# On the Optimal Number of Computational Resources in MapReduce

Htway Htway Hlaing[1(✉)], Hidehiro Kanemitsu[1,2], Tatsuo Nakajima[1], and Hidenori Nakazato[1]

[1] Waseda University, Tokyo, Japan
htwayhtwayhlaing@toki.waseda.jp, kanemitsuh@stf.teu.ac.jp, tatsuo@dcl.cs.waseda.ac.jp, nakazato@waseda.jp
[2] Tokyo University of Technology, Tokyo, Japan

**Abstract.** Big data computing in the cloud needs faster processing and better resource provisioning. MapReduce is the framework for computing large scale datasets in cloud environments. Optimization of resource requirement for each job to satisfy a specific objective in MapReduce is an open problem. Many factors, e.g., system side information and requirements of each client must be considered to estimate the appropriate amount of resources. This paper presents a mathematical model for the optimal number of map tasks in MapReduce resource provisioning. This model is to estimate the optimal number of the mappers based on the resource specification and the size of the dataset.

**Keywords:** Big data · Cloud computing · Resource provisioning

## 1 Introduction

Large amount of the datasets are continuously produced from the web, scientific computing, social media, and IoT application. When input data is too large to handle by a single computational resource, the computation needs to be distributed across a massive number of machines to finish the job in the given time.

MapReduce library partitions the input data into a number of inputsplits. A map task reads an inputsplit and, the user-defined map function processes the inputsplit. The map function takes input key/value pairs and produces a set of intermediate key/value pairs. The intermediate key/value pairs are buffered in the memory of the mapper. When the amount of the data size reaches a threshold of the memory buffer, intermediate key/value pairs are written to the local disk and partitioned by hash function for reduce tasks. Reduce tasks read and sort intermediate data so that the data with the same key are grouped together. The key and the set of intermediate values are passed to the reduce function. The output data from the reduce function is appended to the final output file [1].


D. Da Silva et al. (Eds.): CLOUD 2019, LNCS 11513, pp. 240–252, 2019.
https://doi.org/10.1007/978-3-030-23502-4_17

Due to the flexibility, scalability, simplicity in scheduling and fault tolerance, MapReduce is increasingly used for large scale data processing such as personalized advertising and the other efficient data mining tasks. MapReduce programs are run in public cloud or private cloud. However, running the MapReduce program on the public cloud has become a realistic option for most users. In the public cloud environment, virtualization technique provides services for users to provision a virtual cluster or to release the cluster in a specified time. Users are responsible for determining the appropriate number of virtual machines to execute specific MapReduce tasks.

Optimal configuration for the map and reduce tasks may differ based on the type of application and the amount of input data. Running a MapReduce program in the cloud requires optimization for resource usage to minimize the cost or job finish time. In many research approaches, resource provisioning in MapReduce based on regression analysis such as profiling or sampling a number of parameters with test runs on small scale Hadoop clusters using sample datasets [2–4].

MapReduce performance models are designed to estimate job completion time based on job profile, size of input data and specification of resources [3,4]. The job profile includes the application characteristics during all map and reduce phases of a MapReduce job. Any modifications of the MapReduce program or the underlying Hadoop framework are not necessary for profiling technique. However, test runs are necessary to estimate the number of resources using job profiles.

In MapReduce cloud services such as Amazon Web Services, customers can create MapReduce clusters to analyze large datasets by specifying required resources and submit MapReduce jobs and cloud service provider invokes virtual machines (VMs) to execute the jobs and the VMs are revoked by cloud service providers or released by customers after job completion. The cost of cloud service usage and performance of the specific job depends on the parameters chosen by users, such as the type of virtual machines, the number of virtual machines and the number of mappers per VM. Cloud service models can be generally classified as three approaches. In the first approach, the customer specifies the resource requirement for each job and the cloud service provider simply allocates the requested resources upon job arrival time [19,20]. Customer manages each job for the efficient resource usage and, therefore it is lack of global resource optimization across jobs. In the second approach, customers specify the required resources for each job and cloud provider schedules the jobs [5]. Optimization is partly managed by the customer and partly managed by the cloud service provider. The cloud provider performs optimized resource management to finish the jobs in the specified time to meet the service requirements for the customers. Therefore, the opportunity for delay start is not provided for the jobs of the customer and each job is necessary to start immediately. Inefficient allocation of the resources to a job can result in higher cost for the cloud service provider. In the third approach, customers only require to submit the jobs and specify job completion time and, cloud service provider manages the resource requirements and allocation of resources [4].

