiently aggregated based on the device. In MapReduce module, flow analysis and large-scale traffic data are taken as the input of Parrallel KNN classifier, which will generate the classification results. With the output of MapReduce module and the current data observed, we can achieve real-time aggregation results. The experimental results demonstrate that the proposed approach has a significant performance gain by speeding up the classification

process and reducing the storage requirements. More importantly, with reasonable execution time, the classification performance can be significantly improved by flow aggregation information.

The rest of paper is organized as follows: In Section 2, we review the previous works in the field of traffic aggregation. Section 3 we define our proposed dynamic aggregation and give an overview of the algorithm implemented in MapReduce.Section 4 presents experimental results in term of performance evaluation, and we compare the dynamic and static aggregation algorithms by means of the latency and compare how it performs on a parallel nodes and make our conclusion in Section 5.

2. Related Work

Flow aggregation is vitally important for network management, accounting and performance. The idea of flow aggregation has been explored to address the scalability issue of OpenFlow (OF)-based SDN. Flow aggregation combines a set of individual flows with same forwarding and performance requirements into an aggregated flow, which can be processed as one flow for admission control as well as forwarding decisions. Flow aggregation can mitigate the processing load at the controller, reduce communication overhead between the controller and switches, and limit the sizes of flow tables at switches, enhancing the scalability of OF-based SDN. For example, DevoFlow framework is proposed in [7] to achieve scalable flow management and a MiceTrap mechanism is reported in [8] as a scalable traffic engineering scheme with aggregation of a large number of short-live flows.

In the previous works [9], flow aggregation is performed at the device level, where each network device performs isolated aggregations based on resource availability and efficiency of the device. This might not be an efficient way of flow aggregation in the 5G scenario of massive terminals, where a lot of devices are interconnected. Some aggregation architectures only perform path based aggregation, which might introduce higher latency in terms of different QoS needs of various traffic generated by a single source[10]. In [11], they provide a more refined approach for aggregation with a classification algorithm that classifies traffic based on its qualities on an Openflow network and reduces network load significantly, but still we need a more robust and scalable methodology to handle aggregation for traffic generated by massive terminals, efficiently. 5G slicing networks introduces the concept of networking slicing [12] [13]. There are some proposed models which focus on classifying and measuring QoS requirement and data traffic of M2M applications. In one approach [14], a data aggregation method has been proposed that relies on aggregating data from several M2M devices at the Packet Data Convergence Protocol (PDCP) layer of the Relay Node (RN), using a data traffic slicing algorithm which enhances QoS by efficient utilization of the 5G radio resources for M2M and the principle idea of Priority Queue (PQ) approach.

When analyzing a large volume of traffic data for detailed statistics for a long-period or on a large-scale network, it is not easy to handle Tera or Peta-byte traffic data with a single server. As distributed parallel processing schemes have been recently developed, they could be usefully applied to analyzing big traffic data, especially in 5G network [15]. Thus, in this paper, we use open source Hadoop based MapReduce software framework of the cloud computing platform for a large-scale network data processing. Previously MapReduce has been used for Internet traffic measurement and analysis of network traffic activities, analyzing large volumes of data from various network nodes [16] [17]. It has been proven as a reliable cloud computing platform to process traffic and weblog analysis. It has been

suggested as an efficient solution for processing the flow data collected from switches and performs near real-time computations, which in our case are vital for massive flow aggregation.

3. The Proposed Approach

Generally, Internet traffic measurement and analysis are executed on a high-performance central server. Popular tools such as *tcpdump* or *Coralreef* are usually run on a single host to capture and process packets at a specific monitoring point. *Flowscan*, is widely used to generate traffic statistics. However, when we anatomize traffic in a large-scale network, we are often confronted with hard challenges of handling a huge amount of traffic data for processing and management. For example, when ISPs monitor traffic in a nation-wide network consisting of hundreds or thousands of routers capable of exporting *Cisco NetFlow* data, it is not easy to compute traffic statistics from many large flow files in short time. In order to lessen the volume of continuously streaming flow data, packet sampling or flow aggregation techniques are used. Otherwise, after processing packet traces, we leave only the statistics information results.

Different traffic aggregation strategies have been implemented in enterprise networks, and with the growth of network infrastructure, scalability for traffic aggregation also grows. In [2], a static aggregation strategy is proposed where a common identifier is used to aggregate common traffic across a single path. This method identifies the micro-flows as a single unit meta-flow throughout the network path. One downside of this method is that each device has its own QoS requirement. Aggregation using a single factor cannot guarantee effective QoS for all of the different devices handling traffic. We propose to aggregate traffic dynamically using a controller based aggregation within an overall network. For each network device, we guarantee QoS by trying to aggregate traffic flows based on a combination of various flow tuple characteristics, and considering the device QoS preferences.

Here, we detail our proposed MapReduce-based nearest neighbor approach for massive flow aggregation using term frequency for traffic analysis. By capturing incoming traffic flow metadata, we analyze the flows to generate the aggregated flows. We aim to design and deploy a framework to detect, analyze and aggregate flows. We consider a 5G network with various devices with specific QoS requirements that need to be met. We consider the large amounts of traffic generated by the networks and all need end to end QoS requirements guaranteed. The major concern is to ensure network resources efficiency by managing a large number of flows in the network. Flow aggregation needs to carry out instantaneously as the traffic is going through the network and should be dynamic to the requirements of the network nodes handling the traffic.

3.1 Network Architecture

Our SDN architecture consists of a controller and a Hadoop platform connected on the control plane. The controller is connected to all the access nodes in the network and manages them through an open flow control protocol. The cloud of sensors with massive terminals generates the traffic within our network, and an IoT gateway manages traffic for each cloud. The BTS units are the radio interfaces connected to the IoT gateways. The V-SGW (Serving gateway) within the core network originates and terminates signals and data streams from the BTS. It is also be used as a central switch for inter-BTS connectivity. The edge of network, Internet Gateway Router, is to bind traffic IP addresses to destinations. **Fig. 1** shows the

devices in a network topology that will benefit significantly from flow aggregation of data through the data plane (i) IoT Gateway, content based aggregation on traffic will help in mitigating the processing of high volume mice flows, with diverse protocols, connectivity models and energy profiles generated from the cloud of sensors. (ii) Base Stations are the air interface channels that transmit traffic based on QoS slicing of traffic. The flow aggregation based on QoS will ensure differentiated services. (iii) The V-SGW transmits traffic through port allocation within the core network. The port based flow aggregation will ensure load balancing within the core network. (iv) The access router connected to the Internet will aggregate based on path or destination. The 4-level devices are all connected to a controller, which is able to monitor and control the device operations, and the controller is connected to the MapReduce platform for decision making.

