INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt (2016)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nem.1928

ROUTE: run-time robust reducer workload estimation for MapReduce

Zhihong Liu,[1,3] Qi Zhang,[2,3] Raouf Boutaba,[3,*,†] Yaping Liu[1] and Zhenghu Gong[1]

[1]*College of Computer, National University of Defense Technology, Changsha, Hunan, China*
[2]*Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada*
[3]*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada*

SUMMARY

MapReduce has become a popular model for large-scale data processing in recent years. Many works on MapReduce scheduling (e.g., load balancing and deadline-aware scheduling) have emphasized the importance of predicting workload received by individual reducers. However, because the input characteristics and user-specified map function of a given job are unknown to the MapReduce framework before the job starts, accurately predicting workload of reducers can be a difficult challenge. To address this challenge, we present ROUTE, a run-time robust reducer workload estimation technique for MapReduce. ROUTE progressively samples the partition size of the early completed mappers, allowing ROUTE to perform estimation at run time yet fulfilling the accuracy requirement specified by users. Moreover, by using robust estimation and bootstrapping resampling techniques, ROUTE can achieve high applicability to a wide variety of applications. Through experiments using both real and synthetic data on an 11-node Hadoop cluster, we show ROUTE can achieve high accuracy with error rate no more than 10.92% and an improvement of 40.6% in terms of error rate while compared with the state-of-the-art solution. Besides, through simulations using synthetic data, we show that ROUTE is robust to a variety of skewed distributions. Finally, we apply ROUTE to existing load balancing and deadline-aware scheduling frameworks and show ROUTE significantly improves the performance of these frameworks. Copyright © 2016 John Wiley & Sons, Ltd

Received 25 September 2015; Revised 13 January 2016; Accepted 2 March 2016

1. INTRODUCTION

Today, the explosion of data has generated tremendous demand for large-scale data processing. MapReduce [1], as a big data analytic framework in cloud computing environment, has gained much popularity. In MapReduce, a job consists of two stages of processing, *map* and *reduce*. In the map stage, the processing of a job is divided to a number of smaller sub-problems, each of which is processed by a map task in a distributed manner. Subsequently in the reduce stage, the output of all the sub-problems is aggregated by a number of reduce tasks, thereby generating the final output for the original problem. Because of its advantages in simplicity and scalability, many of the major IT companies such as Facebook and Twitter have been using MapReduce to process large volumes of data on a daily basis.

While the benefit of MapReduce is apparent, managing the performance of MapReduce jobs is often challenging. Substantial efforts have been made towards improving the performance of MapReduce such as load balancing, reducer locality-aware scheduling, and deadline-aware scheduling. For these techniques, accurately and efficiently predicting the workload received by reducers is desirable because: (i) In load balancing, we can easily identify the reducers with heavy workload and help them

*Correspondence to: Raouf Boutaba, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada.
†E-mail: rboutaba@uwaterloo.ca

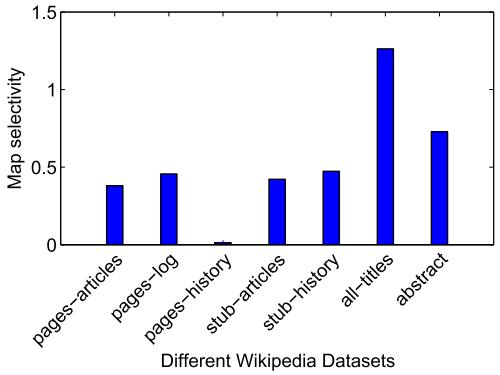


Figure 1. The Selectivity$_M$ for WordCount using different kinds of Wikipedia datasets[1]

finish faster through techniques such as early start [2], workload rebalancing [3–7], and resource allocation adjustment [8,9]; (ii) In reducer locality-aware scheduling, we can execute reducers with heavy workload on machines that can achieve better locality [10,11]; and (iii) in deadline-aware scheduling, we can accurately predict the job running time even when data skew is present, thereby allocating sufficient resources to meet the deadline [12,13].

However, predicting the workload of each reducer is non-trivial. That is because the workload of each reducer is determined by the characteristics of the input datasets and the user-specified map function, both of which can vary from job to job. Some research works [12,13] estimate workload of reducers based on the assumptions that (i) map selectivity Selectivity$_M$, which is defined as the ratio between map output size and map input size, keeps invariant and (ii) the intermediate data are evenly distributed to reducers. Under these assumptions, the workload of a reducer can be computed as ReducuerLoad $=$ Size$_{\text{dataset}} *$ Selectivity$_M$/NumberReducer. However, we would like to point out that even for the same MapReduce job, Selectivity$_M$ may vary if different input datasets are processed. Figure 1 shows the results when running WordCount with combiner using different kinds of Wikipedia datasets. It is clear that there is a drastic difference between two datasets. In addition to this, we would also like to indicate that load distribution in the reduce stage is not necessarily balanced. For example, Zacheilas *et al.* [7] have shown that the largest workload of a reducer can be more than five times larger than the smallest one when running a MapReduce job based on real-world data. Therefore, if unfortunately these approaches encounter such MapReduce workloads in which non-uniform Selectivity$_M$ and data skew are present, their effectiveness will be severely hampered.

Many other solutions for predicting the workload of reducers have been proposed in the context of reducer load balancing [3–7]. These solutions estimate the workload of reducers through the intermediate key distribution and then rebalance the workload of reducers based on the estimation results. However, the techniques they used for estimating the workload of reducers have the following two major limitations. First, in order to obtain the statistics of the key distribution, these solutions either have to wait for the completion of all mappers [3–5] or add a sampling phase before the job execution [6]. Both of these techniques can increase the job running time. Second, monitoring statistics at the granularity of key-value pairs are costly because the number of the key-value pairs can be in the order of the size of input data.

In our prior work, DREAMS [8], we proposed a run-time reducer workload estimation approach using linear regression. DREAMS leverages the statistics of early completed mappers to predict the workload of each reducer before the completion of all mappers. However, DREAMS is based on an assumption that the size of the partitions that have been generated for a reducer is linearly proportional to the number of completed mappers, which limits its generality and applicability.

Motivated by the limitations of existing approaches, in this paper, we present ROUTE, a **R**un-time r**O**bust red**U**cer workload es**T**imation technique for MapReduc**E**. The contributions of this paper are as follows:

- We present a technique for predicting the workload of each reducer at run time without waiting for the completion of all mappers or adding a sampling phase before actual jobs run. In particular, we can achieve high accuracy with the completion of only 5% of all mappers.

DOI: 10.1002/nem

- We introduce a progressive sampler that collects minimum number of samples to satisfy the accuracy requirement specified by users. This not only reduces the time overhead of waiting for more samples but also eliminates users' burden of specifying the number of samples needed.
- We use robust estimation and bootstrapping resampling techniques to predict workload of reducers, allowing ROUTE requires no *a priori* knowledge of the map function and input datasets and can apply to a variety of MapReduce jobs.

Experiments using both real and synthetic data have been conducted to evaluate ROUTE. The results show that ROUTE achieves high accuracy with error rate no more than 10.92% on an 11-node real cluster and a 40.6% improvement in terms of error rate while compared with the existing solution. Besides, in terms of robustness of ROUTE, we show that ROUTE is robust to skewed distributions such as Weibull, log-normal, and exponential distributions. Further, we apply ROUTE as an extension technique in existing load balancing solution [8] and deadline-aware scheduling solution [12]. The results show that ROUTE achieves a significant job completion time reduction in load balancing and enables the scheduler to meet the deadline effectively for various input datasets in deadline-aware scheduling.

The rest of this paper is organized as follows. Section 2 introduces the background and motivations. The design of ROUTE is described in Section 3. Section 4 details the implementation of ROUTE over Hadoop YARN. Sections 5 and 6 present the results of experimental evaluation. Finally, we review the related work in Section 7 and draw our conclusion in Section 8.

2. BACKGROUND

MapReduce [1] is a parallel computing model for large-scale data processing. As shown in Figure 2, the input of a MapReduce job is stored as identical data blocks in the distributed file system. Each mapper processes one block and produces a sequence of intermediate key-value pairs. These key-value pairs are divided into multiple *partitions* and written to local disk. For example, the output of $M1$ is divided into two partitions, $P_{1,1}$ and $P_{1,2}$, which are assigned to $R1$ and $R2$, respectively. Subsequently, each reducer fetches its corresponding partitions, preforms a reduce function on received data, and stores the final result in the distributed file system.

Apache Hadoop MapReduce, one of the most commonly used MapReduce implementations, uses a hash function *Hash(intermediate key) mod NumberReducer)* for partitioning. Because all mappers use the same partitioner, all the intermediate key value pairs with the same key are stored in the same partition. The collection of these pairs is called *a cluster*. Consequently, the workload of each reducer consists of a number of *clusters* that have the same hash value.

However, accurately predicting the workload of each reducer is challenging. That is because the workload of reducers is determined by the characteristics of the input datasets and the user specified map function, both of which can vary from job to job. Even for jobs that are routinely executed, different workload of reducers may be produced for different datasets. Figure 1 shows one such example.

