Delay Distribution Based Remote Data Fetch Scheme for Hadoop Clusters in Public Cloud

doi: 10.1587/transcom.2018EBP3243

PAPER

Delay Distribution Based Remote Data Fetch Scheme for Hadoop Clusters in Public Cloud*

Ravindra Sandaruwan RANAWEERA†, ***Nonmember***, **Eiji OKI**††, ***Fellow***, ***and*** **Nattapong KITSUWAN**†, ***Member***

SUMMARY Apache Hadoop and its ecosystem have become the de facto platform for processing large-scale data, or Big Data, because it hides the complexity of distributed computing, scheduling, and communication while providing fault-tolerance. Cloud-based environments are becoming a popular platform for hosting Hadoop clusters due to their low initial cost and limitless capacity. However, cloud-based Hadoop clusters bring their own challenges due to contradictory design principles. Hadoop is designed on the shared-nothing principle while cloud is based on the concepts of consolidation and resource sharing. Most of Hadoop's features are designed for on-premises data centers where the cluster topology is known. Hadoop depends on the rack assignment of servers (configured by the cluster administrator) to calculate the distance between servers. Hadoop calculates the distance between servers to find the best remote server from which to fetch data from when fetching non-local data. However, public cloud environment providers do not share rack information of virtual servers with their tenants. Lack of rack information of servers may allow Hadoop to fetch data from a remote server that is on the other side of the data center. To overcome this problem, we propose a delay distribution based scheme to find the closest server to fetch non-local data for public cloud-based Hadoop clusters. The proposed scheme bases server selection on the delay distributions between server pairs. Delay distribution is calculated measuring the round-trip time between servers periodically. Our experiments observe that the proposed scheme outperforms conventional Hadoop nearly by 12% in terms of non-local data fetch time. This reduction in data fetch time will lead to a reduction in job run time, especially in real-world multi-user clusters where non-local data fetching can happen frequently.

key words: *Public cloud, Hadoop, Big Data, HDFS*

1. Introduction

The rapid expansion of IoT (Internet of Things) devices, social networking, and online services etc. in recent years allows organizations to collect large amounts of data. Collecting data and leaving them at rest does not bring any business value to an organization. Thus, the collected data must be processed and analyzed to add business value to the organization by taking data-driven actions or to find value in collected data. Analyzing so-called collected Big Data in a realistic time within an economical cost has been a challenge for organizations.

Apache Hadoop (Hadoop) [2], an open-source implementation of Google's MapReduce [3] framework, is a parallel-distributed processing framework that allows organizations to process very large datasets using commodity hardware efficiently in a realistic time. Hadoop has become the most popular parallel-distributed framework for processing large-scale data because it hides the complexity of distributed computing, scheduling, and communication while providing fault-tolerance. Hadoop makes use of inexpensive, industry standard commodity servers to store and process large volumes of data rather than relying on specially built proprietary servers. Thus, most of the fortune 500 companies use Hadoop to process large-scale data within a reasonable budget. Initially, Hadoop was used by large companies such as Yahoo, Facebook, eBay etc., who were already collecting large amounts of data. These large companies have their own data centers due to cost efficiency and security reasons. Therefore, their Hadoop clusters are also deployed in on-premises data centers using physical clusters.

Cloud-based computing has, however, been drawing attention because of its convenient pay-per-use model and almost limitless capacity. Public cloud vendors provide effective and cheap solutions for storing very large datasets. As a result of the increased popularity of Hadoop and cost-effectiveness of cloud-based computing, medium to small size companies [4] have also started using Hadoop to take advantage of data-driven decision making with the data they already have or they are going to collect. Not only medium to small size companies, but also large companies are finding cloud-based Hadoop clusters more attractive. Large companies such as Netflix and Twitter who had PB scale on-premises Hadoop clusters have started moving to cloud-based Hadoop clusters in recent years [5, 6]. A recent study by Gartner [7] shows that the number of public cloud-based and on-premises Hadoop deployments is almost the same and 29% of Hadoop users intend to use Hadoop both on-premises and in the cloud. Also, IDC estimates that nearly 40% of Big Data analyses will be supported by public cloud [8] infrastructure by 2020. It is clear that the Hadoop deployments

Manuscript received August xx, 2018.

†The authors are with the Graduate School of Informatics and Engineering, The University of Electro-Communications, Tokyo 182-8585 Japan.

††The author is with the Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan.

*This is an extended version of the paper presented to IEEE International Conference on High Performance and Smart Computing (IEEE HPSC 2018) [1].

DOI: 10.1587/transcom.E93.B.1

are moving towards cloud-based infrastructure due to increased advantages of flexibility, scalability, and low initial cost of public cloud platforms. In order to attract these potential customers who are interested in cloud-based Hadoop deployments, not only cloud providers such as Amazon, Microsoft, Google, IBM, etc. but also Hadoop distributors such as Cloudera, Hortonworks, and MapR offer straightforward cloud-based Hadoop deployment solutions [9–15].

Cloud environments are designed as highly-multiplexed, shared environments with virtual servers and tasks from numerous tenants coexisting in the same physical server to achieve cost effectiveness and on-demand scaling. Memory, CPU, and network are the basic resources provided by cloud providers. Tenants can create a Hadoop cluster with a set of virtual servers while achieving performance isolation from each server on memory and CPU resources. On the other hand, the network being a distributed resource makes it harder to control the impact on the performance of each server, unlike CPU and memory. As a result, the potential for network performance interference is high and network performance predictability remains a key concern [16] in cloud environments.