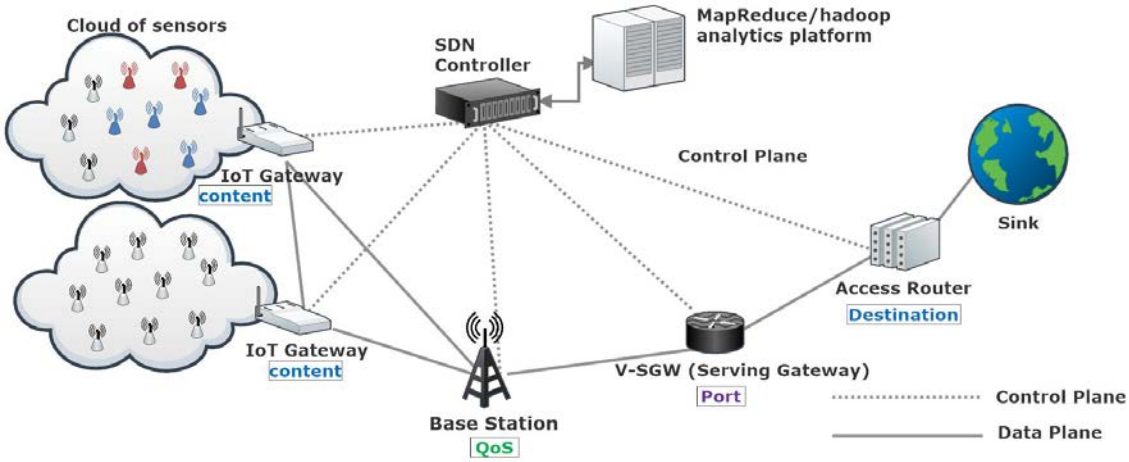


Fig. 1. The proposed big data architecture

3.2 Protocol Design

A system model for flow aggregation in OF-based SDN is illustrated in **Fig. 2**, and then we apply the model to determine the amount of network resources required by a new arrival flow to meet its aggregation requirement. The aggregation decision for the flow is made based on the determined policy requirement of each device.

The network protocol is designed with the control plane and data plane to facilitate our methodology. Our approach will combine a big data analysis engine and an SDN controller to perform parallel flow analysis and aggregation. The generated result sets will be used to determine the flow tables on the devices in the network. We shall be using an improved classification algorithm that has been modified to work with parallel MapReduce methodology. Each Openflow device is equipped with a flow collector. The flow data are captured and saved into files associated with each flow-exporting device. Collecting details of each flow will be analyzed by MapReduce to generate outputs and determine decision on the data plane. The MapReduce engine will be taking into consideration two vital elements to ensure that our algorithm is efficient. (i) The traffic flow data tuples and (ii) The device characteristics, as our key parameters to allow dynamic aggregation.

The MapReduce module is located at the controller with overall network (see section 3.3). All of the aggregated individual flows will be treated as one flow by the device on the forwarding path. The aggregated flow table will be implemented based on the group feature defined in OpenFlow specification [18]. Virtual tables and group tables can be used along

with flow tables to control packet forwarding in the devices. In our scenario, we consider a single flow analysis and aggregation, while there typically are multiple aggregated flows at each device.

The flow collector sends new flow details to the MapReduce on the control plane, where it analyzes various flow details and generates a clustered output. The result is determined for flow aggregation. Individual policy of flow aggregation decision is made based on the current device preference and flow characteristics. If the flow is accepted, the flow manager on the controller will send flow table update information to the processing module in the switch, where bandwidth and buffer space will be allocated for forwarding packets of the newly accepted aggregated flows.

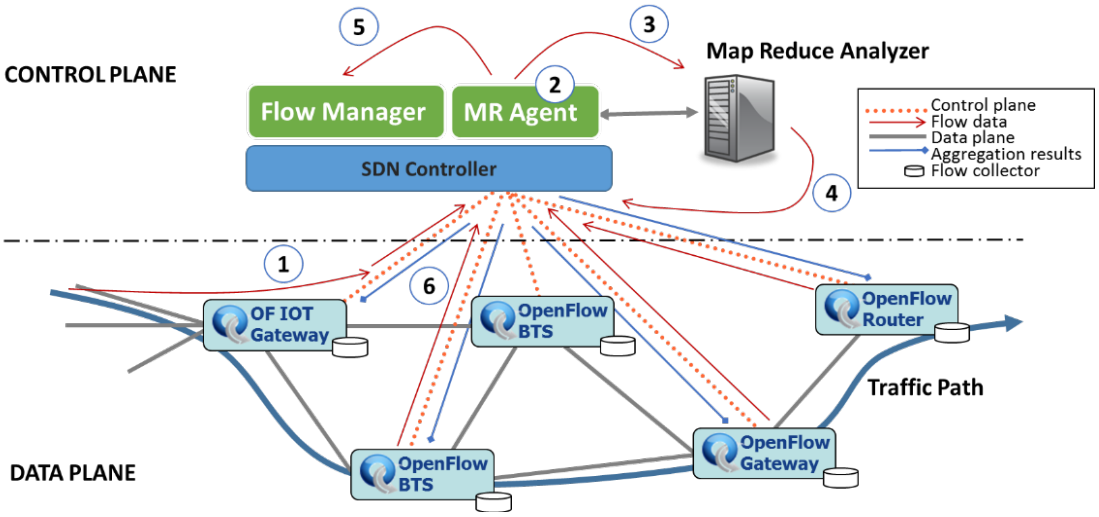


Fig. 2. The proposed protocol design

Our protocol works as shown in **Fig. 2**, with six steps in more detailed procedure. (1) Each open flow device is enabled with a flow collector. The traffic flow details are captured by a flow collector and sent the flow metadata to the controller. The metadata information includes IP destination, content type, QoS level, and the device originating traffic, even packet-size. (2) A MapReduce agent at the controller will capture, sort and process the received metadata from the flow collector, which is ready to be exported to the MapReduce platform. (3) The sorted data is arranged and the MapReduce agent exports the converted data into MapReduce readable format on HDFS (Hadoop File System) to be processed in the analyzer. (4) MapReduce performs analysis, taking into account the device preference characteristic, to define what kind of policy it should action. If a device is an IoT gateway, the flow aggregation policy will be based on content prioritized aggregation to mitigate signaling burden on the air-interface. (5) Once the Analyzer has completed analysis and a new result set is generated composed of aggregated flow results and their specific device origin, they are sent to the MapReduce agent. The MapReduce Agent will send the aggregation result sets as a policy result for the flow manager to decide the actions on the flow tables. (6) The flow manager will use the result as flow table rules, which will be used to generate the flow table instructions. The instructions are sent to the relevant devices on the data plane through the open flow protocol.

Fig. 3 shows the decision framework connecting the control plane and data plane modules. There are two important modules: *MapReduce module* (MRM) and *Online traffic aggregation module* (OTAM). The MRM module takes analy

sis of device preference and flow metadata as input to build our classifier. For the accurate traffic flow classification (see Section 3.3), it particularly focuses on fast finding the nearest neighbors of a traffic sample through speeding up the classification process and reducing the storage requirements on a distributed Hadoop platform. The OTAM module aims to generate the aggregation results in parrallel with the current traffic flow data and the classification results of our ParkNNO classifier obtained from the traffic flow data, and reduce the computational cost and memory consumption of big traffic flow data processing on a parallel MapReduce framework.