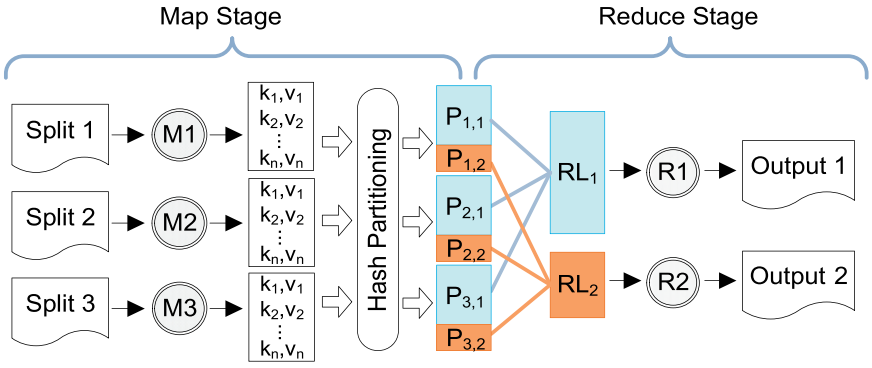


Figure 2. MapReduce programming model

DOI: 10.1002/nem

While processing different kinds of Wikipedia datasets using WordCount, Seletivity$_M$ varies. And therefore, those approaches [12,13], which predicts the workload of reducers based on Seletivity$_M$, are not effective.

Another challenge that arises in practice is that reducers can start before the completion of all mappers. By default, a reducer can start as soon as 5% of mappers have finished. This allows the reduce stage to overlap the map stage and thereby reducing the job completion time.

While this overlap operation can be beneficial, it also increases the difficulty for estimating the workload of reducers before they can be scheduled. Most of existing solutions [3–6] build the load model of reducers based on the data statistics at the granularity of key-value pairs (e.g., tuple count and number of *clusters*), and consequently, they have to wait for the completion of all mappers or gather samples before job executions. However, both waiting for the completion of all the mappers and adding the sampling phase before actual job runs are time-consuming. As reported in [14], executing reducers after the completion of all the mappers can severely prolong the job completion time. Therefore, in this work, we seek an alternative solution, which neither needs job profile nor is based on the statistics of key-value pairs. We estimate the workload of reducers based on the statistics at the granularity of partition level and predict the workload for each reducer at run time without causing a synchronization barrier and thereby providing accurate and *a priori* information for making better scheduling decisions.

3. ROUTE DESIGN

In this section, we describe in detail the design of our reducer workload estimation technique, ROUTE. First, we provide an overview of ROUTE in Section 3.1 and then elaborate main components in subsequence sections.

3.1. Overview

Figure 3 illustrates the architecture of ROUTE. It consists of three main stages: sampling, reducer workload estimation, and accuracy verification. During the sampling stage, mappers are running while reducers are waiting to be scheduled. ROUTE collects sample statistics of partitions from running mappers. Based on the statistics gathered in the sampling stage, the reducer workload estimation is performed, which estimates the workload of each reducer using statistic inference techniques. Because the estimation is inferred from the random sample, it may contain estimation errors (i.e., standard error, bias, and confidence interval). In ROUTE, we allow users to specify a tolerable error level and let ROUTE verify whether the obtained accuracy is satisfied with the user requirement. If the obtained

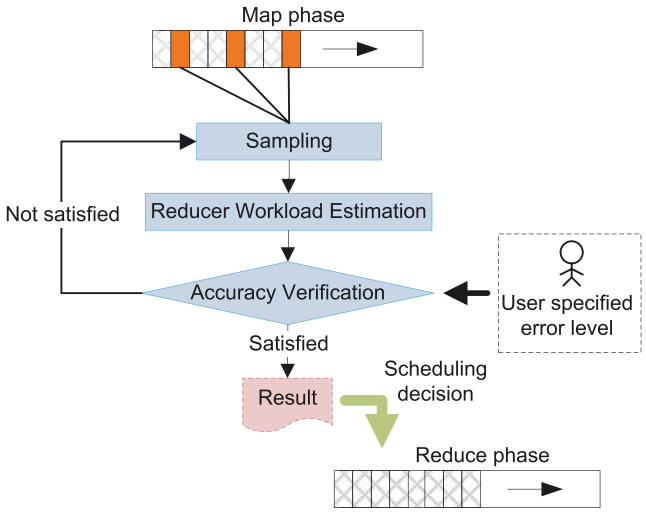


Figure 3. Architecture of ROUTE

DOI: 10.1002/nem

accuracy is unacceptable, the aforementioned process is repeated by increasing the sample size. When a desired accuracy is reached, the final result is returned, thereby providing information for making better scheduling decisions.

3.2. Reducer workload estimation

As mentioned previously, every mapper produces a partition for each reducer, and the collection of these partitions forms the workload of the reducer. We denote the size of a partition produced by mapper i for reducer j as $P_{i,j}$. In our model, we define N as the number of mappers and M as the number of reducers. Then, we can model the workload of a reducer as the sum of $P_{i,j}$ generated by N mappers,

$$\forall j,\ RL_j = \sum_{i=1}^{N} P_{i,j} \tag{1}$$

where RL_j is the workload of reducer $j \in [1, M]$. Because the jobs inputs are divided into many identical blocks and each mapper processes only a single block, $P_{i,j}$ can be considered as an independent identically distributed random variable. And for each reducer j, the set of partitions $T_j = \{P_{1,j}, P_{2,j}, \ldots, P_{N,j}\}$ can be viewed as a finite population of size N. Therefore, the population total of T_j is exactly the workload of reducer j.

Because the map phase can be overlapped with the reduce phase, calculating T_j by gathering all $P_{i,j}$ is not feasible. We consider the sizes of partitions that have been generated by the early completed mappers as a random sample[1] from the population T_j, which is denoted by $\mathbf{x}_j = \{P_{1,j}, P_{2,j}, \ldots, P_{n,j}\}$, where n is the number of completed mappers. Therefore, for a given job, our goal is to estimate the population total τ_j for all $j \in [1, M]$, by random sample $\mathbf{x}_j$.

Because the distribution of the population T_j varies from job to job and from dataset to dataset, accurate estimation of the population total τ_j is challenging. Besides, some extreme cases (i.e., extremely large/small partitions), which are called *outliers*, will even complicate this problem. One common way to estimate population total is to multiply sample mean by population size. However, sample mean is the maximum likelihood estimator of population mean for normal distributed data, and the distribution of the population T_j is unknown before jobs run. Trimmed mean is a widely used robust statistic that removes the largest and smallest β percent of the values and then calculates the mean of the remaining set. By doing this, it not only can reduce the impact of outliers but also will still give a reasonable estimate of central location [15]. Baltagi [16] has also shown that trimmed mean has higher efficiency for mixed distributions. Therefore, we use trimmed mean multiplied by the population size as the estimator of τ_j, which is defined as

$$\hat{\tau}_j = N \cdot \overline{\mathbf{x}}_{j(\beta)} \tag{2}$$

where $\overline{\mathbf{x}}_{j(\beta)}$ is the β percent trimmed mean of $\mathbf{x}_j$:

$$\overline{\mathbf{x}}_{j(\beta)} = \frac{1}{n-2d} \sum_{t=d+1}^{n-d} x_{j(t)}, \quad \text{with } d = \left\lfloor \frac{\beta \cdot n}{2} \right\rfloor$$

and $x_{j(t)}$ is the t^{th} order statistic when the observations are arranged in increasing order. Prescott *et al.* [17] have proposed a solution that adaptively determines the value of β according to underlying distributions, which is out of the scope of this paper. In this paper, we use $\beta = 5$.

3.3. Estimating accuracy

Estimating the accuracy of an estimator plays an essential role in statistical analysis. There are several measures that are widely used in practice such as bias, standard error, and confidence interval, which

[1] *We randomly shuffle the submission order of mappers; therefore, the order of task completions can be approximately considered as random.*

DOI: 10.1002/nem

can determine the quality of the estimator. In ROUTE, we use confidence interval, which is a range of values that does contain the unknown population parameter with high confidence. Note that our approach is independent of the accuracy measure and is applicable to other measures. In the following sections, we present the confidence intervals for our reducer workload estimation.

3.3.1. Analytical confidence interval for reducer workload estimation

Observe that the size of sample $\mathbf{x}_j$ depends on the number of mappers in the MapReduce job, which tends to be large in practice. For instance, Kavulya *et al*. [18] reported that the average number of mappers is 154, according to a 10-month trace data from a Yahoo! production Hadoop cluster. Therefore, we leverage the analytical formula based on large sample theory (e.g., the common value is greater than 30) to derive the confidence interval. Tukey and Mclaughlin [19] suggest that the $1-\alpha$ confidence interval for the β trimmed mean $\overline{\mathbf{x}}_{(\beta)}$ with large sample size can be defined as

$$\overline{\mathbf{x}}_{(\beta)} \pm t_{1-\frac{\alpha}{2},n-2d-1}\hat{se}\left(\overline{\mathbf{x}}_{(\beta)}\right) \tag{3}$$

where $t_{1-\frac{\alpha}{2},n-2d-1}$ is the $100\left(1-\frac{\alpha}{2}\right)th$ percentile of the Student's t-distribution with $n-2d-1$ degrees of freedom. And the standard deviation of β trimmed mean $\hat{se}\left(\overline{\mathbf{x}}_{(\beta)}\right)$ is derived based on the Winsorized sum of squared deviation $\hat{s}_w^2$ [19],

$$\overline{\mathbf{x}}\hat{se}\left((\beta)\right) = \frac{\hat{s}_w}{\sqrt{(n-2d)(n-2d-1)}} \tag{4}$$

Accordingly, we can construct the $100(1-\alpha)$ confidence interval for $\hat{\tau}_j$ by

$$N \cdot \overline{\mathbf{x}}_{j(\beta)} \pm N \cdot t_{1-\frac{\alpha}{2},n-2d-1}\hat{se}\left(\overline{\mathbf{x}}_{j(\beta)}\right),\ j \in [1, M] \tag{5}$$

Here, we use $\alpha = 0.05$ in this paper. However, assuming all of the MapReduce jobs have a large number of mappers is problematic. We believe there are also many jobs that only have a small number of mappers (e.g., less than 30 mappers). For these small jobs, the use of the aforementioned analytical confidence interval based on large sample size is not justified. In the next section, we will provide an alternative approach to construct the confidence interval.