HDFS (Hadoop Distributed File System) is the distributed storage layer that is responsible for storing data in Hadoop. In HDFS data is broken into blocks and stored in multiple (at least three), different servers as replicas for fault-tolerance and availability purposes. Data processing frameworks on top of Hadoop such as MapReduce, Hive or Spark access data stored in HDFS. HDFS is designed with write-once-read-many access model [17, 18] to attain high throughput of data access. Write-once-read-many means that once data is written to HDFS, that particular data will be read by processing tasks many times over the time. Therefore, improving how data is read in HDFS has a larger impact on the overall performance of HDFS compared to improving how data is written to HDFS.

Data locality is the property that defines whether data and processing task are co-located on the same server. Hadoop tries to co-locate data and processing task so that data access is fast because data is local [18]. This is one of the revolutionary concepts that was introduced in Hadoop: "taking calculation to where data is" rather than "taking data to the calculation". Specially for large data sets, moving data over the network is inefficient and costly. Unfortunately, it is not always possible to co-locate data and processing task due to resource unavailability. Experiments done by Ibrahim et al. [19] show that approximately 23% of the map tasks are non-local map tasks. Put differently, 23% of the map tasks fetch non-local data required for the task from a remote server that holds a copy of the data before the data processing starts. Many improvements for enhancing data placement (data write) [20–22] in HDFS were presented. However all of these studies are focused on optimizing replica placement, which means writing data into HDFS, while paying no attention to data read improvement, that occurs more often compared to data write.

When fetching a non-local data block, Hadoop finds the best server to fetch data from comparing the network distance of servers that hold a copy of the particular data. The concept called "rack awareness" is used to calculate the network distance. The "rack awareness" concept and how Hadoop calculates network distance is explained in Section 2.1. Cluster administrators must assign a rack for each server in the cluster based on cluster topology information manually or using a script [23] to enable rack awareness. Cluster administrators have access to the cluster topology information in on-premises Hadoop clusters. This allows cluster administrators to specify rack of each server, enabling "rack awareness". Even though private cloud or virtualized server based Hadoop clusters are different from on-premises Hadoop clusters, they can also utilize rack awareness since the administrators have cluster topology information. Sahara [24] for private cloud-based Hadoop clusters and Hadoop Virtualization Extensions (HVE) [25] for virtualized server based Hadoop clusters were presented to enhance Hadoop clusters running in these environments.

However, unlike on-premises, private cloud or virtualized environments, it is impossible to know the cluster topology in public cloud environments since public cloud vendors do not share information about physical rack layouts. Therefore, assigning racks in public cloud environments becomes impossible. If the rack assignment of servers are not configured, as in public cloud-based Hadoop clusters, HDFS data read performance will be effected taking a longer time to fetch non-local data [18]. In addition, public cloud environments are shared by many tenants and available resources at a given time vary depending on the usage. This may lead to performance degradation and load imbalance in the Hadoop cluster. Therefore, it is necessary to have a dynamic network distance calculation scheme for Hadoop clusters hosted on public cloud environments.

It is clear that Hadoop's current data read mechanism, which heavily depends on static rack awareness configuration, is not able to find the best server to fetch data from in public cloud-based Hadoop clusters. Motivated by this, this paper proposes a scheme to find the best server to fetch data from, by dynamically considering the network delay distribution for public cloud-based Hadoop clusters. More specifically, the proposed scheme uses network delay distribution between server pairs in the cluster to calculate the logical distance. This logical distance is used to select the best server to fetch non-local data. Experiments in public cloud environment indicate that the proposed scheme reduces non-local data fetch time compared to conventional Hadoop resulting shorter job run times and saving valu-

able cluster resources. This paper is an extended version of [1], where we extensively investigate the effect of the number of background jobs and size of fetched data on data fetch time using real-world Hadoop clusters.

The remainder of this paper is organized as follows. Section 2 introduces some basic background of Hadoop framework in detail. Section 3.1 describes the architecture overview of the proposed scheme and section 3.2 describes our proposed scheme. In section 4, we describe the experimental results and section 5 concludes this paper and provides future works.

2. Background

This section gives a brief overview of Hadoop which is the open source implementation of MapReduce [3].

HDFS and YARN (Yet Another Resource Negotiator) are the two main components of Hadoop. HDFS is the distributed storage system. YARN is responsible for cluster resource management and job scheduling. Typically, HDFS and YARN both are co-located in the same server. HDFS and YARN both are designed with master-slave architecture. The master process of HDFS is called namenonde and it oversees and manages data storage. The master process of YARN is called resourcemanager and it oversees cluster resource management and computing functionalities. The slave process of HDFS called datanode, stores the actual data in its local disks and answers to data read/write requests while the slave process of YARN called nodemanager, does the processing within containers.

2.1 Rack Awareness

Figure 1 shows a typical on-premises cluster, which is configured in fat-tree topology [26] with rows of racks. Each rack contains 20-40 servers and they are connected to a top-of-rack switch. These top-of-rack switches connect to one or more core switches, creating multiple paths between two servers in the cluster. The links which connect core layer and edge layer are shared by multiple servers at the same time. Traffic that go from one rack to another, also called cross-rack traffic, travels through the core layer links, making them a bottleneck.