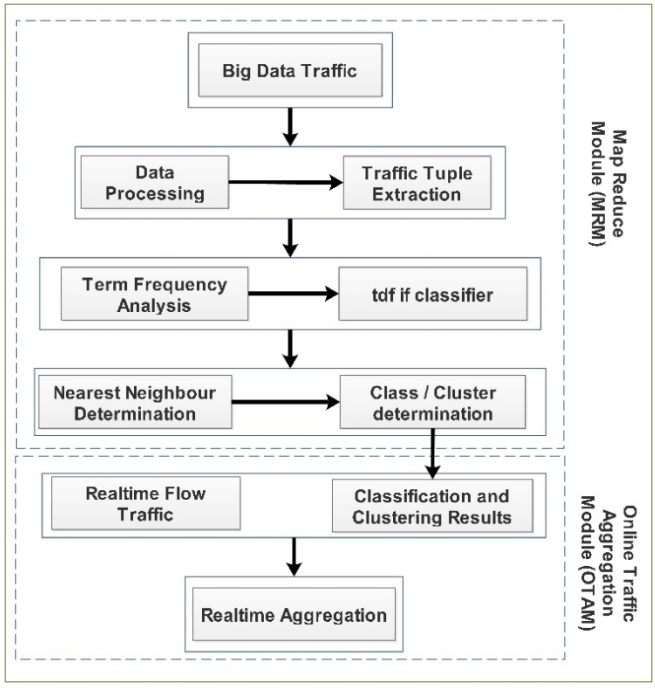


Fig. 3. Traffic Aggregation Decision Framework

3.3 The proposed algorithm on MapReduce

Here, we describe the implementation of our classifier following a MapReduce procedure to improve the classification process and reduce the storage requirements for high volume traffic flow aggregation. To improve the efficiency and scalability, we modify the standard KNN classifier to run in a parallel Mapreduce environment and breakdown the algorithm to Mapreduce jobs. We call our modified KNN classifier as ParkNNO classifier. The multiple MapReduce jobs, split from KNN, are carried out in the Map, Combine, and Reduce phases by the corresponding functions, respectively. **Fig. 4** illustrates the computation flowchart of KNN on MapReduce and its data flow of MapReduce with an example of 4-nearest neighbors. After the Map, Combine and Reduce tasks, we can utilize current traffic flow data based on the selected k nearest neighbors outputted from MapReduce jobs. To generate aggregated traffic policies, as described above, the Mapper, Combiner, and Reducer functions are designed to reduce computational costs and save memory for classification. The number of Mappers relies on input splits, whereas the number of Combiners and Reducers is equal to the group of keys, which is generated from the mapping process.

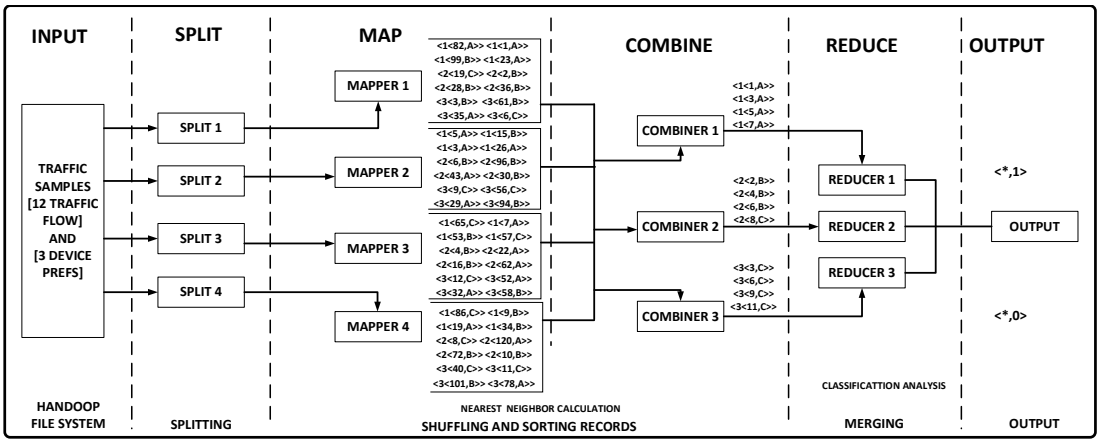


Fig. 4. Flowchart of Our Parallel KNN on MapReduce

3.3.1 KNN classifier on MapReduce

In our approach, we use a modified KNN although standard KNN is a widely applied as a non-parametric NN approach[19]. The *k* nearest neighbor-based approach has a superior classification performance with three significant advantages: (i) it doesn't need a complicated training procedure, (ii) minimal or no risk over fitting of parameters, and (iii) it is ideally capable of processing a huge amount of classes [20]. It commonly makes classification decision on the basis of closest training samples in the feature space [21]. The classifier is presented on the MapReduce framework to enhance the accuracy, efficiency and scalability of traffic flow aggregation, by facilitating the discovery and integration of correlational information between traffic flows, into the classification process[22] [23].

According to KNN, we define x as the data record and y is the corresponding class label. The main objective of k-nearest neighbor classifier is to discover set of k objects in the training set that are similar to the objects in the test group. The KNN algorithm calculates the similarity between each testing sample $te = (x^{'},y^{'})$ and all the training samples $(x,y) \in Tr$ to determine its nearest-neighbor list, Tr_{te}. Where the state space Tr is defined as all the traffic attributes from all the flows combined from training samples of each flow component. Each flow component includes a set of n attributes. The testing sample is the initial flow data to be used to determine its nearest neighbor. Once the nearest-neighbor list is obtained, the testing sample is classified based on the majority class of its nearest neighbors by the Majority Voting approach (see Eq. (3)). In our scenario each flow with n attributes will be classified and filtered using our algorithm to calculate its nearest neighbor, which is calculated by observing which flows have more similar n attributes, once this list is generated it is then filtered using a majority voting approach to ensure flows with the highest number of similarities are classified together. The KNN uses the following search parameters; (i) it defines a state space, in our case a list of all the flows attributes. (ii) it measures the similarity distance between x and each component of y in terms of text documents, using a Euclidean distance, given by

$$d_i(x,y) = \sqrt{\sum_{j=1}^{n}\left(x_j - y_j^i\right)^2} \tag{1}$$

where, x_j is the value of the j^{th} attributes in the current data vectors and y_j^i is the value of the j^{th} attributes of the i^{th} components. $d_i(x,y)$ is the distance between the current data and the i^{th} components in the state space.

In our MapReduce scenario, we find similarity given two flows, d_a and d_b represented by their term vectors $\overrightarrow{t_a}$ and $\overrightarrow{t_b}$ respectively. Using the $tfidf$ value as term weights, $w_{t,a} = tf\ idf\ (d_a, t)$, and $w_{t,b} = tf\ idf\ (d_b, t)$, substituting $x_j = w_{t,a}$ and $y_j^i = w_{t,b}$, from the previous equation (1), we define the Euclidean distance of the two documents as

$$D_E\left(\overrightarrow{t_a}, \overrightarrow{t_b}\right) = \left(\sum_{t=1}^{m} \left|w_{t,a} - w_{t,b}\right|^2\right)^{1/2} \quad (2)$$

where, $D_E\left(\overrightarrow{t_a}, \overrightarrow{t_b}\right)$ is the distance between the current data and the i^{th} flow components in the state space, $w_{t,a}$ is the value of the j^{th} attributes in the current data vectors and $w_{t,b}$ is the value of the j^{th} attributes of the i^{th} flow components, and m is the number of attributes or terms that is set $T = \{t_1, \ldots, t_m\}$. (iii) Each neighbor has the same influence on the classification by the *Majority Voting approach*, thus making the classifier sensitive to the choice of k. Because we will be having multiple aggregations of flows based on nearest neighbour we aim to make the aggregations more granular and reduce huge aggregation clusters. In order to reduce the impact of k, we implement the *Distance Weighted Voting scheme* in our classifier, and the class labels for aggregations can be determined by

$$y' = \arg max \sum_v \quad (x_i, y_i) \in Tr_{te}\, I(v = v_{yi}) \quad (3)$$

where, $\omega_i = \frac{1}{d(x', x_i)^2}$ is the weight of each nearest neighbor x_i according to its distance, v is a class label, v_{yi} is the class. And I(.) is an indicator function the returns the value 1 if the argument is TRUE and 0 if the argument is FALSE. When k = 1, the classifier assigns a testing traffic flow into the class of its nearest training sample which in our case is a cluster of matching flows attributes.