3.3.2. Bootstrap confidence interval for reducer workload estimation

The bootstrap method, first proposed by Efron in 1979 [20], is a data-based simulation technique for statistical inference. This approach does not require theoretical formulas to produce the estimation. Through repeatedly computation of the estimators (e.g., $\bar{\mathbf{x}}$) on a large number of resampled data, bootstrap provides accurate estimation under violations of regularity conditions.

The authors of [21] have shown better accuracy of the bootstrap approximation over the approximation using normal distributions, and Fisher *et al.* [22] have shown that the bootstrap method is suitable for small sample sizes. Therefore, using bootstrap method for estimating the workload of reducers is a better choice while accommodating unknown distribution population with sample of small size.

Accordingly, a bootstrap samples $\mathbf{x}_j^*$ can be generated by randomly sampling n times, with replacement, from the original random sample $\mathbf{x}_j$. Suppose we perform B repetitions of bootstrap sampling, we can obtain the B bootstrap samples denoted as $\mathbf{x}_j^{*1}, \mathbf{x}_j^{*2}, \ldots, \mathbf{x}_j^{*B}$. Let $\hat{\tau}_j = u(\mathbf{x}_j)$ be the estimate of a parameter of interest τ_j, the bootstrap replicates $\hat{\tau}_j^{*b} = u\left(\mathbf{x}_j^{*b}\right)$ with $b \in [1, B]$ can be obtained by calculating the estimator on each bootstrap sample. The sampling distribution of $\hat{\tau}_j$ is then estimated by its bootstrap distribution of $\hat{\tau}_j^{*b}$ with sufficient large B. Hence, we can construct the bootstrap bias-corrected percentile interval (BC) [23] of intended coverage $1-\alpha$ by

$$BC:\ \left(\hat{\tau}_j^{*(\alpha_1)}, \hat{\tau}_j^{*(\alpha_2)}\right) \tag{6}$$

DOI: 10.1002/nem

where,

$$\alpha_1 = \Phi\left(2 \cdot \hat{z}_0 + z^{\left(\frac{\alpha}{2}\right)}\right)$$
$$\alpha_2 = \Phi\left(2 \cdot \hat{z}_0 + z^{\left(1-\frac{\alpha}{2}\right)}\right)$$

Here, $\hat{\tau}^{*(\alpha_1)}$ denotes the $100\alpha_1$th percentile of bootstrap distribution of $\hat{\tau}_j^{*b}$, Φ indicates the standard normal distribution function, and $z^{\left(\frac{\alpha}{2}\right)} = \Phi^{-1}\left(\frac{\alpha}{2}\right)$. $\hat{z}_0$ is called *bias-correction* that adjusts the bias of the original estimator and $\hat{z}_0 = \Phi^{-1}\left(\frac{\#\left(\hat{\tau}_j^{*b} < \hat{\tau}_j\right)}{B}\right)$.

3.4. Progressive sampling

The relationship between sample size and model accuracy can be depicted by a *learning curve* [24] shown in Figure 4. As the sample size increases, the model accuracy improves monotonically in the early portion of the curve. After reaching a point n_{min}, where the model accuracy has converged, there will be little improvement for increasing the sample size. On the other hand, the sample in our context is a set of completed mappers. The larger the sample size, the more time it takes to wait for the completion of mappers, which is quite expensive. Therefore, it motivates us to design a sampler that collects the minimum number of samples, based on which the workload estimation will guarantee the accuracy meeting with the user specified requirement.

Suppose (θ_L, θ_H) is a $100(1-\alpha)\%$ confidence interval for estimator $\hat{\theta}$, where θ is the parameter of interest that needs to be estimated, and θ_L and θ_H are the lower and upper limits, respectively.

Let $\left|\theta - \hat{\theta}\right|$ define the error in estimating θ by $\hat{\theta}$, we can infer that this error is less than $\max\left\{\left|\theta_H - \hat{\theta}\right|, \left|\theta_L - \hat{\theta}\right|\right\}$ with $100(1-\alpha)\%$ confidence. This value is called *margin of error*. In particular, if this confidence interval is symmetrical, the *margin of error* can be obtained by $\frac{\theta_H - \theta_L}{2}$ [25].

Observed that as the sample size increases, the accuracy is increased. Therefore, equation (7) arises for the accuracy verification on the reducer workload estimation, and the sampler can stop when the accuracy is retained, that is,

$$\forall j, \frac{\max\left\{\left|\hat{\tau}_j^{(H)} - \hat{\tau}_j\right|, \left|\hat{\tau}_j^{(L)} - \hat{\tau}_j\right|\right\}}{E(\hat{\tau}_j)} \leq \epsilon \tag{7}$$

where $\left(\hat{\tau}_j^{(H)}, \hat{\tau}_j^{(L)}\right)$ is the $100(1-\alpha)\%$ confidence interval for $\hat{\tau}_j$, $E(\hat{\tau}_j)$ is the expectation value of $\hat{\tau}_j$, and ϵ is the user specified error level. Because different reducers may have different scales of workload even in the same MapReduce job, we normalize the margin of error for each reducer by the expectation

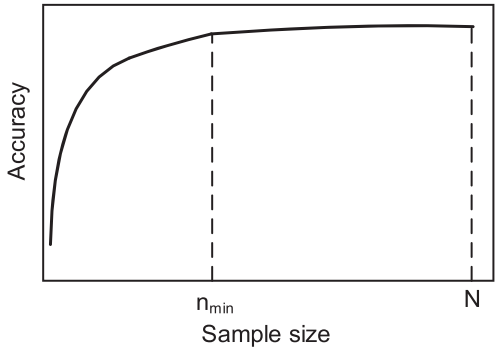


Figure 4. Learning curve and progressive samples [24]

DOI: 10.1002/nem

of its workload estimation $\hat{\tau}_j$. Then, the user specified error level ϵ can be set as a scale-independent value (e.g., 0.05 and 0.1). When the confidence interval width is satisfied with equation (7), it is highly reliable that the estimation error is less than ϵ times as its expectation value.

4. ROUTE IMPLEMENTATION

We have implemented ROUTE on both Hadoop v1 and YARN as an extended function, which provides accurate workload estimation for reducers before making reducer scheduling decisions. Because of the limited space, we only detail the implementation of ROUTE on YARN in this paper. It consists of the following components shown in Figure 5:

- **Partition Monitor**: It monitors the statistics of partitions generated by mappers at run time and sends them to the ApplicationMaster through *heartbeat* messages.
- **Progressive Sampler**: It is responsible for collecting the statistics and drawing the random sample for the reducer workload estimation. Because the required sample size is not known beforehand, the Progressive Sampler continues sampling until the required accuracy is obtained.
- **Reducer Workload Estimator**: It estimates the workload for each reducer using the random sample drawn from the sampler and then requests the accuracy verificator to verify the estimating accuracy. Once the required accuracy is reached, it notifies the sampler to stop sampling and finalizes the reducer workload estimation.
- **Accuracy Verificator**: It calculates the confidence interval for the workload estimation and then verifies whether the corresponding confidence interval width is satisfied according to user specified error level.

Note that ROUTE leverages the task status report mechanism in Hadoop and attaches the partition size statistics $P_{i,j}$ to the heartbeat messages (i.e., TaskUmbilicalProtocol). Hence, collecting the samples will not incur noticeable overhead over Hadoop. Besides, ROUTE performs the reducer workload estimation every time the heartbeat message between ApplicationMaster and ResourceManager (i.e., ApplicationMasterProtocol) is received. ROUTE is not triggered every time a mapper completes. Therefore, while burst processing scenarios happen, ROUTE can still work normally. The

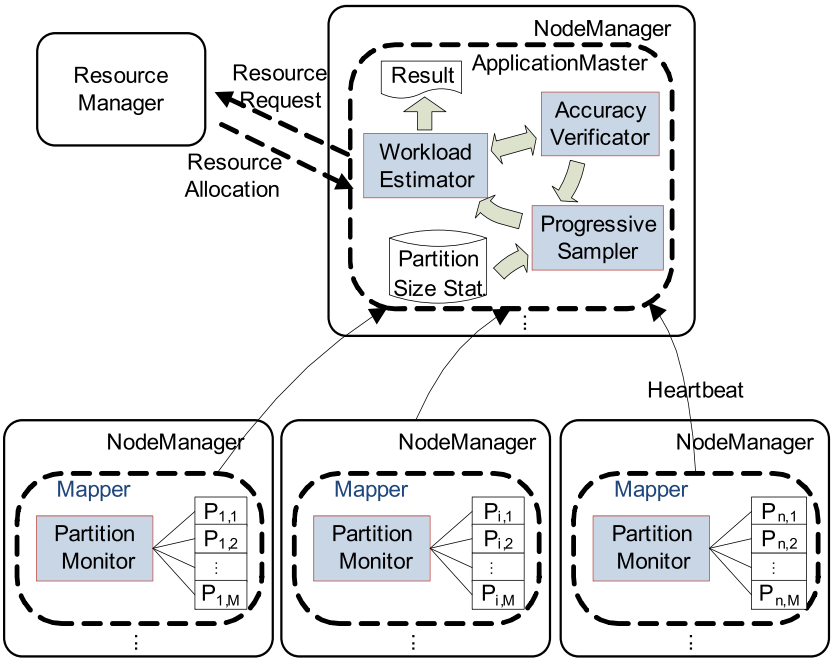


Figure 5. ROUTE implementation

DOI: 10.1002/nem

Algorithm 1 Reducer workload estimation algorithm

Input: ϵ - User specified error level;
$\phi_{\max}$- Maximum sample size;
ϕ_{init}- Initial sample size.
Output: V_{RL} - Vector of Workload estimation for each reducer j, $j \in [1, M]$.