Hadoop is designed to reduce cross-rack traffic and utilize in-rack resources as much as possible because cross-rack paths get congested easily. Unfortunately, Hadoop is not able to understand the network topology by itself without any human help. Rack assignment information of of each server, especially slave servers, must be configured by cluster administrator so that Hadoop is able to utilize in-rack resources as much as possible. The network distance or the closeness between slave servers are calculated using this rack information. Two servers in the same rack are closer compared to two servers in separate racks. The network distance calculated using rack information will be

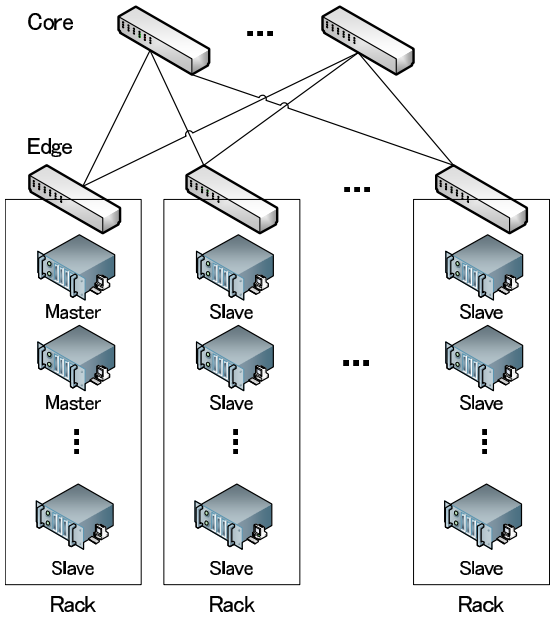

Fig. 1 Fat tree topology based on-premises Hadoop cluster

static as long as the rack layout and rack information are not changed.

Hadoop heavily depends on this static rack information when reading data stored in HDFS. The data reading procedure of HDFS is explained below.

1. A client that wants to read data in HDFS contacts the namenode to determine the locations of the data.
2. The namenode finds the addresses of the datanodes that have a replica of the requested data and sorts the addresses according to the proximity to the client using rack information. The sorted list of datanode addressed are sent back to the client which requested data locations.
3. If the client itself is a datanode and holds a replica of the data block, it reads data directly from the local disk (data locality). Otherwise, the client connects to the closest datanode according to the sorted datanodes list it received from namenode and fetches non-local data via the network.

Hadoop considers the cluster topology as flat in cloud-based Hadoop clusters where rack information is not available. This leave Hadoop to misunderstand that all servers are in a single rack even though it is different in reality. As a result, Hadoop fails to calculate the network distance between datanodes when reading data and selects a datanode randomly to fetch non-local data. Selecting a random datanode causes longer data transfer time resulting longer job run time and wasting valuable cluster resources.

3. Proposed scheme

3.1 Architecture overview

The proposed non-local data fetch scheme based on delay distribution for Hadoop clusters in public cloud environments is presented in this section. The basic idea of the proposed scheme is to compare the distribution of round-trip time (RTT) between datanodes and select a datanode with least delay when it is necessary to fetch non-local data. The proposed scheme adds two changes to conventional Hadoop: (1) adds a new feature to datanode daemon to measure round-trip time between servers, and (2) extends HDFS data read procedure to use RTT-based delay distribution. The flowchart of round-trip time measurement and delay distribution calculation is shown in Figure 3. The calculated delay distributions between datanode pairs are compared when a datanode is selected to fetch data. The namenode is responsible for sorting the datanodes according to the proximity to the client in conventional Hadoop. However, delay distribution is measured by each datanode and it is not available to the namenode. Therefore, in the presented scheme, a datanode which acts as an HDFS client when fetching non-local data, sorts the datanodes that hold replicas of a data block according to the delay distribution, and connects to the first datanode to fetch data.

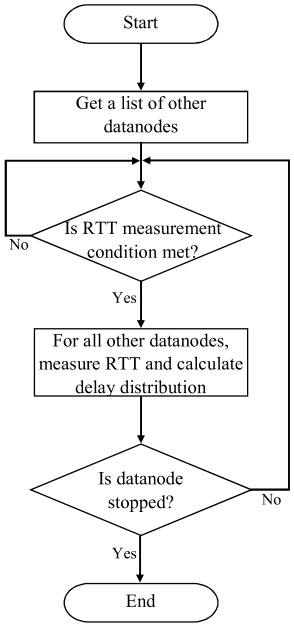

Fig. 2 Flowchart of measuring RTT between servers

3.2 Non-local data fetch scheme

Conventional Hadoop calculates a network distance between servers by using the physical rack layout information. Unfortunately, public cloud vendors do not provide physical rack layout information to their tenants as we explained in section 1. Even if the "static" physical distance is obtained, it does not represent the resource availability correctly. It is necessary to dynamically calculate the network distance according to the state of the available resources because public cloud environments are shared by many unrelated tenants and performance interferences can happen frequently. That is why we use a logical distance that can be calculated by using commonly available/obtainable network information in public cloud environments. Instead of using a static metric such as hop count, the proposed scheme uses RTT-based delay distribution to calculate the logical distance between servers. We adopt RTT since it is relatively easy and inexpensive to use in any environment.

The procedure of measuring RTT between servers is explained below.

1. Each datanode gets a list of datanodes connected to the cluster at startup.
2. Each datanode sends an Internet Control Message Protocol (ICMP) echo request to all the other datanodes periodically.
3. Each datanode records the time it took to get an ICMP echo reply (RTT) from other datanodes since sending the echo request.