3.3.2 Mapper Function

Algorithm 1 *Mapper Function* ()

Input: <*key*, *value*>
key: the record id of traffic flows
value: the value of the tuples
Output: <key_1, $value_1$>
key_1: the record id of the testing samples, *id*
$value_1$: vector (the calculated distance, *d*, and the selected class, *c*)

```
1: for j = 0 to m do // for each equipment sending flow data  id classification
2:         for i = 0 to n do
3:          //these are the traffic samples, i, and the extracted category labels from Flow tuples, F_Tr
4:          F_Tr = GetFlowLabel (i);
5:                 for all p ∈ Traffic flow samples do
6:                    //each traffic sample, p from specific device category
7:                    d = parkNNOFunction (p,i);
8:                    // Eqs. 1
9:                    Context.write(id, vector (d, F_Tr));
10:                end for
11:        end for
12: end for
13: return <id, <d, c>>;
```

Fig. 5. Mapper Function

The *Mapper function*, (see Algorithm 1) receives each line of the flow sets as a different *key-value* pair, then goes through each traffic flow and computes the similarity (or distance) between each pair of incoming data points and existing data points, $te = (x',y')$ and all the training samples $(x,y) \in Tr$ and finally outputs the intermediate results, <*id*, <*d*, *c*>>;, which form the input to the *Combiner function*. That is, each produced pair is composed of the corresponding record id emitted as the key, and the calculated distance and the selected class, <*d*, *c*>, emitted as the value.

3.3.3 Combiner Function

The *Combiner function*, (see Algorithm 2) is implemented to reduce data shuffled between the Map and Reduce tasks and reduce computational complexity. It is used to perform the determination task of the local *k* nearest neighbors. Each *Combiner function* receives the set of <*id*, <*d*, *c*>> pairs with the same key (i.e., record id), and then sorts the key-value pairs by the distance in ascending order. According to the sorted pairs, only the *k* key-value pairs with the smallest distance will be output to Reducer. That is, the selected pairs are confirmed as the nearest neighbors. Obviously, the intermediate data would be reduced through the following rate equation:

$$R = \left(1 - \frac{k}{N_{Tr}}\right) \times 100\% \tag{4}$$

where, *k* is the number of the selected nearest neighbors, N_{Tr} is the number of all the traffic records in the state space.

Algorithm 2 *Combiner Function* ()

Input: <key_1, $value_1$>
key_1 : the same *key* of Map output for each Combiner
$value_1$: the corresponding value with the same *key* of Map output
Output: <key_2, $value_2$>
key_2: *id* of the output mapped flows from the combiner to be used in the reducer
$value_2$: <*d*, *c*>
1: //based on proximity calculations by the distance (*d*) we arrange all flow entries in an ArrayList to sorted by the *key-value* pairs
2: **for** all *key* and *value* **do**
3: ArrayList.add (vector(<*d*, *c*>));
4: Sort (ArrayList);
5: //Select *k key-value* pairs of flow entries with the smallest distance & output to the Reducer
6: Context.write(*id*, ArrayList.get(*k*));
7: **end for**
8: **return** < key_2, $value_2$> pairs;

Fig. 6. Combiner Function

3.3.4 Reducer Function

Finally, the *Reducer function* (see Algorithm 3 in **Fig. 7**) is designed to determine the global nearest neighbors. it emits the Combiner output (i.e., the *k* key-value pairs with the

smallest distance) as an input. Next, the Distance Weighted Voting scheme (Eq. 2) is employed to determine the class label. Then, the reducer judges whether the classification result is accurate and accordingly returns the value 1 or 0 to indicator function. Finally, the classification result will be sent to the main program and the compatibility value will be computed. Then, a cluster *id* is assigned to the flow.

Algorithm 3 *Reducer Function* ()

Input: < key_2, $value_2$>
key_2: the key of Combiner output
$value_2$: the value of Combiner output
Output: < key_3, $value_3$>
Key_3: *id*
$value_3$: traffic aggregation cluster id for proximity flows
1: Add the *key-value* pairs to the ArrayList for the sorting operation
2: **for** all *key* and *value* **do**
3: ArrayList (vector (<*d*,*c*>));
4: Sort (ArrayList);
5: New ArrayList result;
6: //Add *k* samples to result
7: result.add (*id*, ArrayList.get(*k*));
8: //Determine the class label of testing sample using Eq. 2: and return 1 or 0 to indicator function by judgement
9: **for** all in ArrayList (parkNNO(result))
10: Context.write (*id*, C_{id});
//Send the classification results to the output with similar flows having being appended an unique identifier to flow of similar this is used to now determine the flows are in a unique cluster
11. **end for**
12: **end for**
13: **return** <key_3, $value_3$> pairs;

Fig. 7. Reducer Function

4. Performance Evaluation

This section reports the experimental results, which are used to evaluate the performance of the proposed approach in terms of practicability, efficiency and scalability. The experimental setup and evaluation metrics are introduced, then different aggregation methods are tested on the same data sets, and finally the results are analyzed in detail. For the efficiency and scalability, we adopt two well-accepted metrics: speedup and sizeup (See section 4.6 and 4.7).

We conduct simulations studies through evaluating our scheme, with various criteria to observe its performance. In each simulation scenario, we assume ideal requirements and we assume we have all the traffic at hand in the simulator and the aggregation algorithm is working. The users will send the traffic to the network and the MapReduce cloud will capture the traffic characteristics from the controller that is listening to moving traffic. In our simulations, we focus on performance of aggregation, time latency and network propagation. Each simulation has been repeated at least 10 times to get good statistical results.

4.1 Evaluation Setup

The test bed environment, is based on a Hadoop platform on a desktop machine with Intel Xeon (R) E7-4820 2.00GHz CPU (dual core) and 4.00GB RAM. All of the experiments are performed on Ubuntu 12.04 with 5 Hadoop 1.2.1 virtual machines, each with 2 pseudo nodes. The JDK 1.6.0. Floodlight and Mininet environments is set up. We use MapReduce test bed on Hadoop environment.

Sample traffic flows are generated for performance evaluation based on a Markovian Additive Process to ensure random distribution of characteristics. We use 4 Tuples (Content, Qos, Port, Path) to generate the 100,000 flows characteristics. To ensure random distribution of the sequence of 4 tuple categories, mixed flow sizes are considered for mice flow traffic and elephant flow traffic. The packet sizes are limited to the 4 10B, 150B, 10Kb, 150Kb for mice flows and elephant flows.Before the beginning of experiments, we need to consider a few observations.

- Mapreduce will consume data in form of text files, so the traffic load or traffic data will be converted from generated pcap files into text files using a java pcap to txt converter.
- Random traffic is generated using a python network simulator to generate customized network traffic streams. The input is a list of defined parameters. The output is packets with various attributes and payloads for mice and elephant flows.

In this performance evaluation, we try to observe the effectiveness of our proposed solution. First, we conduct experiments to compare our algorithm to a simple sort algorithm. Second, we compare aggregation latency on the control plane and the data plane. The third experiment will show our proposed algorithm can perform aggregation based on device preferences dynamically. In the end, for further verifying the efficiency and scalability, we evaluate the 2Ss characteristics (i.e., Speedup, and Sizeup [24] [25]) of our classifier.