1: Initialize $V_{RL}[M] = \{0\}$ and Verified$[M] = \{0\}$
2: **When a heartbeat from the ResourceManager is received:**
3: Obtain the partition size statistics $P_{i,j}$ and number of completed mappers n from the Hadoop Counters
4: **if** $n < \phi_{\text{init}}$ **then**
5: Wait for the next heartbeat from the ResourceManager
6: **break**
7: **else if** $n \geq \phi_{\max}$ **then**
8: obtain a sample $\mathbf{x}_j$ drawn from the progressive sampler for each reducers $j \in [1, M]$
9: **for** each reducer $j \in [1, M]$ **do**
10: Calculate $\hat{\tau}_j$ based on $\mathbf{x}_j$, $V_{RL}[j] = \hat{\tau}_j$
11: **end for**
12: **return** V_{RL}
13: **else if** $n \geq \phi_{\text{init}}$ & $n < \phi_{\max}$ **then**
14: obtain a sample $\mathbf{x}_j$ drawn from the progressive sampler for each reducers $j \in [1, M]$
15: **end if**
16: **for** each reducer $j \in [1, M]$ **do**
17: Calculate $\hat{\tau}_j$ based on $\mathbf{x}_j$
18: Verify whether the estimating accuracy for reducer j is satisfied with ϵ using the accuracy verificator
19: **if** reducer j is verified **then**
20: $V_{RL}[j] = \hat{\tau}_j$, Verified$[j] = 1$
21: **end if**
22: **end for**
23: **if** $\sum_{m=1}^{M}$ Verified$[m]) < M$ **then**
24: Wait for the next heartbeat from the ResourceManager
25: **else**
26: **return** V_{RL}
27: **end if**

detailed schema is shown in Algorithm 1. Specifically, upon receiving a heartbeat from the ResourceManager, the estimator obtains the partition size statistics and the number of completed mappers n (lines 2–3). Only when n is greater than ϕ_{init}, the estimator will obtain a sample $\mathbf{x}_j$ (lines 4–13). In the iteration from line 14 to 18, the workload estimation is performed based on the sample $\mathbf{x}_j$. The estimator returns the final estimation results when all the reducers satisfy the user specified requirement or when the sample size reaches a user specified maximum $\phi_{\max}$. Otherwise, it will wait for the next map task completion event to obtain a larger sample size, thereby trying to increase the estimating accuracy.

The computational complexity of this algorithm is $O(M \cdot (n + B))$, where M is the number of reducers, n is the sample size, and B is the number of bootstrap replicates. Specifically, the computational complexity is dominated by the iteration (line 14 to 18), where ROUTE verifies the confidence intervals for every reducer. It takes $O(M \cdot n)$ for calculating the analytical confidence intervals with large sample size, whereas $O(M \cdot B)$ is needed for calculating the bootstrap confidence intervals when sample size is less than 30 (the common critical value in large sample theory). We found that $B = 100$ and 5% sample rate can already achieve high accuracy in practice (see details in Section 5.2). Thus, the time overhead is small in all our experiments (<500 ms).

5. EVALUATION

In this section, we want to evaluate the accuracy and robustness of ROUTE. The experiments are performed on 11 virtual machines (VMs) in the SAVI test bed [26]. Each VM has four 2 GHz cores, 8 GB RAM, and 80 GB hard disk. We deploy the Hadoop YARN with one VM as ResourceManager

DOI: 10.1002/nem

and NameNode and the remaining 10 VMs as workers. Each worker is configured with eight virtual cores and 7 GB RAM (leaving 1 GB for other processes). The HDFS block size is set to 64 MB, and the replication level is set to 3. The MapReduce jobs used in our evaluation are as follows:

1 **Sort**: This job takes input data generated by RandomWriter as input and outputs the data sorted by the key. Each map task sorts one split of the input dataset, and then each reduce task merges the output of the map tasks for a given partition key. The default RandomWriter uses the random number generator that follows a uniform distribution. In order to generate skewed intermediate key-value pair, we modify the random number generator in RandomWriter to generate random data follow Zipf 0.5 distribution.
3 **WordCount**: WordCount computes the occurrence frequency of each word in the large collection of documents. Each map task emits $< word, count >$ pairs. The reduce task sums up the counts for a given key, which maybe several words, from all map tasks and outputs the final counts.
4 **RelativeFrequency**: RelativeFrequency is introduced in [27]. Other than measuring the number of times word w_i co-occurs with word w_j within a specific context, this job measures the proportion of time word w_j appears in the context of w_i. It is also denoted as $F(w_j|w_i)$. To compute $F(w_j|w_i)$. RelativeFrequency counts up the number of co-occurrences of the bigram (w_i, w_j), and then divides it by the number of occurrences of all the bigrams $(w_i, *)$. We use the implementation of this job provided by Lin and Dyer [27].
5 **KMeans**: KMeans is a data mining algorithm that classifies input data to k clusters. We use the implementation of this job provided by PUMA [28]. More specifically, it classifies movies based on their ratings using Netflix movies rating data and initial centroid data. Each map task determines which clusters the movies belong to based on the similarity computation, and then emits $< centroid\_id, (similarity\_value, movie\_data) >$. Each reduce task merges all the movie data with the same $centroid_id$, and computes the average of similarity of all the movies in this cluster. The movie closest to the average is used as the new centroid data for the next iteration. After that, each reduce task emits the movie data with its corresponding cluster and the new centroid data.

Table 1 summarizes these MapReduce jobs and their configurations used in our experiments. Both synthetic and real-world data (e.g., Wikipedia data dumps *pages-articles* and Netflix) are used in the benchmarks. Besides, in order to better show the generality of ROUTE, we will present results of running these MapReduce jobs with small and large datasets in the following sections.

5.1. Accuracy of ROUTE

In this set of experiments, we validate the accuracy of ROUTE and compare it with the existing runtime reducer workload estimation technique, DREAMS [8], which is based on online linear regression. We use the mean absolute percentage error (MAPE) as the accuracy metric, which is defined by

$$MAPE = \frac{1}{M}\sum_{j=1}^{M}\frac{\left|RL_j^{\text{pred}} - RL_j^{\text{measrd}}\right|}{RL_j^{\text{measrd}}} \tag{8}$$

where M is the number of reducers in this job, RL_j^{pred} and RL_j^{measrd} are the predicted and measured load of reducer j, respectively. Note that the smaller the MAPE, the more accurate the estimation is.

Table 1. Benchmarks characteristics

Application	Dataset type type	Input size, small (GB)	#Map, reduce tasks	Input size, large (GB)	#Map, reduce tasks
Sort	RandomWriter	5.381	89, 16	30.757	512, 64
WordCount	Wikipedia	5.759	92, 16	29.049	467, 64
RelativeFrequency	Wikipedia	5.759	92, 16	29.049	467, 64
KMeans	Netflix	4.686	76, 12	28.077	451, 12

DOI: 10.1002/nem

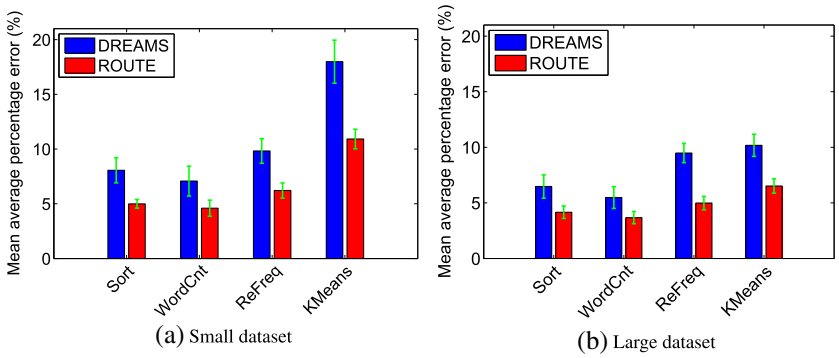


Figure 6. Comparison of estimation error between DREAMS and ROUTE

In DREAMS, a threshold of the percentage of completed mappers δ is required to be specified (e.g., $\delta = 5\%$). In order to fairly compare these two approaches, we configure $\phi_{\max} = 5\%$ in ROUTE, so it uses equal sample size as DREAMS in this evaluation. ϕ_{init} and ϵ are configured as 5% and 0.1, respectively.

Figure 6 compares the MAPE between DREAMS and ROUTE by running the benchmarks on small and large datasets. We repeat each experiment 10 times and adopt the averages. It is clear that ROUTE outperforms DREAMS in all cases. In particular, DREAMS incurs high estimation error reaching at more than 15% for KMeans. In comparison, ROUTE can significantly improve the accuracy for this job, and the error rate is reduced to 10.92%. In average, over all these jobs, ROUTE can achieve 40.6% improvement over DREAMS in terms of error rate.