However, all conventional resource provisioning models are on profiling technique, test runs and sampling. Such approaches are not realistic to apply for MapReduce resource provisioning system in practical use. Automatic provisioning approach can help solve the problems to reduce both customer's burden and cloud provider's burden. According to the best of our knowledge, our mathematical model is the first approach for the estimation of computational resources in the cloud environment.

In this paper, we propose a mathematical model to estimate the optimal number of mappers based on the resource specification and the size of datasets. This model can be used for both customer-managed and cloud-managed environments. The mathematical model is derived to find the number of map tasks from the phases of mapper to optimize resource provisioning in MapReduce. This paper is organized as follows. Section 2 presents our MapReduce performance model. The preliminary evaluation is shown in Sect. 3. Related works are described in Sect. 4. Section 5 summarizes the paper and describes future work.

## 2 MapReduce Performance Model

In this resource provisioning approach, computational nodes are assigned as mappers and reducers. Input data is distributed across mapper nodes in Hadoop File System (HDFS) and partitioned into equal-sized inputsplits. Mappers read inputsplits from HDFS, create key/value pairs of input records and process user-defined map function. Output data is partitioned and formatted by partitioning and serialization utilities of MapReduce. Output partitions are stored in the memory buffer of mappers. The buffer consists of two parts: accounting part that stores 16 bytes of metadata per record and serialization part that stores serialized map output records. When either of two parts fills up to threshold, the spill process begins to write data records to disk. If more than one spill file is created, all spill files are merged into a single output file. Thus, the total processing time for the map task includes time taken for read phase, map phase, collect phase, spill phase and merge phase. In a merged file on a mapper, several partitions are generated and evenly distributed to all reducers by using the hash function. A partition corresponds to the set of key/value, where all keys are the same. Reducer merges all partitions from mappers and writes to shuffle files. One shuffle file handles only one key. All shuffle files are merged into one input file for the reduce function. The reducer processes the input file and then writes the result on HDFS. Total processing time at reducer includes time taken for shuffle phase, sort phase, reduce phase and write phase. This paper presents the details of procedures at a mapper and estimation of the optimal number of mappers. Figure 1 shows the procedures of mapper and the notations for parameters of the equations are shown in Table 1.

**Table 1.** Notations

| |
|---|
| I, input data |
| N, number of inputsplit |
| $I_i$, $i$-th inputsplit |
| $\|I_i\|$, size of $i$-th inputsplit |
| $C_I$, constant for $\|I_i\|$ |
| M, the set of mappers |
| m, number of mappers |
| $M_k$ , $k$-th mapper |
| $\alpha_k^p$, processing speed of $k$-th mapper |
| $\alpha_k^r$, I/O read speed of $k$-th mapper |
| $\alpha_k^w$, I/O write speed of $k$-th mapper |
| $R_{rec}^{in}(i)$, number of input records in $i$-th inputsplit |
| $w_{i,k}^{in}$, workload to process the $k$-th input record in $I_i$ |
| $\overline{w}_{split}$, workload to generate an inputsplit |
| $T_{gen}(i,k)$, time to generate $N$ inputsplits |
| $T_{map}(i,k)$ , processing time of $i$-th inputsplit on $k$-th mapper $M_k$ |
| $w_{i,j}^{in}$, workload of $j$-th input record in $i$-th inputsplit |
| $\overline{w}_i^{in}$, mean workload for each input record in $I_i$ |
| $R_{rec}^{out}(i)$, number of output records in $I_i$ |
| $\rho_{rec}(i)$, the ratio of the number of records between $R_{rec}^{in}(i)$ and $R_{rec}^{out}(i)$ |
| $W_{in}$, width of each input record |
| $W_{out}$, width of each output record |
| $\overline{w}_i^{part}$, workload of partitioning the $k$-th output record from $I_i$ |
| $\overline{w}_i^{ser}$, workload of serializing the $k$-th output record |
| $T_{ser}(i,k)$, serialization time of $k$-th output record |
| $T_{collect}(i,k)$, total time of collect phase |
| $R_{max}^{ser}(i)$, number of maximum serialization records |
| pSortMB, io.sort.mb |
| pSortRatio, Sorting ratio |
| pSpillRatio, Spill ratio |
| $\|O_i\|$, output size for $I_i$ |
| $R_{max}^{acc}(i)$, maximum number of accounting records |
| $R_{spillb}(i)$, number of spill buffer records |
| $N_{spill}(i)$, number of spills |
| $\|B_{spill}\|$, spill buffer size |
| $\|F_{spill}(i)\|$, spill file size |
| $T_{spill}(i,k)$, total time for spill phase |
| $N_{red}$, number of reducers |
| $\overline{w}_{sort}$, workload to sort a record |
| $\overline{w}_{merge}$, workload to merge a record |
| $N_{max}^{spill}(i)$, maximum number of spill |
| $N_{merge}$, number of merges |
| $T_{merge}(i,k)$, merge time of of $i$-th inputsplit on $k$-th mapper $M_k$ |
| $T_{map}^{total}(i,k)$ , total processing time of $i$-th inputsplit on $k$-th mapper $M_k$ |
| $N_{opt}$, optimal number of inputsplits |
| $\alpha_{ave}^*$, average performance |