RTT can suddenly change depending on the network traffic and server workload, which will result in abnormal measurements of RTT. In order to minimize the impact of the sudden changes of RTT, the distribution of RTT, or delay distribution that shows the probability characteristic of the delay is used. Delay distribution between datanodes can be calculated by using the periodically measured RTT. We update the delay distribution by using exponential smoothing technique [27, 28, 28] as shown Eq. (1). Exponential smoothing technique is often used for time-series data and it can be easily applied for making some determination based on prior observations. Let t be the measured RTT between two servers and $p_\tau(t)$ be the measured RTT distribution at time τ. $f_\tau(t)$ is the smoothed RTT distribution at time τ, $f_{\tau-1}(t)$ is the smoothed RTT distribution calculated at time $\tau-1$, and α is the smoothing factor.

$$f_0(t) = p_0(t), \tau = 0 \tag{1a}$$

$$f_\tau(t) = \alpha p_\tau(t) + (1-\alpha) f_{\tau-1}(t), \tau > 0 \tag{1b}$$

$$0 \le \alpha \le 1 \tag{1c}$$

In other words, the smoothed RTT distribution $f_\tau(t)$

is a simple weighted average of the current measurement of RTT distribution and the previous smoothed RTT distribution. The value selected for α determines how $f_\tau(t)$ is updated. A larger value of α has less of a smoothing effect and gives a greater weight to recent changes in the measured data. In the extreme case with $\alpha = 1$, the output series are the same as the RTT distribution. A smaller value of α has more of a smoothing effect and gives a greater weight to measurements from the more distant past.

The average time, maximum time, minimum time, or a randomly selected point of the delay distribution can be used to compare the delay distributions between server pairs. In this study, we used the average, maximum, minimum, ϵ, and $1-\epsilon$ as comparison policies. The delay distribution between servers can be calculated by using the measured RTT between servers. t_{min} is the minimum delay time, t_{max} is the maximum delay time, and t_{avg} is the average delay time. ϵ is the percentile of delay distribution and it is defined as,

$$\epsilon = \int_{t_\epsilon}^{t_{max}} f_\tau(t)dt. \quad (2)$$

By using Eq. (2) for a given ϵ, we can calculate t_ϵ. Similarly, $t_{1-\epsilon}$ is defined as,

$$\epsilon = \int_{t_{min}}^{t_{1-\epsilon}} f_\tau(t)dt. \quad (3)$$

Figure 3 shows the positions of t_{min}, $t_{1-\epsilon}$, t_{avg}, t_ϵ, and t_{max} in the probability density function. The minimum delay time, t_{min}, and the maximum delay time, t_{max}, are the two extremes of the delay distribution and they do not accurately represent the delay characteristic of server pairs. Values of t_{max} and t_{min} are relatively unstable and contain abnormal delay times due to sudden changes of network conditions in the environment. Therefore, this study only uses t_{min} and t_{max} for comparison purposes.

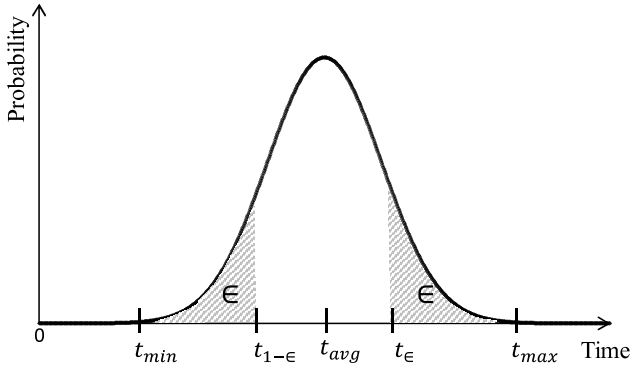

Fig. 3 Probability function's distribution

The modified data reading procedure of HDFS utilizing the delay distribution is explained below.

1. A client contacts the namenode to determine the locations of the data.
2. The namenode sends a list of the datanodes that have a replica of requested data to the client.
3. If the client itself is a datanode and holds a replica of the data, it directly reads data from the local disk. Otherwise, the client sorts datanodes list according to the delay distribution comparison policy and connects to the datanode with the least delay to fetch non-local data.

Let N be a set of datanodes in the cluster, and $M \subset N$ be a set of datanodes that holds a particular data block where $|M| = 3$. Let P be the set of policies where $P \in \{min, max, avg, \epsilon, 1-\epsilon\}$ and the selected remote datanode can be expressed as,

$$datanode(i) = \arg \min_{i' \in M} \min_{policy \in P} t_{policy}(i'). \quad (4)$$

4. Performance Evaluation

4.1 Experimental Environment

This study compares HDFS data read performance of the proposed scheme with conventional Hadoop by fetching data from HDFS and running MapReduce jobs. The total time took to fetch data from HDFS and MapReduce job run time are the performance measures of the evaluation.