4.2 Algorithm Evaluation

Table 1. Variables for Algorithm Evaluation

Parameters	Setting Values
Number Of Flows	Dynamic (10k~100k)
Aggregation Algorithm 1	Dynamic Parknno Algorithm
Aggregation Algorithm 2,3	Static, Non-aggregation Algorithm
Traffic Generation	Random Traffic
Traffic Tuples	4 Tuples (Content, Qos, Port, Path)
Aggregation nodes	1 node
Simulator	MapReduce

This experiment is conducted on a MapReduce platform, we analyze over 100,000 flows, using 4 Tuples (Content, Qos, Port, Path) that constitute the traffic variables. We try to evaluate the amount of aggregation done on the flows by comparing our algorithm to a simple sort clustering algorithm. The static algorithm is preconfigured to cluster or aggregate the flows using the best effort or highest aggregation matches by ranking. Our dynamic algorithm will take into considerations for multiple characteristics to provide various options to aggregate flows, thus having varied aggregation based on the tuples characteristics, while the static algorithm only considers a general aggregation scheme based on a general nearest neighbor clustering methodology.

This experiment aims to show that the reduction of flow count may lead to better flow handling with aggregated flows being served. By dynamically increasing the amount of flows being processed, we are able to determine the aggregation output and compare the result sets. We observe in **Fig. 8** the aggregation algorithm significantly reduces the flow count by up to a factor of 30%.

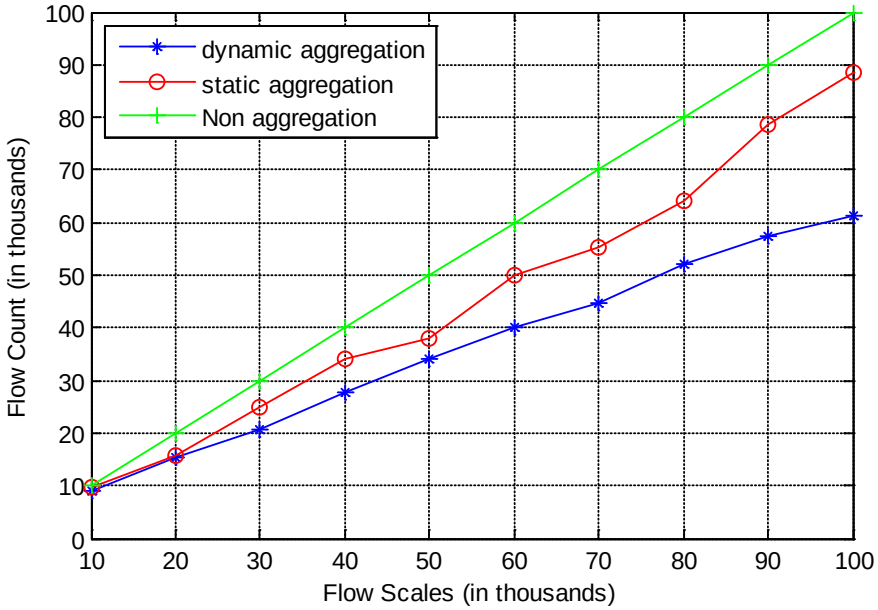


Fig. 8. Flow Aggregation Comparison

4.3 Aggregation Latency Evaluation

Table 2. Variables for Algorithm Latency

Parameters	Setting Values
Number Of Flows	Dynamic (10k-100k)
Traffic Payload	10Bytes, 150Bytes, 10Kbits, 150Kbits
Algorithm used	Parknno
Traffic Generation	Random Traffic
Traffic Tuples	4 Tuples (Content, Qos, Port, Path)
Transmission rate	2.5Mbps
Aggregation nodes	2 nodes
Simulator	Python Network Load Simulator
Simulator 2	MapReduce

In this scenario, we consider that all the traffic are either mice flows or elephants flows with a constant payload size and check how to monitor the aggregation latency. We consier the packet size for mice flows of signaling and voice traffic with a minimum of 10Bytes and a maximum of 150Bytes. We take into account elephant flows of h.264 packet size with a minimum of 10Kbits and a maximum of 150Kbits. The control plane latency is defined as the time taken to process the incoming non-aggregated flows and generate the result of aggregated flows on the analysis process. The data plane latency is the delay in transfer of the original non-aggregated flows from initial.

Firstly we conduct the simulation on the control plane, we use a flow collector java application. It collects the traffic files, converts the data and assembles it in text files to be readable in MapReduce. We consider the traffic details from the flow parameters, as input, the application converts the data into MapReduce readable format. Flow data is saved into files associated with each flow characteristic. We make an assumption that the processing time of the files accounts for control plane processing latency. We consider the processing of these traffic parameters between the control plane and the initial MapReduce platform as the processing latency for the files.

In **Fig. 9**, there is not much significant latency change in terms of processing time taken by the different flow scale in thoughsands of flows. On the control plane processing, packet load size is not considered when processing on the MapReduce. The processing time will depend on the processing load and other operational interferences.

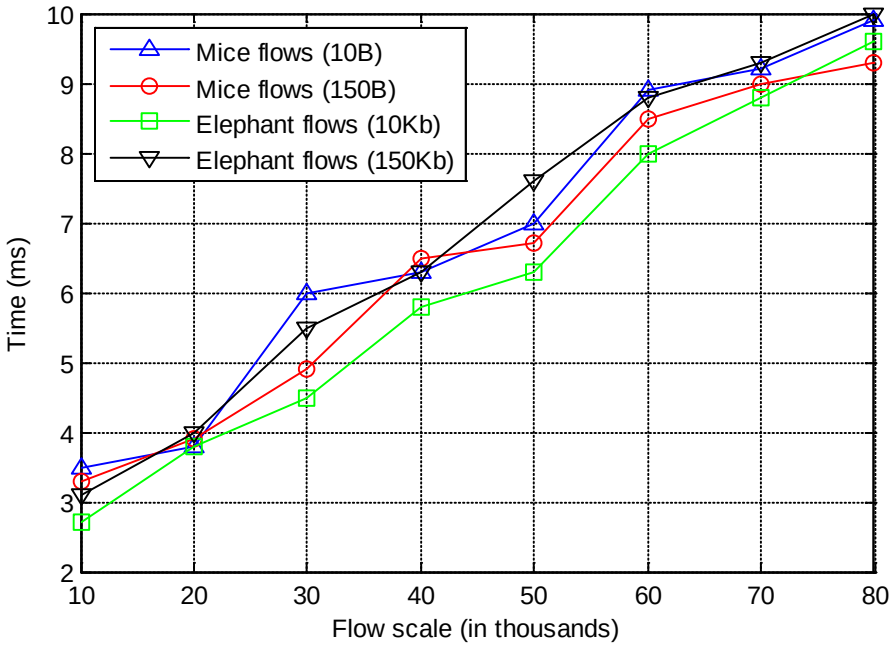


Fig. 9. The Processing Time on Control Plane

In the second part, the same payload parameters are generated with the packet sizes and we run through our python simulator with our aggregation algorithm. Python simulator will help us craft out network traffic, we input the parameters to generate the traffic packets, and we input the packet generation range of parameters. We define a range for the four tuples (Content, Qos, Port, and Path) and packets will be generated based on the characteristics of the 4 tuples. Once the packets are generated the several streams at different rates, we define a user script with parameter preferences to simulate each network node aggregation We limit the amount of transmission rate at a maximum of 2.5Mbps for the data processing of the payloads on the network simulator. We observe in **Fig. 10.** that it takes more significant time to perform aggregation for the heavier payload flow.