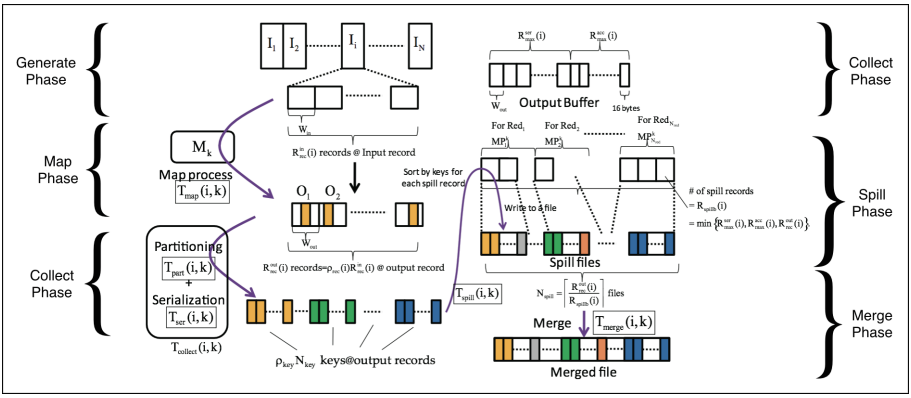


**Fig. 1.** Procedures at a mapper

### 2.1 Partitioning Input Data and Execution at Mappers

The input data is defined as $I = \{I_1, I_2, \ldots, I_N\}$ where $I_i$ is the $i$-th inputsplit, $|I_i|$ is the size of $I_i$, and $N$ is the number of inputsplits. In MapReduce, $I_i$ is located on a local disk of a mapper in Hadoop File System (HDFS). The set of mappers is defined as $M = \{M_1, M_2, \ldots, M_m\}$. $\alpha_k^p$, $\alpha_k^r$, and $\alpha_k^w$ are the processing speed, I/O read speed, and I/O write speed of mapper $k$, respectively. Let the number of input records in $I_i$ be $R_{rec}^{in}(i)$, and the workload to process the $j$-th input record in $I_i$ be $w_{i,j}^{in}$. Since input data is divided into equal-sized inputsplits, $|I_i| = C_I, \forall I_i \in I$ for a constant $C_I$. If the workload to generate an inputsplit is defined as $\overline{w}_{split}$, the time to prepare $N$ inputsplits is defined as

$$T_{gen}(i,k) = \frac{\overline{w}_{split}}{\alpha_k^p}\left(\frac{|I|}{C_I} - 1\right) = \frac{\overline{w}_{split}}{\alpha_k^p}(N-1) \tag{1}$$

where $|I|$ is the size of input $I$.

The sum of the time to transfer inputsplit $I_i$, to read $I_i$, and to process the map task on mapper $M_k$ is defined as

$$\begin{aligned} T_{map}(i,k) &= \frac{C_I}{\alpha_k^r} + \frac{1}{\alpha_k^p}\sum_{j=1}^{R_{rec}^{in}(i)} w_{i,j}^{in} \\ &= \frac{C_I}{\alpha_k^r} + \frac{\overline{w}_i^{in} R_{rec}^{in}(i)}{\alpha_k^p}, \end{aligned} \tag{2}$$

where $\overline{w}_i^{in}$ is the mean workload to process each input record in inputsplit $I_i$.