Six clusters, one cluster for each policy, are used for this experiment and their configurations are described as follows. All the clusters are based on CDH (Cloudera's Distribution including Apache Hadoop) 5.11.1 [30] and deployed on AWS EC2 Tokyo region. Each cluster consists of seven compute optimized c4.4xlarge [31] instances (virtual servers) with one master server and six slave servers. Each instance has 16 vCPUs, 32 GiB (GiB: gibibyte is a multiple of the unit byte digital data storage where $1GiB \approx 1.074GB$) of memory with EBS storage [31]. CentOS 7.3 is installed as the operating system. CDH 5.11.1 includes Hadoop-2.6 with backports of latest patches. Default scheduler of CDH, *Fair Scheduler*, is used without configuring any additional queues. 2GB of memory and one virtual CPU for each map and reduce task is configured. The maximum memory allocated for YARN containers in each nodemanager is 30GB leaving 2GB of memory operating system. Similarly, 15 virtual CPUs are allocated for YARN containers in each nodemanger leaving one virtual CPU for the operating system tasks. A total of 210GB of memory and 105 virtual CPUs are available in the cluster. Rack assignment is not configured for any of the servers since we do not have any physical rack layout information. Hence, Hadoop assumes that all the servers are in one rack named *default*. For performance evaluation, we create random text data using Hadoop's *randomtextwriter* program. All the jobs

are executed at the same time from six clusters. Clusters are deployed once well in advance before the jobs run and each job is executed ten times. The averaged results and the standard deviations are shown in section 4.2.

In this experiment, the value of α is set to 0.5 in order to consider both new and old probabilities of RTT. RTT between datanode pairs is measured every second and delay distribution is calculated based on the measured RTT. ϵ is set to 0.15 after comparing data fetch times for different values of ϵ. In order to implement the proposed scheme, a new Java class is added to the datanode to measure the delay time between datanodes and *DistributedFileSystem* class is customized to compare the delay distributions.

4.2 Results and discussions

First, HDFS data fetch time of conventional Hadoop and the proposed scheme are measured. The results are shown in Table 1. A 1GB file (eight data blocks) is fetched from HDFS to one of the datanodes using built in *hdfs dfs* command. For this experiment, we select the datanode that has the least number of replicas of the 1GB file so that Hadoop can fetch more replicas from remote datanodes. In this experiment, there are only one replica stored at the particular datanode and seven replicas are fetched from remote datanodes. Data fetch time without background jobs and with background jobs are measured.

To simulate a real-world multi-user cluster, data fetch time while running background jobs are measured. Three *wordcount* MapReduce jobs, each counting words in separate 20GB file are used as background jobs. *Wordcount* is selected because it achieves a balance in both map and reduce stages. Each map task of the *wordcount* job reads the input file line by line and breaks it into words with key/value pair of the word and 1. Each reduce task sums the counts of each word and creates a single key/value with the word and sum as the result of the job.

Conventional Hadoop is expressed as T_{con}. T_P, $P \in \{avg, max, min, \epsilon, 1-\epsilon\}$ shows which policy is used as the delay distribution comparison policy. T_{avg} expresses that t_{avg} is used as the delay distribution comparison policy, T_{max} expresses that t_{max} is used as the delay distribution comparison policy, etc.

When there are no background jobs running,

$$T_\epsilon < T_{avg} < T_{1-\epsilon} < T_{min} < T_{max} < T_{con} \quad (5)$$

is observed. For the scenario with background jobs running,

$$T_{avg} < T_\epsilon < T_{1-\epsilon} < T_{max} < T_{min} < T_{con} \quad (6)$$

is observed. In the case that there are background jobs running, all the policies including conventional Hadoop take longer time to fetch data compared to the results of no background jobs. With background jobs, there are more traffic (non-local data fetch of map tasks, shuffle data etc.) transferred between the datanodes. This reduces the overall network throughput leading to longer data transfer times.

Table 1 HDFS data fetch times

Policy	Without background jobs		With background jobs	
	Time [sec]	Stdev. [sec]	Time [sec]	Stdev. [sec]
T_{con}	4.3988	0.141	5.7020	0.184
T_{avg}	4.2700	0.135	5.0283	0.192
T_{max}	4.3587	0.161	5.5263	0.188
T_{min}	4.3454	0.162	5.5682	0.181
T_ϵ	4.2366	0.155	5.1521	0.177
$T_{1-\epsilon}$	4.3131	0.144	5.3139	0.198

When fetching non-local data from remote datanodes using conventional Hadoop, it randomly selects a remote datanode to fetch data from. This randomly selected datanode might be close to the data fetching datanode or distant from the data fetching datanode. Both Eqs. (5) and (6) show that these randomly selected remote datanodes are not closer to the data fetching server, resulting longer data fetch times when conventional Hadoop is used. On the other hand, T_P, $P \in \{avg, max, min, \epsilon, 1-\epsilon\}$, respectively uses t_{avg}, t_{max}, t_{min}, t_ϵ, and $t_{1-\epsilon}$ of delay distribution as the logical distance between servers to compare and select the closest datanode resulting shorter data fetch times. However, there is a possibility that T_{max} or T_{min}, which compares the extremes of the delay distribution, may be inferior compared to T_{con} if there are abnormal delays in RTT. In this experiment, there were no abnormal delays that would affect the performance.

In the case without background jobs, limited number of data fetching from the same datanote occur. In this case, reducing the worst-case delay time, t_{max}, completes the data fetch in the least amount of time. However, t_{max} and t_{min} are the two extremes of delay distribution and they do not accurately represent the delay characteristic of the cluster. Values of t_{max} and t_{min} are relatively unstable and contain abnormal delay times due to sudden network condition changes which are more likely to occur in public cloud environments. Therefore, a policy that is robust against sudden network condition changes but reduces the worst-case delay is desirable. Policy ϵ excludes worst-case delay and compares the delay times that are ϵ% from the worst-case delay, which is robust against sudden network changes. The experimental results also show that T_ϵ is able to fetch data in a shorter time compared to conventional Hadoop and other policies when there are no background jobs.