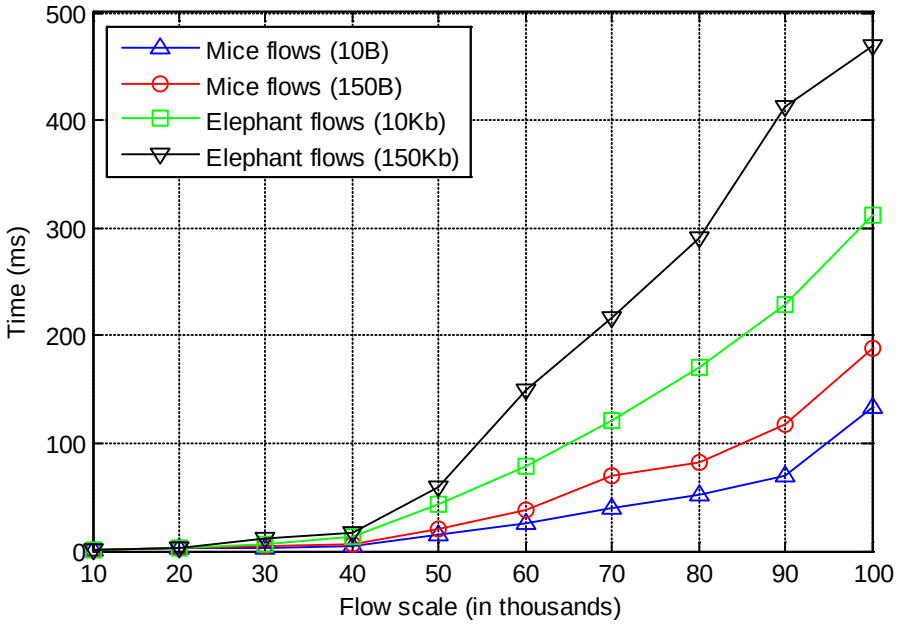


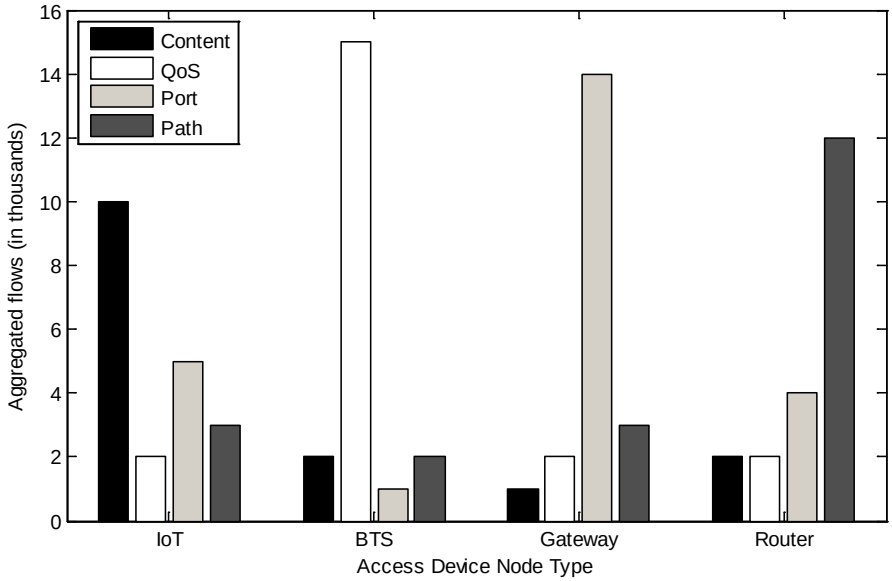
Fig. 10. The Processing Time on Data Plane

4.4 Dynamic Aggregation Evaluation

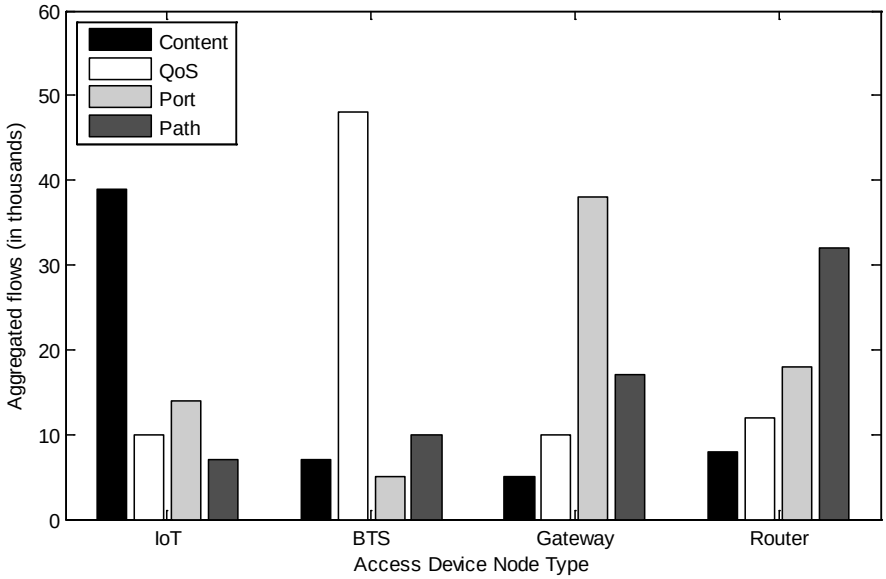
Table 3. Variables for Dynamic Aggregation Evaluation

Parameters	Setting Values
Number of Flows	Dynamic (20k, 40k, 70k, 80k)
Algorithm	ParkNNO
Traffic Generation	Random Traffic
Traffic Tuples	4 Tuples (Content, Qos, Port, Path)
Aggregation Nodes	3 nodes
Simulator	MapReduce

In this scenario, we want to do experiment on how our dynamic aggregation will perform based on each network node characteristics. We simulate device preferences to determine the aggregation policy on each network node. With the increased number of traffic flows, we compare what is the aggregation performance on the similar traffic sets. We run the proposed aggregation scheme on 3 MapReduce nodes in order to observe how it works on a parallel scenario. We want to observe the aggregation ratios as the flows increase. **Fig. 11** shows a comparison for different traffic flows set, 2000 flows in (a) and 7000 flows in (b).



(a) 20,000 flows



(b) 70,000 flows

Fig. 11. Aggregation with various preferences over network devices

From the above experiment results in **Fig. 11**, we can determine that our algorithm has a good performance in terms of dynamic aggregation based on device preferences, considering the same flows being aggregated based on device preferences. We observe higher amounts of aggregation pivoting on each device characteristics if flow scale is increased from 2000 to 7000 flows.

4.5 Speedup Evaluation

Speedup metric measures how much the parallel working algorithm is processing data faster than the corresponding sequential algorithm does not on a MapReduce platform, which is defined as

$$Speedup = \frac{T_1}{T_p} \quad , \tag{4}$$

where, T_1 represents the sequential execution time of the algorithm on one node for traffic flow aggregation using the given data set, and Tp denotes the parallel execution time of the algorithm for solving the same issue using the same data set on a MapReduce cluster with p nodes. We conduct simulations for evaluating our dynamic ParkNNO algorithm, with various criteria to observe its performance, in each simulation scenario. It is assumed that we have ideal requirements with all the traffic at hand in the simulator and the aggregation algorithm works correctly. The users will send the traffic to the network and the MapReduce cloud will capture the traffic characteristics from the controller. To validate our algorithm, we perform experiments iteratively with up to 4 nodes and varying the amount of flows from 10-70,000 flows, observe how our algorithm performs, and see how it performs in a parallel MapReduce environment. It can be observed in Fig. 12. that the proposed ParkNNO has significantly good speedup performance over the big data platform with large datasets and is near linearly increasing with the increased number of nodes. It significantly outperforms with all of data sets.