### 2.2 Collect Phase

The number of output records for $I_i$ from mapper $M_k$ is defined as

$$R_{rec}^{out}(i) = R_{rec}^{in}(i) \times \rho_{rec}(i), \tag{3}$$

where $\rho_{rec}(i)$ is the ratio of the number of output records to input records. If the size of each input record is defined as $W_{in}$, the number of input records is $R^{in}_{rec}(i) = \frac{C_I}{W_{in}}$. Thus, (3) can be rewritten as

$$R^{out}_{rec}(i) = \frac{C_I \rho_{rec}(i)}{W_{in}}. \tag{4}$$

Let the workload to partition the output generated from $I_i$ be $w^{part}_i$ and their average be $\overline{w}^{part}_i$. Then the time of partitioning the output of $I_i$ assigned to mapper $M_k$ is defined as follows:

$$T_{part}(i,k) = \frac{1}{\alpha^p_k} \sum_{j=1}^{\frac{C_I \rho_{rec}(i)}{W_{in}}} w^{part}_{i,j} = \frac{\overline{w}^{part}_i}{\alpha^p_k} \frac{C_I \rho_{rec}(i)}{W_{in}}. \tag{5}$$

As for the serialization time, let $w^{ser}_i$ be the workload to serialize the output from $I_i$ assigned to mapper $M_k$ and the mean workload to serialize the output records be $\overline{w}^{ser}_i$. Then the time for serialization is defined as follows:

$$T_{ser}(i,k) = \frac{1}{\alpha^p_k} \sum_{j=1}^{\frac{C_I \rho_{rec}(i)}{W_{in}}} w^{ser}_{i,j} = \frac{\overline{w}^{ser}_i}{\alpha^p_k} \frac{C_I \rho_{rec}(i)}{W_{in}}. \tag{6}$$

Thus, the total processing time in this collect phase can be derived by Eq. (5) + Eq. (6).

$$\begin{aligned} T_{collect}(i,k) &= \frac{\overline{w}^{part}_i}{\alpha^p_k} \frac{C_I \rho_{rec}(i)}{W_{in}} + \frac{\overline{w}^{ser}_i}{\alpha^p_k} \frac{C_I \rho_{rec}(i)}{W_{in}} \\ &= \frac{C_I \rho_{rec}(i)}{\alpha^p_k W_{in}} \left( \overline{w}^{part}_i + \overline{w}^{ser}_i \right). \end{aligned} \tag{7}$$

### 2.3 Spill Phase

In this phase, each output key/value pair is sorted and written to "spill" files. Spill phase starts when the output data (output records and accounting part) in the memory buffer exceeds the threshold value (e.g., 0.8 × output buffer size). The size of each output records is $W_{out} = \frac{|O_i|}{R^{out}_{rec}(i)} = \frac{|O_i|}{R^{in}_{rec}(i)\rho_{rec}(i)}$, where $|O_i|$ is the output size for $I_i$.

The output buffer is divided into two parts: serialization buffer and accounting buffer. Each buffer stores its own serialization records and accounting records. The number of maximum serialization records in the output buffer can be defined as

$$\begin{aligned} R^{ser}_{max}(i) &= \frac{pSortMB \times 2^{20} \times (1 - pSortRatio) \times pSpillRatio}{W_{out}} \\ &= \frac{pSortMB \times 2^{20} \times (1 - pSortRatio) \times pSpillRatio}{|O_i| / \{R^{in}_{rec}(i)\rho_{rec}(i)\}}, \end{aligned} \tag{8}$$

where $pSortMB$ is "io.sort.mb" and therefore $psortMB$ is multiplied by $2^{20}$.