In the case with background jobs, background jobs also fetch data in addition to the data fetch command that we run. This leads to multiple data fetches from

the same remote datanode. Reducing just one data fetch does not shorten the total data fetch time when there are multiple data fetches because it is difficult to estimate which data fetch among multiple data fetches should be reduced. A policy that reduces the total time of multiple data fetches at the same time is more suitable. The policy that compares the average delay time is most suitable since it considers delay times of multiple data fetches. The experimental results also show that T_{avg} is able to fetch data in a shorter time compared to conventional Hadoop and other proposed policies with background jobs.

Table 2 shows the results of *wordcount* MapReduce job, processing 10GB file without background jobs. Rack-local map tasks are the map tasks that fetch data from other datanodes. In this experiment, only data-local and rack-local map tasks exist, since rack locations are not configured leaving Hadoop to consider that all the servers are in a single rack. The number of rack-local map tasks are verified from the job counters information.

Table 2 Wordcount job completion time without background jobs

Policy	Time [sec]	Stdev. [sec]	Average ratio of rack-local map tasks [%]
T_{con}	85.294	3.92	19.89
T_{avg}	80.551	3.42	19.89
T_{max}	83.966	3.45	20.00
T_{min}	84.880	4.11	20.11
T_{ϵ}	80.053	3.84	20.00
$T_{1-\epsilon}$	82.851	3.71	20.11

From Table 2,

$$T_{\epsilon} < T_{avg} < T_{1-\epsilon} < T_{max} < T_{min} < T_{con} \qquad (7)$$

is observed. Equation (7) shows that the proposed scheme is able to reduce the *wordcount* job run time compared to conventional Hadoop even though the average ratio of rack-local map tasks is slightly higher. In the case that there are no background jobs running, *wordcount* job is finished in shortest time when ϵ policy is used. This is similar to the observation of Eq. (5). Therefore, we can say that the policy which reduces the worst-case delay time is preferable when there are only a few data fetches occurring in the cluster.

Table 3 shows the results of *wordcount* job, processing 10GB file with background jobs. Three *wordcount* jobs are executed in as background jobs, processing separate 20GB files. Table 3 also shows the average rack-local map task ratio.

From Table 3,

$$T_{avg} < T_{\epsilon} < T_{1-\epsilon} < T_{min} < T_{max} < T_{con} \qquad (8)$$

is observed. This shows that the proposed scheme

Table 3 Wordcount job completion time with background jobs

Policy	Time[sec]	Stdev[sec]	Average ratio of rack-local map tasks [%]
T_{con}	167.931	10.2	30.40
T_{avg}	152.293	11.42	30.00
T_{max}	159.902	10.9	31.32
T_{min}	159.591	11.29	30.04
T_{ϵ}	153.417	10.85	30.00
$T_{1-\epsilon}$	156.081	11.13	30.32

is able to reduce the job run time even with background jobs. The average job run time compared to Table 2 is higher for all the policies including conventional Hadoop. This is related to the fact that there are more tasks running in the cluster which adds extra traffic and more wait time to schedule tasks. Equation (8) shows that T_{avg} finishes in the shortest time. This is similar to what we observed in Eq. (6). Therefore, the policy that reduces the average delay time is most suitable for real-world workloads where there are multiple non-local data fetches occurring at the same time.

In real-world clusters there are multiple jobs running at the same time and from the above experimental results it is confirmed that reducing the average of the delay distribution is most effective in such clusters. In order to further investigate the effectiveness of the policy that compares average delay distributions, data fetch times are measured while changing the number of background jobs. Figure 4 shows the data fetch times of conventional Hadoop and the most effective policy in the proposed scheme, which reduces the average delay time. A 1GB file is fetched from HDFS to measure the data fetch times. *Wordcount* MapReduce jobs that process a 20GB file each are used as background jobs.

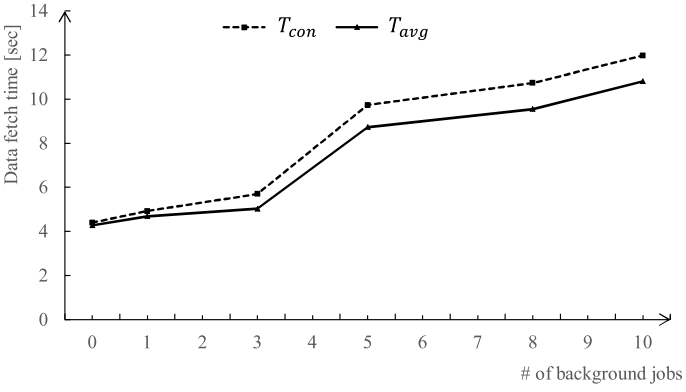

Fig. 4 HDFS data fetch time with changing number of background jobs

$$T_{avg} < T_{con} \qquad (9)$$

is observed from Figure 4. These results further prove that the proposed scheme is effective and continues to

outperform conventional Hadoop with the number of background jobs. However, the gap between the conventional Hadoop and proposed scheme stops increasing after three background jobs. This is mainly related to the network throughput reduction due to the increase in number of background jobs. More background jobs cause more traffic in the network, reducing the network throughput. As a result, the time it takes to transfer data over the network increases similarly for both conventional Hadoop and the proposed scheme. However, the block locations and the number of blocks of the 1GB file are fixed and the time to fetch non-local data of the 1GB file becomes identical. Therefore, the effectiveness of the proposed scheme peaks irrespective of the number of background jobs.