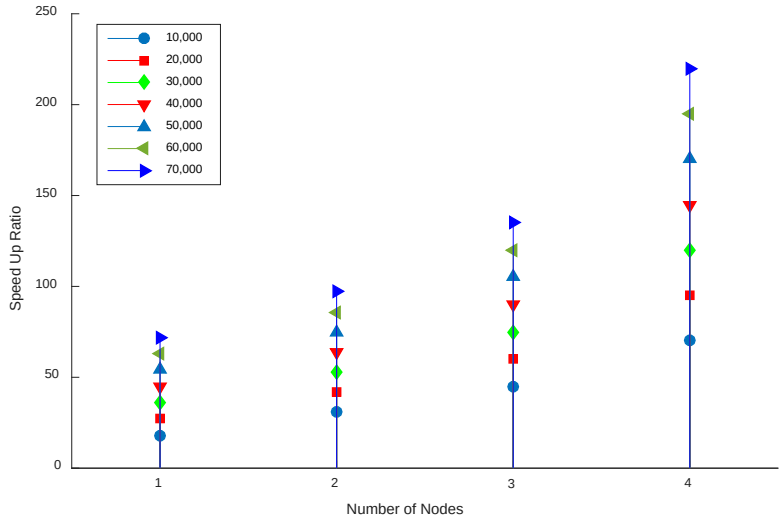


Fig. 12. SpeedUp

4.6 Sizeup Evaluation

Sizeup metric validates how much longer the parallel algorithm takes to perform aggregation on a given fixed number of nodes, when the size of the data flow set is larger than the original data set. Which is defined as

$$Sizeup = \frac{\widetilde{T_x}}{T_x}, \tag{5}$$

where, T_x denotes the execution time of the algorithm for processing the original data set on the given fixed number of nodes, and $\widetilde{T_x}$ represents the execution time of the algorithm for coping with x-times larger sized data sets on the fixed number of nodes . Here, the aim of the

sizeup analysis is to keep the number of nodes *p* constant and incrementaly grow the size of flow set *x*, in order to evaluate the variation of execution time. For our experiment we conduct simulations studies with up to 4 available distributed nodes in order to evaluate its sizeup performance. The experiment is conducted on a cluster with fixed number of nodes, from 1 node to 4 nodes, and increasing the amount of traffic flows and recording the executions times for the increased data sets, the experiment is repeated for each node cluster from 1 to 4 and the executions times plotted as shown in **Fig. 13**. It is observed that the proposed algorithm has a good sizeup ratio with an increasing traffic data sets. However, we should also take into account communication costs between the increasing number of nodes, which might also account for some latency in processing time.

$$Scaleup = \frac{Sizeup}{p} \tag{6}$$

In addition to *sizeup*, **scaleup metric** for the algorithm can also be measured from the above experiment, it is defined as the average sizeup per node. Which is calculated by From the experiment results, as shown in **Fig.13**, we determine that ParkNNO under the MapReduce framework has significantly good ***Sizeup*** and ***Scaleup*** metrics and achieves aggregation more efficiently with an increasing node scale and flow scale to run parallel transaction. And it can be further improved and implemented in a real world scenario.

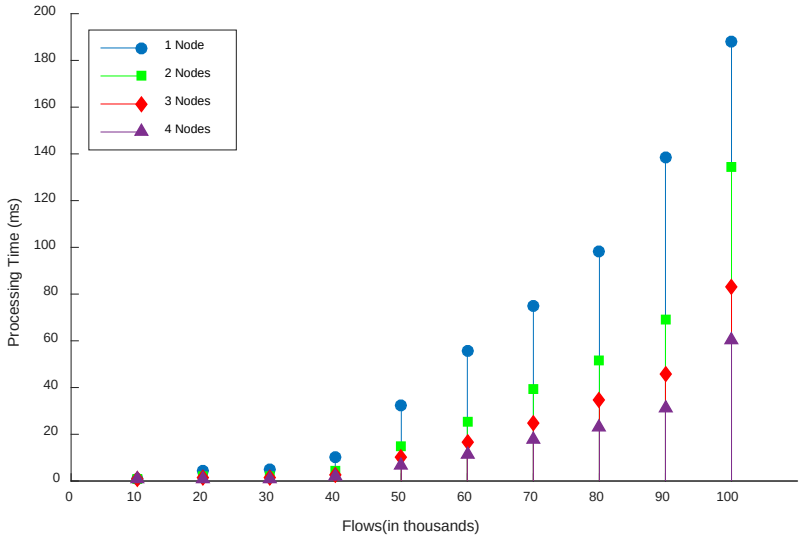


Fig. 13. SizeUp

5. Conclusion

In this paper, we focused on the dynamic aggregation of huge traffic flows and thus proposed a new MapReduce-based nearest neighbor approach for traffic flow aggregation methodology on a Hadoop platform. In order to lower memory usage and reduce the computational costs of large-scale calculations, we integrated a parallel aggregation framework composed of the MRM module and the OTAM module. In particular, to enhance the robustness of realtime applications with very large traffic samples, a modified parallel k-nearest neighbor optimization classifier, ParkNNO, was built to model traffic flow data in MRM, and an aggregation method was put forward to generate aggregated traffic in OTAM. Furthermore, we evaluated the proposed approach in terms of practicability, efficiency and

scalability. It is still an open issue on how to aggregate flows appropriately and needs to be further studied as the topic of big data. Our future works will investigate the behavior of multiple clustered traffic, analyze 5G traffic statistical characteristics and improve the aggregation architecture.

Acknowledgment

This work is supported by the Fundamental Research Funds for the Central Universities, under grant no. ZYGX2014J060, the Science and Technology Planning project of Sichuan Province, China, under grant no. 2016GZ0075 and the ZTE Innovation Research Fund for Universities Program 2016.

References

[1] Y. Yuan and X. Zhao, "5G: vision, scenarios and enabling technologies," *ZTE Communications*, vol. 13, no. 1, pp. 3-10, Mar. 2015. Article (CrossRef Link)

[2] Dolzer, Klaus, Wolfgang Payer, and Markus Eberspacher., "A simulation study on traffic aggregation in multi-service networks," in *Proc. of the IEEE Conference on ATM on High Performance Switching and Routing*, 2000. Article (CrossRef Link)

[3] Li, Xu, Jaya Rao, and Hang Zhang, "Engineering Machine-to-Machine Traffic in 5G," *IEEE Internet of Things Journal*, vol. 3, no. 4, pp. 609-618, 2016. Article (CrossRef Link)

[4] Hu, Yan, Dah-Ming Chiu, and John CS Lui, "Entropy based adaptive flow aggregation," *IEEE/ACM Transactions on Networking*, vol. 17, no.3, pp. 698-711, 2009. Article (CrossRef Link)

[5] Imran, Ali, Ahmed Zoha, and Adnan Abu-Dayya, "Challenges in 5G: how to empower SON with big data for enabling 5G," *IEEE Network*, vol.28, no. 6, pp. 27-33, 2014. Article (CrossRef Link)

[6] Jardak C, Oldewurtel F. "Parallel processing of data from very large-scale wireless sensor networks," *ACM International Symposium on High Performance Distributed Computing, HPDC 2010*, Chicago, Illinois, USA, pp.787-794, Jun. 2010. Article (CrossRef Link)

[7] Curtis A R, Mogul J C, Tourrilhes J, et al., "DevoFlow: scaling flow management for high-performance networks," in *Proc. of ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Toronto, On, Canada, pp. 254-265, August. 2011. Article (CrossRef Link)

[8] R.Trestian, G.-M. Muntean, and K. Katrinis, "MiceTrap: Scalable traffic engineering of datacenter mice flows using OpenFlow," in *Proc. of the 2013 IEEE Symposium on Integrated Network Management (IFIP 2013)*, pp. 904–907, May 2013. Article (CrossRef Link)