16-byte data is written as metadata for each accounting record. The number of accounting records is defined as

$$R_{max}^{acc}(i) = \frac{pSortMB \times 2^{20} \times pSortRatio \times pSpillRatio}{16} \tag{9}$$

From (4), (8), and (9), the number of spill buffer records when spill is performed is defined as

$$R_{spillb}(i) = \min\{R_{max}^{ser}(i), R_{max}^{acc}(i), R_{rec}^{out}(i)\} \tag{10}$$

The number of spills is defined as

$$\begin{aligned} N_{spill}(i) &= \left\lceil \frac{R_{rec}^{out}(i)}{R_{spillb}(i)} \right\rceil \\ &\Leftrightarrow N_{spill} - 1 < \frac{R_{rec}^{out}(i)}{R_{spillb}(i)} \leq N_{spill} \end{aligned} \tag{11}$$

The spill buffer size is defined as follows:

$$\begin{aligned} |B_{spill}| &= R_{spillb}(i) \times W_{out}. \\ |F_{spill}(i)| &= |B_{spill}|. \end{aligned} \tag{12}$$

where $|F_{spill}(i)|$ is the spill file size. The total time for spill phase is defined as:

$$\begin{aligned} T_{spill}(i,k) &= N_{spill}(i) \times \left\{ R_{spillb}(i) \log\left(\frac{R_{spillb}(i)}{N_{red}}\right) \times \frac{\overline{w}_{sort}}{\alpha_k^p} + \frac{|F_{spill}(i)|}{\alpha_k^w} \right\} \\ &= \left\lceil \frac{R_{rec}^{out}(i)}{R_{spillb}(i)} \right\rceil \left\{ R_{spillb}(i) \log\left(\frac{R_{spillb}(i)}{N_{red}}\right) \times \frac{\overline{w}_{sort}}{\alpha_k^p} + \frac{R_{spillb}(i) W_{out}}{\alpha_k^w} \right\} \\ &= R_{rec}^{out}(i) \left\{ \log\left(\frac{R_{spillb}(i)}{N_{red}}\right) \times \frac{\overline{w}_{sort}}{\alpha_k^p} + \frac{W_{out}}{\alpha_k^w} \right\} \\ &= R_{rec}^{out}(i) \left\{ \log\left(\frac{R_{spillb}(i)}{N_{red}}\right) \times \frac{\overline{w}_{sort}}{\alpha_k^p} + \frac{W_{out}}{\alpha_k^w} \right\}, \end{aligned} \tag{13}$$

where $\overline{w}_{sort}$ is the processing time to sort a record and $T_{spill}(i,k)$ is the total time to spill the records which exceed the threshold value. Thus, the time complexity is $O\left(R_{spillb}(i) \log\left(\frac{R_{spillb}(i)}{N_{red}}\right)\right)$, where $\log\left(\frac{R_{spillb}(i)}{N_{red}}\right)$ steps for the merging are needed for generating partitions in total. By applying (4) to (13),

$$T_{spill}(i,k) = \frac{|I_i|\rho_{rec}(i)}{W_{in}} \left\{ \log\left(\frac{R_{spillb}(i)}{N_{red}}\right) \times \frac{\overline{w}_{sort}}{\alpha_k^p} + \frac{W_{out}}{\alpha_k^w} \right\}. \tag{14}$$

### 2.4 Merge Phase

This phase is performed after the spill phase finishes. Both spill phase and merge phase are optional. The objective of the merge phase is to generate one output

file from several spill files. The number of merges depends on the parameter of "io.sort.factor" which is the number of spill files to generate a new file. Thus, generally, the number of merges is

$$\begin{aligned} N_{merge}(i) &= \left\lfloor \frac{N_{spill}(i)}{N_{max}^{spill}(i)} \right\rfloor (N_{max}^{spill} - 1) + N_{spill}(i) mod(N_{max}^{spill}(i)) - 1 + \left\lfloor \frac{N_{spill}(i)}{N_{max}^{spill}(i)} \right\rfloor \\ &= \left\lfloor \frac{N_{spill}(i)}{N_{max}^{spill}(i)} \right\rfloor N_{max}^{spill} + \left( N_{spill}(i) - \left\lfloor \frac{N_{spill}(i)}{N_{max}^{spill}(i)} \right\rfloor N_{max}^{spill}(i) - 1 \right) \\ &= N_{spill}(i) - 1. \end{aligned} \tag{15}$$