The reason for the gain over DREAMS is that the assumption that DREAMS relies on is not necessarily true. In other words, the assumption that the size of intermediate data that have been generated for a reducer is linearly proportional to the number of completed mappers is not accurate. Figure 7 shows an example of the workload estimation in DREAMS for a random selected reducer while running 30G KMeans. As shown in Figure 7a, the statistics before the completion of 5% mappers are used as training data[2] (blue points). DREAMS performs linear regression based on this training data. After the linear model is determined, DREAMS predicts the workload of the reducer. However, there is an error of approximately 50 MB in this case. In order to show the linear regression result more clearly, we plot the detailed drawing of the linear regression in Figure 7b. We can see that the regression line does not accurately fit the training data. Because these training data only represents the beginning of the regression line, and DREAMS predicts the workload of the reducer using the tail of the regression line, slight error may have a significant impact on the workload prediction. In comparison, ROUTE uses trimmed mean to estimate the workload of the reducer, which mitigates the impact of outliers and improves its robustness. Figure 8 shows the workload estimation for the same reducer of the same job while using ROUTE. It is clear in Figure 7 that as the fraction of mappers increased, the predicted values are getting closer to the real value. In particular, when the fraction of mappers stays at 5%, the prediction error is less than 30 MB.

We also compare ROUTE with map selectivity-based approach (SElECTB) while running WordCount with different kinds of datasets shown in Table 2. For ROUTE, we use the same configuration as the previous experiments. With regard to SElECTB, we profile Selectivity$_M$ from running WordCount on *pages-articles* datasets and then use it to estimate the workload of each reducer according to Verma *et al*. [12]. We repeat each experiment 10 times and take the averages. Figure 9 shows the results. When estimating that the datasets have similar Selectivity$_M$ as *pages-articles* datasets, SElECTB can predict the workload of reducers. However, when Selectivity$_M$ changes, SElECTB loses its applicability. In comparison, ROUTE can achieve high accuracy in all experiments. Note that SElECTB will obtain the same result for repeated experiments; thus, the corresponding error bar is zero.

[2] *These statistics are the fraction of completed mappers* (F^j) *and the size of the partition generated by the completed map tasks for each reducer* $\left(S_i^j\right)$.

DOI: 10.1002/nem

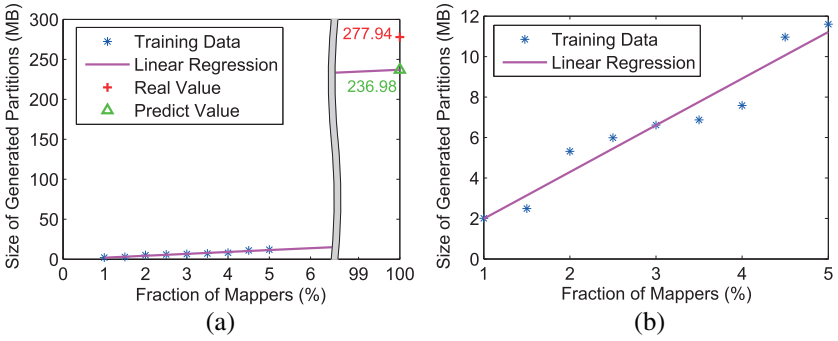


Figure 7. Workload estimation using DREAMS

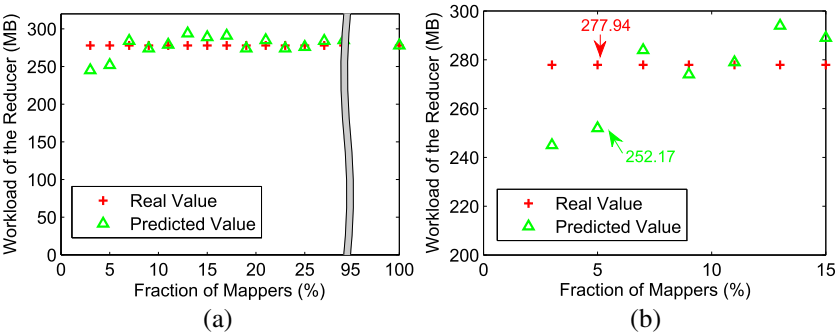


Figure 8. Workload estimation using ROUTE

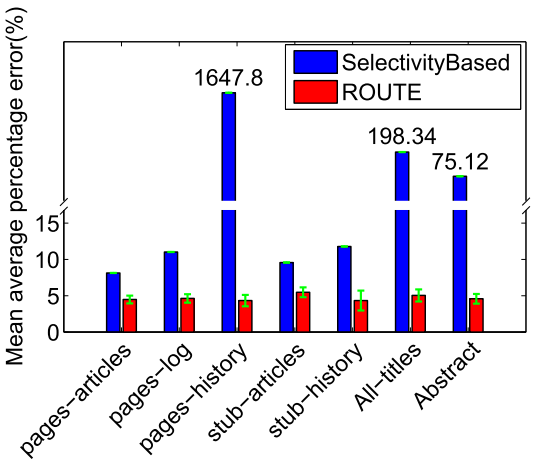


Figure 9. Comparison of estimation error between selectivity-based approach and ROUTE for WordCount

5.2. The number of bootstraps replicates and sample size

To compute an $ideal$ bootstrap estimation, the number of bootstrap replicates B should be as large as possible. However, because the computational complexity increases monotonously with B, we must minimize the value of B. However, doing so will lead to a trade-off between the number of replicates and the accuracy of the estimation. A minimum number of replicates is usually desired for the specific accuracy requirement in the application of bootstrap method. Take the bootstrap standard error $\hat{\sigma_B}$ as example, Efron [29] showed that there is no significant improvement on coefficient of variation of

DOI: 10.1002/nem

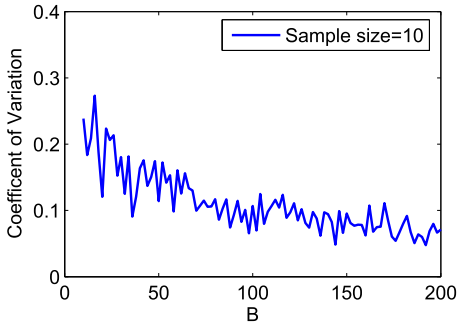


Figure 10. Effect of number of bootstrap replicates

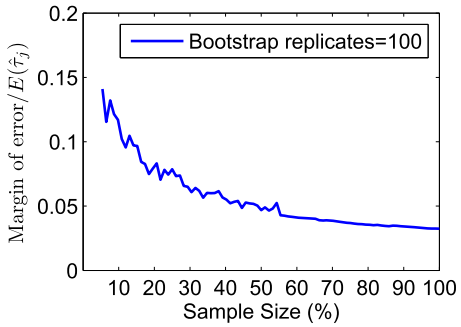


Figure 11. Effect of the sample size

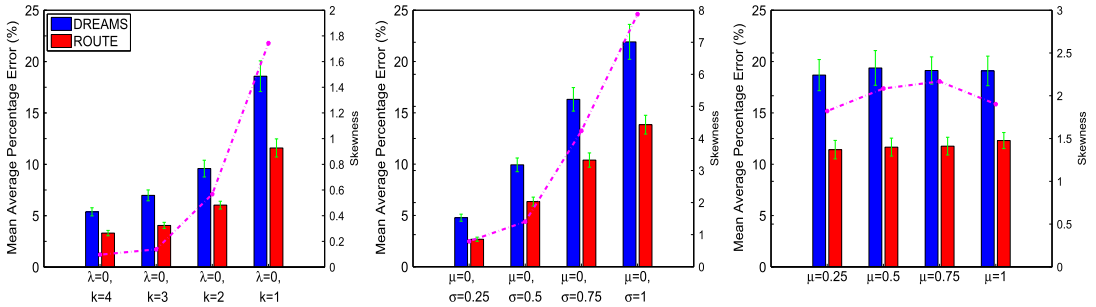
$\hat{\sigma_B}$ beyond $B = 100$. He also pointed out that, in the case of bootstrap confidence intervals, more replicates are recommended. However, we found 100 bootstrap replicates is already enough for the reducer workload estimation in practice. Figure 10 shows how B affects the coefficient of variation of the width of the bootstrap confidence interval. It is clear that little improvement can be achieved when we increase B beyond 100. Therefore, we use $B = 100$ in ROUTE.

Figure 11 shows how the sample size affects the *margin of error* (defined in Section 3.4) for a reducer. As described previously, we normalized the margin of error by the expectation of its workload estimation $\hat{\tau}_j$. We can see that as the sample size increases, the error level continuously decreases. But after sample size reaches 40%, little improvement can be achieved. This is consistent with the *learning curve* demonstrated in Section 3.4. Therefore, based on the user specified error level, the progressive sampler stops sampling when it has collected enough samples to guarantee the accuracy requirement. Note that the statistics in Figures 10 and 11 are obtained from a reducer when running 5G WordCount. Similar results can be also found when running other MapReduce jobs.

5.3. Robustness evaluation

In this section, we want to evaluate the robustness of ROUTE to different intermediate data distributions. Unlike the evaluations in the previous sections where the experiments are carried out on a real Hadoop Cluster, the experiments in this section are run on a simulator. The simulator allows us to input synthetic data with controlled distributions for T_j and perform the reducer workload estimation according to Algorithm 1. Each synthetic dataset, which is simulated as the partition size statistics of a MapReduce job, can be generated by (i) Weibull, (ii) log-normal, and (iii) exponential distributions with various parameters. All of these distributions include some departures from normality. Each synthetic dataset has 1000×100 partitions, which suggests that the simulated MapReduce job has 1000 mappers and 100 reducers. The configurations for DREAMS and ROUTE are the same as that in Section 5.1. We repeat each simulation 100 times and report the averages in Figure 12. From the figure, we can see that ROUTE achieves low error rate when the partition size statistics follow skewed

DOI: 10.1002/nem



(a) Weibull (b) Log-normal (c) Exponential

Figure 12. Robustness of ROUTE to different distributions

distributions and outperforms DREAMS significantly. In particular, for Weibull and log-normal distributions, ROUTE stays at 11.59% and 13.84% error rate, respectively, when the skewness[3] is the greatest. With respect to exponential distribution, because the skewness changes little while varying the parameter μ of the distribution, the MAPE for ROUTE hovers over at 11%. In summary, ROUTE outperforms DREAMS in all experiments, and it can achieve from 55.13% to 78.4% improvement compared with DREAMS.