[9] Guo W, Mahendran V, Radhakrishnan S., "Achieving throughput fairness in smart grid using SDN-based flow aggregation and scheduling," in *Proc. of IEEE International Conference on Wireless and Mobile Computing, networking and Communications*, pp.1-7, 2016. Article (CrossRef Link)

[10] Velan, Petr., "EventFlow: Network flow aggregation based on user actions," *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Istanbul, Turkey, 2016. Article (CrossRef Link)

[11] Khalili, Ramin, et al., "Reducing State of OpenFlow Switches in Mobile Core Networks by Flow Rule Aggregation," in *Proc. of International Conference on Computer Communication and Networks 2016*, pp.1-9, 2016. Article (CrossRef Link)

[12] Sama, Malla Reddy, et al., "Reshaping the Mobile core network via function decomposition and network slicing for the 5G era," in *Proc. of IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2016. Article (CrossRef Link)

[13] Katsalis, Kostas, et al., "5G Architectural Design Patterns," in *Proc. of 2016 IEEE International Conference on Communications Workshops (ICC)*, pp.32-37, 2016. Article (CrossRef Link)

[14] Lee, GM, "Data Traffic Model in Machine to Machine Communications over 5G Network Slicing," in *Proc. of the International Conference on Developments in eSystems Engineering (DeSE 2016)*, Liverpool, September 2016. Article (CrossRef Link)
[15] A. A. Chandio, N. Tziritas, and C.-Z. Xu, "Big-data processing techniques and their challenges in transport domain," *ZTE Communications*, vol. 13, no. 1, pp. 50-59, Mar. 2015. Article (CrossRef Link)
[16] R. Cibin, K. P. Sudheer, I. Chaubey, "Sensitivity and identifiability of stream flow generation parameters of the SWAT model," *Hydrological Process.*, vol. 24, no. 9, pp. 1133-1148, Apr. 2010. Article (CrossRef Link)
[17] Anjali P P, Binu A. "Network Traffic Analysis: Hadoop Pig vs Typical MapReduce," *Computer Science & Information Technology*, 2013. Article (CrossRef Link)
[18] Open Networking Foundation, OpenFlow Switch Specification Version 1.1.0 [White paper] Dec. 2011. Article (CrossRef Link)
[19] Smith B L, Williams B M, Oswald R K., "Comparison of parametric and nonparametric models for traffic flow forecasting," *Elsevier Transportation Research Part C: Emerging Technologies*, vol. 10, no. 4, pp. 303–321, Aug. 2002. Article (CrossRef Link)
[20] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," *IEEE Trans. Parallel Distribution System*, vol. 24, no. 1, pp. 104–117, Jan. 2013. Article (CrossRef Link)
[21] R. O. Duda, P. E. Hart, and D. G. Stork, Pattern Classification., New York, NY, USA: Wiley, 2012. Article (CrossRef Link)
[21] P.-N. Tan, M. Steinbach, and V. Kumar, Introduction to Data Mining. New York, NY, USA: Pearson Education Inc., 2006. Article (CrossRef Link)
[22] Z. Zheng and D. Su, "Short-term traffic volume forecasting: A k-nearest neighbor approach enhanced by constrained linearly sewing principle component algorithm," *Elsevier Transportation Research Part C: Emerging Technologies*, vol. 43, pp. 143–157, Jun. 2014. Article (CrossRef Link)
[24] Ranger C, Raghuraman R, Penmetsa A, et al., "Evaluating MapReduce for multi-core and multiprocessor systems," in *Proc. of IEEE International Symposium on High PERFORMANCE Computer Architecture*, Scottsdale, AZ, USA, pp. 13–24, Feb. 2007. Article (CrossRef Link)
[25] C.-S. Li and M.-C. Chen, "A data mining based approach for travel time prediction in freeway with non-recurrent congestion," *Neurocomputing*, vol. 133, pp. 74–83, Jun. 2014. Article (CrossRef Link)

Guolin Sun received his B.S., M.S. and Ph.D. degrees all in Comm. and Info. System from the University of Electronic Sci.&Tech. of China (UESTC), Chengdu, China, in 2000, 2003 and 2005 respectively. After Ph.D. graduation in 2005, Dr. Guolin has got eight years industrial work experiences on wireless research and development for LTE, Wi-Fi, Internet of Things (ZIGBEE and RFID, etc.), Cognitive radio, Location and navigation. Before he join the School of Computer Science and Engineering, University of Electronic Sci.&Tech. of China, as an Associate Professor on Aug. 2012, he worked in Huawei Technologies Sweden. Dr. Guolin Sun has filed over 30 patents, and published over 30 scientific conference and journal papers, acts as TPC member of conferences. Currently, he serves as a vice-chair of the 5G oriented cognitive radio SIG of the IEEE (Technical Committee on Cognitive Networks (TCCN) of the IEEE Communication Society. His general research interest is 5G/2020 oriented wireless network, such as software defined networks, network function virtualization, wireless networks.

Bruce Mareri received his BSc in Software Engineering from Kenyatta University, Nairobi Kenya. He got his Certification in Integrated Mobile Telecommunications Technology from the Emobilis Mobile Academy, Nairobi, Kenya, in 2010. He is currently pursuing his MS in the Computer Science in University of Electronic Science and Technology of China (UESTC), Chengdu, China. His interests include Software Defined Networks, Future Networks and Ultra low latency 5G networks.

Guisong Liu received his B.S. degree in Mechanics from the Xi'an Jiao Tong University, Xi'an, China, in 1995, M.S. degree in Automatics and Ph.D. degree in Computer Science both from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2000 and 2007 respectively. Now, he is an associated professor in the School of Computer Science and Engineering, UESTC. His research interests include cloud computing, big data, and computational intelligence.

Xiufen Fang received his B.S. degree and M.S. degree in Department of Applied Mathematics both from the University of Electronic Science and Technology of China, Chengdu, China, in 1995 and 2001, respectively. She was a visiting scholar at Northern Illinois University (NIU) during March to September in 2012. Now, she is an associate professor in the School of Mathematical Sciences, University of Electronic Science and Technology of China. Her research interests include big data, neural networks and scientific computing.

Wei Jiang received his Ph.D degree from Beijing University of Posts and Telecommunications (BUPT) in 2008. Since Mar. 2008, he has been worked 4 years in Central Research Institute of Huawei Technologies, in the field of wireless communications and 3GPP standardization. In Sept. 2012, he joined the Institute of Digital Signal Processing, University of Duisburg-Essen, Germany, where he was a Postdoctoral researcher and worked for EU FP7 ABSOLUTE project and H2020 5G-PPP COHERENT project. Since Oct. 2015, he joined the Intelligent Networking Group, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany, as a senior researcher and works for H2020 5G-PPP SELFNET project. Meanwhile, he also works for the Department of Electrical and Information Technology (EIT), Technische University (TU) Kaiserslautern, Germany, as a senior lecturer. He served as a vice Chair of IEEE TCCN special interest group (SIG) "Cognitive Radio in 5G". He is the author of more than 30 papers in top international journals and conference proceedings, and has 27 patent applications in wireless communications, most of which have already been authorized in China, Europe, United States or Japan. He wrote a chapter "From OFDM to FBMC: Principles and Comparisons" for the book "Signal Processing for 5G: Algorithms and Implementations" (Wiley, 2016). His present research interests are in digital signal processing, multi-antenna technology, cooperative communications, 5G, and machine learning.