Furthermore, data fetch time is measured while changing the size of the fetched data. Three *wordcount* jobs are used as background jobs. Figure 5 summarizes the data size, data fetch time, and data fetch time reduction rate of the proposed scheme compared to conventional Hadoop. Results from Figure 5 shows that the proposed scheme becomes more effective with the fetched data size. The number of data blocks that needs to be fetched from remote datanodes increases with the data size. More data blocks provide more opportunities for the proposed scheme to expand the difference with conventional Hadoop.

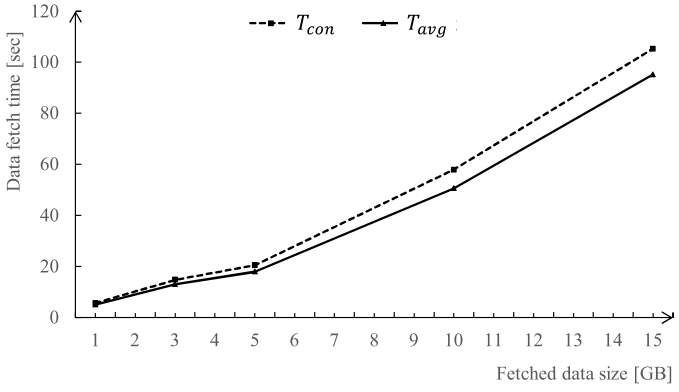

Fig. 5 Comparison of data fetch time with fetched data size

The experimental results show that the proposed scheme, which uses delay distribution to compare the logical distance between datanodes, is able to fetch non-local data from remote datanodes efficiently compared to conventional Hadoop. This results in shorter job run times, saving valuable cluster resources which can be used for other data processing jobs. Equations (6) and (8) resemble the results that are closer to real-world multi-user clusters where there are jobs running at the same time. The number of non-local data fetches increases with the number of concurrent jobs since it is harder to schedule processing tasks on the same server where data is. Therefore, T_{avg}, which reduces the average delay time of concurrent data fetches, is most suitable for real-world clusters.

AWS EC2 is used as the experiment environment in this study. The proposed scheme can easily be used in any other public/private cloud environment since it does not use any AWS specific tools. We selected relatively easy to implement and inexpensive RTT based delay distribution to compare server pairs, thus paving the way to use our proposed scheme in any other cloud environment. The experiments performed by changing the number of background jobs and fetched data size confirmed that the proposed scheme is effective even if the workload changes. The measurement values shown in this section may change depending on the available resources in the environment at the job execution time as we explained in section 1. However, the proposed scheme will outperform conventional Hadoop because it selects datanode by comparing the delay distributions rather than randomly selecting a datanode to fetch data.

The proposed scheme compares RTT based delay distributions to select a datanode to fetch data. Hadoop exposes statistical information regarding Hadoop daemons, called metrics [32] for monitoring, performance tuning, and debug purposes. Some of the metrics exposed by Hadoop daemons can also be used when a datanode is selected to fetch remote data with some customization, since non of them measures time between server pairs in the cluster.

5. Conclusion

Cluster administrator configured rack assignment information of servers in the cluster is used by conventional Hadoop to calculate the network distance between datanodes. Hadoop uses the network distance (hop count) calculated using rack assignment information to select the best server to fetch non-local data. However, there are cases where it is impossible to know the rack assignment information such as public clouds. Public cloud providers do not share the physical rack layout information of servers with tenants. Therefore, the administrators of public cloud-based Hadoop clusters are unable to configure rack assignment of servers. When rack information is not configured, Hadoop considers all servers to be in one rack. This can lead to fetching non-local data from randomly selected datanodes, causing longer data fetch time and wasting valuable cluster resources.

This paper proposed a delay distribution based scheme to find the best datanode to fetch data from. It determines the best remote datanode by comparing delay distributions of datanode pairs. The average, maximum, minimum, ϵ, and $1-\epsilon$ of delay time is used to compare the delay distributions between datanode pairs. Experiments based on a public cloud environ-

ment showed that the proposed scheme reduces non-local data fetch time compared to conventional Hadoop resulting shorter job run times. The results also suggested the policy that reduces the average delay time is better for real-world multi-user Hadoop clusters, since it reduces the total delay time of multiple data fetches. Moreover, the results suggested that this policy increases the data fetch time difference from conventional Hadoop as the data size increases.

In this paper, we used a probabilistic method to improve non-local data fetching of public cloud-based Hadoop clusters. Confirming the effectiveness of the proposed scheme on on-premises Hadoop clusters is left as part of our future work.

References

[1] R. S. Ranaweera, E. Oji, and N. Kitsuwan, "Non-local Data Fetch Scheme Based on Delay Distribution for Hadoop Clusters in Public Cloud," The 4th IEEE International Conference on High Performance and Smart Computing (IEEE HPSC 2018), May 2018.

[2] "Apache Hadoop," http://hadoop.apache.org/, accessed July 2017.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in USENIX Symposium on Operating Systems Design and Implementation, San Francisco, CA, Dec. 2004, pp. 137150.

[4] "Powered by Apache Hadoop," https://wiki.apache.org/hadoop/PoweredBy, accessed July 2017.

[5] "Completing the Netflix Cloud Migration," https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration, accessed July 2018.

[6] "A new collaboration with Google Cloud," https://blog.twitter.com/engineering/en_us/topics/infrastructure/2018/a-new-collaboration-with-google-cloud.html, accessed July 2018.