6. APPLICATIONS OF ROUTE

In this section, we demonstrate the effectiveness of ROUTE in real scenarios. While ROUTE can be applied in many cases such as load balancing, reducer locality-aware scheduling, and deadline-aware scheduling, we focus on using ROUTE in load balancing and deadline-aware scheduling in the sequence sections.

6.1. Load balancing

The existing load balance solutions are detailed in Section 7. Because ROUTE only collects the statistics of intermediate data at the granularity level of partitions, those solutions [3,4,30], which are based on the statistics of the key-value pairs, cannot exploit ROUTE. Here, we choose DREAMS [8], which dynamically adjusts resource allocation among reducers based on their partition sizes to mitigate data skew. In this experiment, we compare the job completion time improvement over Native Hadoop YARN between DREAMS and DREAMS with ROUTE (DREAMS_R). In DREAMS_R, we use ROUTE to estimate the workload of reducers instead of using the linear regression technique. After the workload of each reducer is predicted, we keep using DREAMS's resource allocation algorithm. The same setup with Liu *et al.* [8] is deployed, which is an 11-node cluster with Hadoop 2.4.0, and the benchmarks listed in Table 1 are used for evaluation.

Figure 13 shows the results. The percentage in this figure is the job completion time reduction over Native Hadoop YARN; higher bars represent larger improvement. It is clear that both DREAMS and DREAMS_R improve the job completion time compared with Native Hadoop YARN, and DREAMS_R outperforms DREAMS. In particular, DREAMS_R achieves the highest job completion time reduction of 34.65% while running 5G Sort. In comparison, the gain of DREAMS is less than DREAMS_R in this case, which stays at 22.17%. This is because ROUTE estimates the load of each reducer more accurately than DREAMS, which in turn improves the resource allocation in DREAMS. It is also clear that for the WordCount application, both DREAMS and DREAMS_R cannot have much improvement. It is because the combiner (combines the key-value pairs shared with the same key in the map phase) relieves the load imbalance among reducers for WordCount. As a result, the skewness in the workload distribution among reducers is very small, thereby leaving little room for load balancing techniques to improve the job completion time.

[3] *Measured by the standardized third central moment of the variable.*

DOI: 10.1002/nem

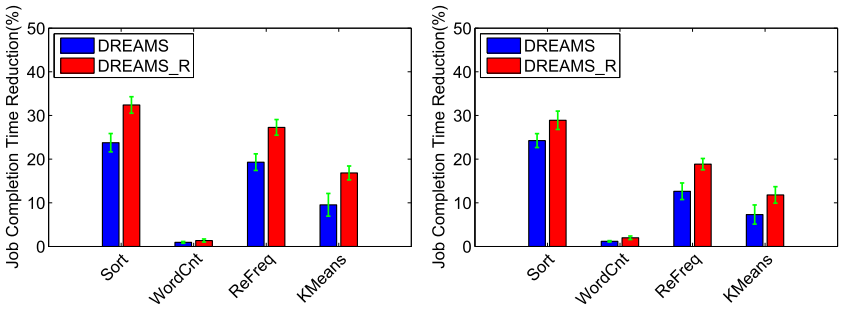


Figure 13. The job completion time reduction

Table 2. Different types of Wikipedia datasets

Types	Input size (GB)	#Map, reduce tasks
Pages-articles	11.16	185, 20
Pages-log	10.76	172, 20
Pages-history	9.85	158, 20
Stub-articles	8.79	141, 20
Stub-history	15.68	252, 20
All-titles	5.28	88, 20
Abstract	8.51	137, 20

6.2. *Deadline-aware scheduling*

Many solutions have been proposed in the area of deadline-aware scheduling. Salah *et al*. [31] present a queueing model to achieve resource elasticity for satisfying the MapReduce job's SLO response time. Verma *et al*. [12] propose a deadline-aware resource provision algorithm (called DARP in this paper) for meeting the MapReduce job's deadline. Here, we choose DARP and extend DARP with ROUTE for evaluation. Similar to the setup in [12], we use Hadoop 0.21.0 with one VM as JobMaster and NameNode and other 10 VMs as workers on the same cluster in Section 5. Each worker has four map and four reduce slots. Because of the limited space, we only take WordCount as an example and use the Wikipedia datasets shown in Table 2 for evaluation.

In [12], the lower and upper bounds of the job completion time $\left(T^{\text{low}} \text{ and } T^{\text{up}}\right)$ are computed as follows:

$$T^{\text{low}} = \frac{N_M^J \cdot M_{avg}}{S_M^J} + \frac{N_R^J \cdot \left(Sh_{avg}^{typ} + R_{avg}\right)}{S_R^J} + Sh_{avg}^1 - Sh_{avg}^{typ} \tag{9}$$

$$T^{\text{up}} = \frac{\left(N_M^J - 1\right) \cdot M_{avg}}{S_M^J} + M_{\max} + \frac{\left(N_R^J - 1\right) \cdot \left(Sh_{avg}^{typ} + R_{avg}\right)}{S_R^J} + Sh_{\max}^1 - Sh_{\max}^{typ} \tag{10}$$

Table 3 lists the notations for the symbols presented in equations (9) and (10). Accordingly, we collect job profiles for WordCount using *pages-articles* datasets with 40 map slots and 10 reduce slots, which are summarized in Table 4. Besides, as the size of the dataset increased, Verma *et al*. [12] estimates the workload of each reducer by $ReducerLoad = \text{Size}_{\text{dataset}} * \text{Selectivity}_M / \text{NumberReducer}$ and scales up the shuffle and reduces durations using linear regression by following equation[4]:

$$C_{0,avg}^{Sh} + C_{1,avg}^{Sh} \cdot RL = Sh_{avg}^{typ} \tag{11}$$

[4]*Here, we present the equation for scaling up the* Sh_{avg}^{typ}. *More details can be seen in* [12].

DOI: 10.1002/nem

Table 3. Notations for the job performance model in DARP

Name	Description
N_M^J	The number of mappers in the job
N_R^J	The number of reducers in the job
M_{avg}	The average duration of map phases
$M_{\max}$	The maximum duration of map phases
Sh_{avg}^1	The average duration of first shuffle phases
$Sh_{\max}^1$	The maximum duration of first shuffle phases
Sh_{avg}^{typ}	The average duration of typical shuffle phases
$Sh_{\max}^{typ}$	The maximum duration of typical shuffle phases
R_{avg}	The average duration of reduce phases
$R_{\max}$	The maximum duration of reduce phases

Table 4. Job profiles for WordCount

Parameters	Values	Parameters	Values
M_{avg}	28.62 s	$M_{\max}$	34.57 s
Sh_{avg}^1	6.81 s	$Sh_{\max}^1$	8.25 s
Sh_{avg}^{typ}	11.31 s	$Sh_{\max}^{typ}$	15.46 s
R_{avg}	11.53 s	$R_{\max}$	15.27 s
Selectivity$_M$	0.38	Selectivity$_R$	0.17

Table 5. Scaling factors for WordCount

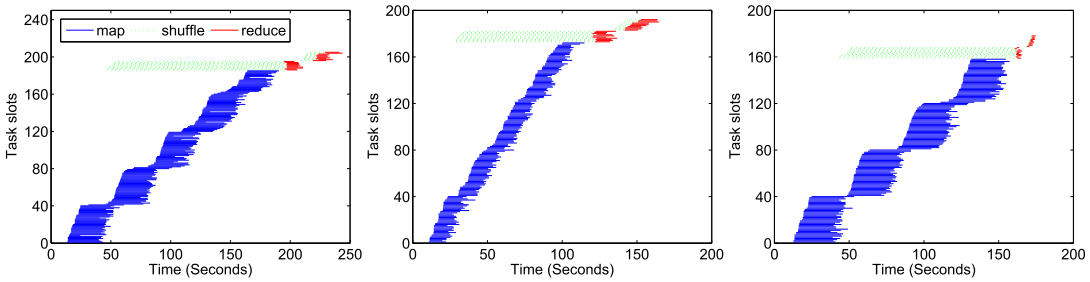
Parameters	Values	Parameters	Values
Dataset size (GB)	4.77–21.58	$C_{0,avg}^{Sh}$, $C_{1,avg}^{Sh}$	2.32, 0.79
#Map tasks	80–354	$C_{0,\max}^{Sh}$, $C_{1,\max}^{Sh}$	3.74, 0.98
#Reduce tasks	20	$C_{0,avg}^{R}$, $C_{1,avg}^{R}$	2.86, 0.76
#Map, reduce slots	40, 10	$C_{0,\max}^{R}$, $C_{1,\max}^{R}$	4.28, 0.99

where $C_{0,avg}^{Sh}$ and $C_{1,avg}^{Sh}$ are scaling factors need to be determined and RL is the workload of the reducer. We run a set of experiments by varying the size of the dataset and determined all the scaling factors for WordCount. Table 5 shows their values.