The file read time for each spill file, the file write time for each spill file, and the number of records for merging into one records affect the time of the merge phase. By using (15), the time for merging is defined as

$$T_{merge}(i,k) = N_{merge}(i) \left( \frac{|F_{spill}(i)|}{\alpha_k^r} + \frac{|F_{spill}(i)|}{\alpha_k^w} + \frac{R_{spillb}(i)\rho_{rec}\overline{w}_{merge}}{\alpha_k^p} \right) \tag{16}$$

where $R_{spillb}(i)\rho_{rec}$ is the number of records in a spill file, and we assume that each single spill file is read and then written down to the merged file in one by one basis. By using (12), (16) is rewritten as follows:

$$\begin{aligned} T_{merge}(i,k) &= (N_{spill}(i) - 1) \left\{ R_{spillb}(i) W_{out} \left( \frac{1}{\alpha_k^r} + \frac{1}{\alpha_k^w} \right) + \frac{R_{spillb}(i)\rho_{rec}\overline{w}_{merge}}{\alpha_k^p} \right\} \\ &= (N_{spill}(i) - 1) \left\{ R_{spillb}(i) \left( W_{out} \left( \frac{1}{\alpha_k^r} + \frac{1}{\alpha_k^w} \right) + \frac{\rho_{rec}\overline{w}_{merge}}{\alpha_k^p} \right) \right\}. \end{aligned} \tag{17}$$

### 2.5 Total Processing Time at a Mapper

Let the total processing time at a mapper $M_k$ to process $I_i$ be $T_{map}^{total}(i,k)$. Thus, we have

$$T_{map}^{total}(i,k) = T_{gen}(i,k) + T_{map}(i,k) + T_{collect}(i,k) + T_{spill}(i,k) + T_{merge}(i,k). \tag{18}$$

By developing (18), we have

$$\begin{aligned} T_{map}^{total}(i,k) = {} & \frac{\overline{w}_{split}}{\alpha_k^p} \left( \frac{|I|}{C_I} - 1 \right) + \frac{C_I}{\alpha_k^r} + \frac{\overline{w}_i^{in} R_{rec}^{in}(i)}{\alpha_k^p} \\ & + \frac{C_I \rho_{rec}(i)}{\alpha_k^p W_{in}} \left( \overline{w}_i^{part} + \overline{w}_i^{ser} \right) \\ & + \frac{C_I \rho_{rec}(i)}{W_{in}} \left\{ \log \left( \frac{R_{spillb}(i)}{N_{red}} \right) \times \frac{\overline{w}_{sort}}{\alpha_k^p} + \frac{W_{out}}{\alpha_k^w} \right\} \\ & + (N_{spill}(i) - 1) \left\{ R_{spillb}(i) \left( W_{out} \left( \frac{1}{\alpha_k^r} + \frac{1}{\alpha_k^w} \right) + \frac{\rho_{rec}\overline{w}_{merge}}{\alpha_k^p} \right) \right\}. \end{aligned} \tag{19}$$

At (19), if the system is homogeneous, total processing time at the mapper can be rewritten as