[7] "Hadoop Expansion Boosts Cloud and Unsupported On-Premises Deployments," https://www.gartner.com/doc/3124018/hadoop-expansion-boosts-cloud-unsupported, accessed July 2017.

[8] "The digital universe in 2020," http://idcdocserv.com/1414, accessed July 2017.

[9] "Amazon EMR," https://aws.amazon.com/emr/, accessed July 2017.

[10] "HDInsight," https://azure.microsoft.com/en-us/services/hdinsight/, accessed July 2017.

[11] "CLOUD DATAPROC," https://cloud.google.com/dataproc, accessed March 2018.

[12] "IBM BigInsights on Cloud," https://www.ibm.com/analytics/hadoop, accessed July 2017.

[13] "Hortonworks Cloud Solutions," https://hortonworks.com/products/data-platforms/cloud/, accessed January 2018.

[14] https://www.cloudera.com/products/cloud.html, accessed January 2018.

[15] "MapR Orbit Cloud Suite," https://mapr.com/products/orbit-cloud/, accessed January 2018.

[16] A. Sheih, S. Kandula, A. Greenberg, C. Kim, B. Saha, "Sharing the Data Center Network," in USENIX conference on Networked systems design and implementation, Boston, USA, 2011, pp. 309-322.

[17] "HDFS Architecture," https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, accessed July 2017.

[18] Hadoop The Devinitive Guide 4th Edition, Tom White, O'reilly, April 2015.

[19] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, S. Wu, "Maestro: Replica-Aware Map Scheduling for MapReduce," in 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Canada, 2012, pp. 435-442.

[20] H. Jin, X. Yang, X. H. Sun, and I. Raicu, "ADAPT:Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing," in IEEE International Conference on Distributed Computing Systems, Macau, China, 2012, pp. 516525.

[21] W. Dai, I. Ibrahim, M. Bassiouni, "A New Replica Placement Policy for Hadoop Distributed File System," 2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), New York, USA, 2016, pp. 262-267.

[22] M. Y. Eltabakh, Y. Tian, F. Ozcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop," Proc. VLDB Endow., vol. 4, no. 9, pp. 575585, Jun. 2011.

[23] https://wiki.apache.org/hadoop/topology_rack_awareness_scripts, accessed July 2017.

[24] https://docs.openstack.org/sahara/latest/, accessed January 2018.

[25] https://issues.apache.org/jira/browse/HADOOP-8468, accessed January 2018.

[26] C. E. Leiserson, "Fat-trees: Universal Networks for Hardware-efficient Supercomputing," IEEE Trans. Comput., vol. 34, no. 10, pp.892-901, Oct. 1985.

[27] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," (O.N.R. Memorandum No. 52). Carnegie Institute of Technology, Pittsburgh USA, 1957.

[28] R. G. Brown, "Statistical forecasting for inventory control," McGraw/Hill, 1959.

[29] "Forecasting Principles and Practice," https://otexts.org/fpp2/expsmooth.html, accessed November 2018.

[30] "CDH Packaging and Tarball information," https://www.cloudera.com/documentation/enterprise/release-notes/topics/cdh_vd_cdh_package_tarball_511.html#concept_vvq_42n_yk, accessed September 2017.

[31] "Amazon EC2 Instance Types," https://aws.amazon.com/ec2/instance-types/, accessed January 2018.

[32] "Metrics Guide," https://hadoop.apache.org/docs/r2.6.5/hadoop-project-dist/hadoop-common/Metrics.html.

Ravindra Sandaruwan Ranaweera received his B.E. and M.E. degrees in Communication Engineering and Informatics from The University of Electro-Communications, Tokyo, Japan in 2012 and 2014, respectively. He is currently with the Graduate School of Informatics and Engineering, The University of Electro-Communications, Tokyo, Japan. His research interests include network optimization, distributed deep learning, and distributed computing. He is a student member of IEEE.

Eiji Oki received B.E. and M.E. degrees in instrumentation engineering and a Ph.D. degree in electrical engineering from Keio University, Yokohama,

Japan, in 1991, 1993, and 1999, respectively. He was with Nippon Telegraph and Telephone Corporation (NTT) Laboratories, Tokyo, from 1993 to 2008, and The University of University of Electro-Communications, Tokyo, from 2008 to 2017. From 2000 to 2001, he was a Visiting Scholar at the Polytechnic Institute of New York University, Brooklyn. He is a Professor at Kyoto University, Kyoto, Japan, from 2017. His research interests include routing, switching, protocols, optimization, and traffic engineering in communication and information networks. He is a Follow of IEEE.

Nattapong Kitsuwan Nattapong Kitsuwan received B.E. and M.E. degrees in electrical engineering (telecommunication) from Mahanakorn University of Technology, King Mongkut's Institute of Technology, Ladkrabang, Thailand, and a Ph.D. in information and communication engineering from the University of Electro-Communications, Japan, in 2000, 2004, and 2011, respectively. From 2002 to 2003, he was an exchange student at the University of Electro-Communications, Tokyo, Japan, where he performed research regarding optical packet switching. From 2003 to 2005, he was working for ROHM Integrated Semiconductor, Thailand, as an Information System Expert. He was a postdoctoral researcher at the University of Electro-Communications from 2011 to 2013. He worked as a researcher for the Telecommunications Research Centre (CTVR), Trinity College Dublin, Ireland from 2013 to 2015. Currently, he is an assistant professor at the University of Electro-Communications, Tokyo, Japan. His research focuses on optical network technologies, routing protocols, and software-defined networks.