We then compare the accuracy of predicted job completion time between DARP and DARP with ROUTE (DARP_R). In DARP_R, we use ROUTE to estimate the workload of each reducer; meanwhile, we keep using the job performance model and the scaling regression rules in DARP. We found that even for the same MapReduce job, the average map duration varies for different datasets. Figure 14 shows the execution timelines of WordCount with different datasets. The average map duration for pages-articles, pages-log, and pages-history datasets are 28.62, 13.79, and 27.91 s, respectively. In order to isolate this effect and concentrate on the effect of the workload estimation, in both DARP and DARP_R, we dynamically collect the average and maximum map durations of the early completed mappers for prediction.

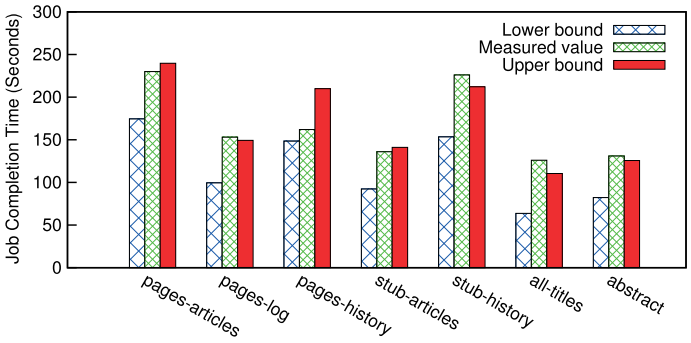
Figure 15 compares the predicted and measured durations for WordCount between DARP and DARP_R while processing different types of datasets. For the pages-articles dataset, DARP and DARP_R obtain the same estimated bound as the job profile is built on page-articles dataset. And their estimated lower bound and upper bound do contain the measured value. However, for other datasets, DARP is not accurate. For example, DARP underestimates the job completion time for the stub-history,

DOI: 10.1002/nem

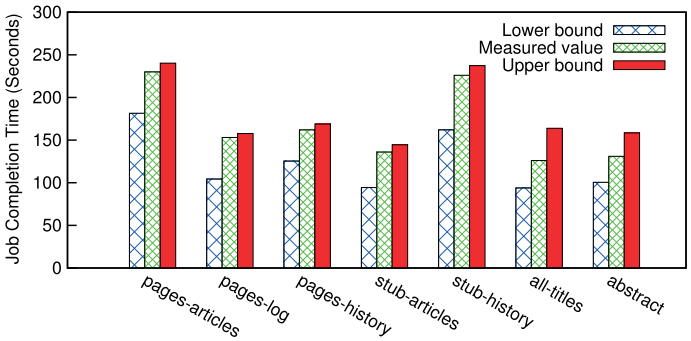


(a) pages-articles (b) pages-log (c) pages-history

Figure 14. The execution timelines of WordCount using different Wikipedia datasets



(a) DARP



(b) DARP_R

Figure 15. Comparison of predicted and measured job completion time between DARP and DARP_R for WordCount with different Wikipedia datasets

all-title, abstract datasets, and so on. This is because DARP cannot predict the workload of reducers when the map selectivity of the input dataset is different from the job profile, which in turn causes error while calculating the reduce and shuffle durations based on equation (11). In comparison, DARP_R can accurately bound the measured job completion time in all cases. When using the average of lower and upper bounds for prediction, the relative error between predicted and measured job completion times for DARP and DARP_R are 17.81% and 9.12%, respectively, where DARP_R outperforms DARP by 48.79%.

Besides, we want to evaluate whether DARP and DARP_R can meet deadlines based on their predicted bound of the job completion time. First, we calculate the slot allocation for WordCount according to the resource allocation algorithm in [12], to meet the deadline of 150 s. Figure 16a and b

DOI: 10.1002/nem

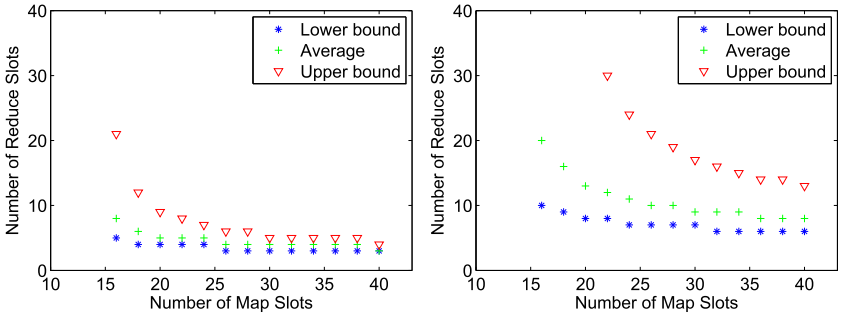


Figure 16. Slot allocation for WordCount (using *all-title* dataset) with deadline of 150 s

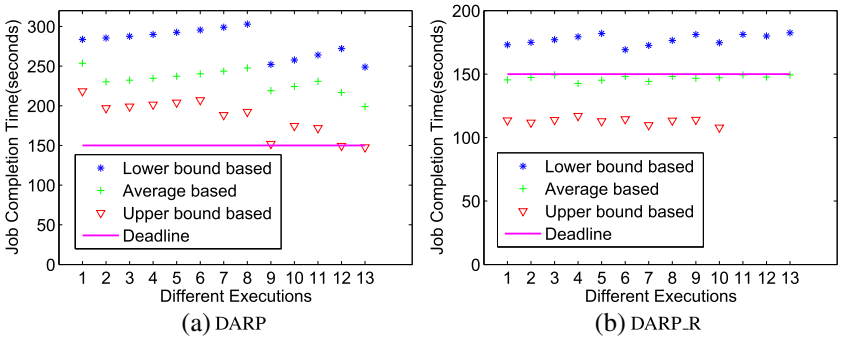


Figure 17. Actual job completion time with recommended allocations for WordCount (using *all-title* dataset)

shows the slot allocation results for DARP and DARP_R, respectively. Each point in these figures suggests a recommended slot allocation for meeting the deadline. Next, we run a set of experiments with recommended allocations to measure the actual job completion times. Figure 17 shows the results of these experiments. In Figure 17a, it is clear that even for upper bound-based group, most of job executions miss the deadline. In comparison, in Figure 17b, DARP_R can guarantee the job meeting the deadline while using the average and upper bound-based resource allocations. Note that the resource allocation choice depends on the service agreement between the user and the service provider. We can also see similar result for *page-log*, *stub-history*, and *abstract* datasets. It is true that for some datasets, like *stub-articles*, DARP can meet the deadline. That is because *stub-articles* has similar Seletivity$_M$ as the job profile. However, the input datasets of MapReduce jobs are usually unknown before the actual processing starts. Hence, there is no guarantee for DARP to meet the deadline even using the upper bound-based resource allocations.

7. RELATED WORK

In the past few years, many efforts have been made towards improving the performance of MapReduce such as load balancing, reducer locality-aware scheduling, and deadline-aware scheduling. Estimating the workload of reducers is one of the building blocks in these techniques.

Most of the existing solutions for load balancing in MapReduce [3–7] consist of the following two steps: (i) estimate the workload of each reducer and (ii) reassign the workload among reducers to achieve a better balance. For example, Gufler *et al.* [3] define a cost model that estimates the workload of each partition based on the histogram of some statistics of key-value pairs (e.g., tuple count and number of *clusters*) and then reassign intermediate keys to reducers by bin packing algorithms. In order to reduce the overhead for monitoring statistics at the key-value pair level, Gufler *et al.* [4] later propose an approach that only monitors and aggregates the statistics of the top k clusters. And, Yan *et al.* [5] propose a *sketch*-based key group size estimation, which aggregates the sketch

DOI: 10.1002/nem

information of key-value pairs in a centralized manner. The approach in [7] also uses the *sketch* scheme to estimate the size of partitions and then tries to assign the partitions with large sizes to machines with better performance to achieve load balance.

However, these solutions will cause a synchronization barrier between map and reduce stages: they have to wait for the completion of all the mappers.

Besides, they have to monitor the statistics at the granularity level of key-value pairs and aggregate them in a centralized manner. Although many solutions use some techniques such as top k algorithms [4,6] and *sketch* [5,7] to reduce the overhead, they may still be costly because they still need to collect the statistics of each individual key. Compared with these solutions, ROUTE accurately estimates the workload of reducers at run time, requiring only aggregating statistics at partition level. Admittedly, the solutions in [14,30] intend to achieve load balancing in an online manner. However, SkewTune [30] will cause a significant run-time overhead (30 s as reported in [30]). Yan *et al.* [14] assume the size of each key-value pair is identical and the load of each machine is the number of assigned key-value pairs, which are not true in real MapReduce jobs.

In terms of deadline-aware scheduling, Verma *et al.* [12] propose a framework that can calculate the resource allocation to routine MapReduce jobs while guaranteeing their service level objectives. They use job profiles derived from small datasets to estimate the size of intermediate data of jobs with larger datasets. Chen *et al.* propose a resource provision approach [13] that tries to minimize the financial cost for running MapReduce jobs in public clouds. However, these approaches estimate the workload of reducers based on the assumptions that the ratio of the map output size to the map input size (Selectivity_M) keeps invariant and the intermediate data are evenly distributed to reducers, which are not necessarily true in reality. In contrast, ROUTE does not relies on job profiling, and it can accurately estimate the load of each reducer in the early beginning of the job execution, even when data skew is present. Salah *et al.* [32] present a queueing model to achieve resource elasticity in the cloud while satisfying the job's SLO response time. Subsequently, Salah *et al.* [31] propose a continuing work that focuses on achieving proper elasticity for MapReduce jobs. However, the solution in [31] assumes the service times for reducers are exponentially distributed, without taking the workload of each reducer into consideration. ROUTE can provide accurate workload estimation for each reducer. As a result, ROUTE can enhance the prediction accuracy of the reducers' service times, which is complementary to the work in [31].

8. CONCLUSIONS

In this paper, we presented ROUTE, a run-time robust reducer workload estimation technique for MapReduce. ROUTE leverages the partition size statistics of early completed mappers and predicts the workload of each reducer without causing a synchronization barrier. In ROUTE, a progressive sampler is developed that determines the minimum number of samples automatically for satisfying the accuracy requirement specified by users. Furthermore, by using robust statistic interference and bootstrapping resampling technique, ROUTE requires no *a priori* knowledge of the map function and input datasets, nor making assumptions on the underlying distribution of the intermediate data. Experimental results showed that ROUTE can achieve high accuracy with the highest error rate at 10.92% and deliver an average error rate improvement of 40.6% compared with the state-of-the-art solution on 11-node real Hadoop cluster. We also showed that ROUTE is robust to a variety of skewed distributions. Finally, we demonstrated ROUTE can enhance the existing load balancing and deadline-aware scheduling solutions. More specifically, in load balancing, ROUTE clearly improves the scheduler to achieve a larger reduction in the job completion time. In deadline-aware scheduling, ROUTE enables the scheduler to meet the deadline effectively with various input datasets.

ACKNOWLEDGEMENTS

This work is supported in part by the National Natural Science Foundation of China (no. 61472438), and in part by the Smart Applications on Virtual Infrastructure (SAVI) project funded under the National Sciences and Engineering Research Council of Canada (NSERC) Strategic Networks grant number NETGP394424-10.

DOI: 10.1002/nem

REFERENCES

1. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters, *Communications of the ACM* 2008; **51**(1): 107–113.
2. Ananthanarayanan G, Kandula S, Greenberg A, Stoica I, Lu Y, Saha B, Harris E. Reining in the outliers in Map-Reduce clusters using mantri, In *OSDI'10, USENIX,* Vancouver, Canada, 2010; 1–16.
3. Gufler B, Augsten N, Reiser A, Kemper A. Handing data skew in MapReduce, In *Proceedings of the 1st Intl. Conf. on Cloud Computing and Services Science*, Vol. 146, Noordwijkerhout, The Netherlands, 2011; 574–583.
4. Gufler B, Augsten N, Reiser A, Kemper A. Load balancing in MapReduce based on scalable cardinality estimates, In *ICDE, 2012 IEEE 28th Intl. Conf. on,* IEEE: Washington DC, USA, 2012; 522–533.
5. Yan W, Xue Y, Malin B. Scalable and robust key group size estimation for reducer load balancing in MapReduce, In *Big Data, 2013 IEEE Intl. Conf. on,* IEEE: Santa Clara Marriott, USA, 2013; 156–162.
6. Ramakrishnan S R, Swart G, Urmanov A. Balancing reducer skew in MapReduce workloads using progressive sampling, In *Proceedings of the Third ACM Symposium on Cloud Computing,* ACM: San Jose, USA, 2012; 16.
7. Zacheilas N, Kalogeraki V. Real-time scheduling of skewed MapReduce jobs in heterogeneous environments, In *Proceedings of 11th Intl. Conf. on Autonomic Computing, USENIX,* Philadelphia, USA, 2014; 189–200.
8. Liu Z, Zhang Q, Zhani M F, Boutaba R, Liu Y, Gong Z. Dreams: dynamic resource allocation for MapReduce with data skew, In *Integrated Network Management, 2015 IFIP/IEEE International Symposium on*. Ottawa, Canada, 2015; 18–26.
9. Liu Z, Zhang Q, Boutaba R, Liu Y, Wang B. Optima: on-line partitioning skew mitigation for MapReduce with resource adjustment, *Journal of Network and Systems Management*, posted on 2016: 1–25, DOI 10.1007/s10922-015-9362-8, (to appear in print).
10. Hammoud M, Sakr MF. Locality-aware reduce task scheduling for MapReduce, In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third Intl. Conf. on*: IEEE, Athens, Greece, 2011; 570–576.
11. Tan J, Meng S, Meng X, Zhang L. Improving ReduceTask data locality for sequential MapReduce jobs, In *INFOCOM, 2013 Proceedings IEEE*. IEEE: Turin, Italy, 2013; 1627–1635.
12. Verma A, Cherkasova L, Campbell RH. Resource provisioning framework for MapReduce jobs with performance goals, In *Middleware 2011*. Springer, 2011; 165–186.
13. Chen K, Powers J, Guo S, Tian F. Cresp: towards optimal resource provisioning for MapReduce computing in public clouds, *Parallel and Distributed Systems, IEEE Transactions on* 2014; **25**(6): 1403–1412.
14. Le Y, Liu J, Ergun F, Wang D. Online load balancing for MapReduce with skewed data input, In *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014; 2004–2012.
15. Brown MB, Forsythe AB. Robust tests for the equality of variances, *Journal of the American Statistical Association* 1974; **69**(346): 364–367.
16. Baltagi BH. *The Oxford Handbook of Panel Data*: Oxford University Press, 2014.
17. Prescott P. Selection of trimming proportions for robust adaptive trimmed means, *Journal of the American Statistical Association* 1978; **73**(361): 133–140.
18. Kavulya S, Tan J, Gandhi R, Narasimhan P. An analysis of traces from a production MapReduce cluster, In *CCGrid, 2010 10th IEEE/ACM Intl. Conf. on, IEEE*, 2010; 94–103.
19. Tukey JW, McLaughlin DH. Less vulnerable confidence and significance procedures for location based on a single sample: trimming/Winsorization 1, *Sankhyā: The Indian Journal of Statistics, Series A* 1963; **25**(3): 331–352.
20. Efron B. Bootstrap methods: another look at the jackknife, *The Annals of Statistics* 1979; **7**(1): 1–26.
21. Hall P, Padmanabhan A. On the bootstrap and the trimmed mean, *Journal of Multivariate Analysis* 1992; **41**(1): 132–153.
22. Fisher NI, Hall P. Bootstrap algorithms for small samples, *Journal of Statistical Planning and Inference* 1991; **27**(2): 157–169.
23. Efron B. Nonparametric standard errors and confidence intervals, *Canadian Journal of Statistics* 1981; **9**(2): 139–158.
24. Provost F, Jensen D, Oates T. Efficient progressive sampling, In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM: San Diego, USA, 1999; 23–32.
25. Newcombe RG. Two-sided confidence intervals for the single proportion: comparison of seven methods, *Statistics in medicine* 1998; **17**(8): 857–872.
26. *Smart applications on virtual infrastructure (SAVI)*, 2015. http://www.savinetwork.ca/.
27. Lin J, Dyer C. Data-intensive text processing with MapReduce, *Synthesis Lectures on Human Language Technologies* 2010; **3**(1): 1–177.
28. Ahmad F, Lee S, Thottethodi M, Vijaykumar T. Puma: purdue MapReduce benchmarks suite, In *Purdue ECE Tech Report TR-12-11,* West Lafayette, USA, 2012.
29. Efron B. Better bootstrap confidence intervals, *Journal of the American statistical Association* 1987; **82**(397): 171–185.
30. Kwon Y, Balazinska M, Howe B, Rolia J. Skewtune: mitigating skew in MapReduce applications, In *Proceedings of the 2012 ACM SIGMOD Intl. Conf. on Management of Data*. ACM: Scottsdale, USA, 2012; 25–36.
31. Salah K, Alcaraz Calero J M. Achieving elasticity for cloud MapReduce jobs, In *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on*. IEEE: San Francisco, USA, 2013; 195–199.
32. Salah K. A queueing model to achieve proper elasticity for cloud cluster jobs, In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE: Santa Clara Marriott, USA, 2013; 755–761.

DOI: 10.1002/nem

AUTHORS' BIOGRAPHIES

Zhihong Liu received his BASc and MSc degrees in computer science from South China University of Technology, China, and National University of Defense Technology, China, in 2009 and 2011, respectively. He is a PhD candidate in National University of Defense Technology with research interests in big-data analytics and resource management in cloud computing. Currently, he is a visiting student at the University of Waterloo, Canada.

Qi Zhang received his BASc, MSc, and PhD from the University of Ottawa, Canada, Queen's University, Canada, and University of Waterloo, Canada, respectively. His current research focuses on resource management for cloud computing systems. He is currently pursuing a post-doctoral fellowship at the University of Toronto, Canada. He is also interested in related areas including big-data analytics, software-defined networking, and network virtualization and management.

Raouf Boutaba received the MSc and PhD degrees in computer science from the University Pierre and Marie Curie, Paris, France, in 1990 and 1994, respectively. He is currently a Professor of Computer Science at the University of Waterloo, Waterloo, ON, Canada. His research interests include control and management of networks and distributed systems. He is a fellow of the IEEE and the Engineering Institute of Canada.

Yaping Liu received the PhD degree in computer science from the National University of Defense Technology, China, in 2006. She is currently a Professor in the School of Computer at the National University of Defense Technology. Her current research interests include network architecture, interdomain routing, network virtualization, and network security.

Zhenghu Gong received the BE degree in Electronic Engineering from Tsinghua University, Beijing, China, in 1970. He is currently a Professor in the School of Computer at the National University of Defense Technology, Changsha, China. His research interests include computer network and communication, network security, and datacenter networking.

DOI: 10.1002/nem