$$\begin{aligned}
T_{map}^{total}(N,k) &= \frac{\overline{w}_{split}}{\alpha_{ave}^{p}}N + \frac{|I|}{N}\left\{\frac{N}{\alpha_{ave}^{r}} + \frac{\rho_{rec}(i)}{\alpha_{ave}^{p}W_{in}}\left(\overline{w}_{i}^{part} + \overline{w}_{i}^{ser}\right)\right\} \\
&+ \frac{|I|}{N}\frac{\rho_{rec}(i)}{W_{in}}\left\{\frac{\overline{w}_{sort}}{\alpha_{ave}^{p}}\log\left(\frac{R_{spillb}(i)}{N_{red}}\right) + \frac{W_{out}}{\alpha_{ave}^{w}}\right\} \\
&+ \frac{|I|\rho_{rec}(i)}{NW_{in}}\left\{W_{out}\left(\frac{1}{\alpha_{ave}^{r}} + \frac{1}{\alpha_{ave}^{w}}\right) + \frac{\rho_{rec}\overline{w}_{merge}}{\alpha_{ave}^{p}}\right\} \\
&- R_{spillb}(i)\left\{W_{out}\left(\frac{1}{\alpha_{ave}^{r}} + \frac{1}{\alpha_{ave}^{w}}\right) + \frac{\rho_{rec}\overline{w}_{merge}}{\alpha_{ave}^{p}}\right\} - \frac{\overline{w}_{split}}{\alpha_{ave}^{p}} \\
&= \frac{\overline{w}_{split}}{\alpha_{ave}^{p}}N + \frac{|I|}{N}\left\{\frac{1}{\alpha_{ave}^{r}} + \frac{\rho_{rec}(i)}{\alpha_{ave}^{p}W_{in}}\left(\overline{w}_{i}^{part} + \overline{w}_{i}^{ser}\right)\right\} \\
&+ \frac{|I|}{N}\frac{\rho_{rec}(i)}{W_{in}}\left\{\frac{\overline{w}_{sort}}{\alpha_{ave}^{p}}\log\left(\frac{R_{spillb}(i)}{N_{red}}\right) + \frac{W_{out}}{\alpha_{ave}^{w}}\right. \\
&\left. + W_{out}\left(\frac{1}{\alpha_{ave}^{r}} + \frac{1}{\alpha_{ave}^{w}}\right) + \frac{\rho_{rec}\overline{w}_{merge}}{\alpha_{ave}^{p}}\right\} \\
&- R_{spillb}(i)\left\{W_{out}\left(\frac{1}{\alpha_{ave}^{r}} + \frac{1}{\alpha_{ave}^{w}}\right) + \frac{\rho_{rec}\overline{w}_{merge}}{\alpha_{ave}^{p}}\right\} - \frac{\overline{w}_{split}}{\alpha_{ave}^{p}} \qquad (20)
\end{aligned}$$

where $\alpha_{ave}^{*}$ is the average performance for reading time, processing time and writing time of mappers.

If we define the optimal number of inputsplits as $N_{opt}$, it is obtained by differentiating equation (20) with respect to $N$. $N_{opt}$ is the value of $N$ when we have $\frac{dT_{map}^{total}(N,k)}{dN} = 0$. $N_{opt}$ is derived as

$$N_{opt} = \left[\frac{\alpha_{ave}^{p}|I|}{\overline{w}_{split}}\left\{\frac{1}{\alpha_{ave}^{r}} + \frac{\rho_{rec}(i)}{\alpha_{ave}^{p}W_{in}}\left(\overline{w}_{i}^{part} + \overline{w}_{i}^{ser}\right) + \frac{\rho_{rec}(i)}{W_{in}}A\right\}\right]^{\frac{1}{2}}, \qquad (21)$$

where

$$A = \frac{\overline{w}_{sort}}{\alpha_{ave}^{p}}\log\left(\frac{R_{spillb}(i)}{N_{red}}\right) + \frac{W_{out}}{\alpha_{ave}^{w}} + W_{out}\left(\frac{1}{\alpha_{ave}^{r}} + \frac{1}{\alpha_{ave}^{w}}\right) + \frac{\rho_{rec}\overline{w}_{merge}}{\alpha_{ave}^{p}}. \qquad (22)$$

## 3 Preliminary Evaluation

In our MapReduce experiment, a program that estimates the value of Pi ($\pi$) using quasi-Monte Carlo method was tested to compare the performance with original approach. Hadoop MapReduce 3.0 is configured for 8 nodes cluster in cloud environment created by CloudStack. The $\pi$ sample uses a statistical method to estimate the value of $\pi$. Random points are placed in a unit square which contains a circle. The area of the unit square is 1 and the probability of the points that fall within the circle are equal to the area of the circle, $\pi/4$. The value of $\pi$ can be estimated by the value of $4R$ in which $R$ is the ratio of the number of

points inside the circle to the total number of points within the square. Mapper generates points in a unit square and counts the points inside and outside of the inscribed circle of the square. Reducer accumulates points inside and outside of the circle from the output of the mapper. If the sample of points is large, the estimation is better. The optimal number of map tasks is obtained after running the user program to estimate the value of $\pi$ by varying the number of map tasks from 1 map task to 32 map tasks using $100 \times 10^3$ samples to $10 \times 10^6$ samples. The MapReduce program was tested on small scale cluster of 8 virtual machines with QEMU Virtual CPU built on Intel Xeon (R) CPU E5-1660 v4 @ 3.20 GHzx15 processor with 64 GB memory. The processing time for map tasks of the program is shown in Fig. 2.

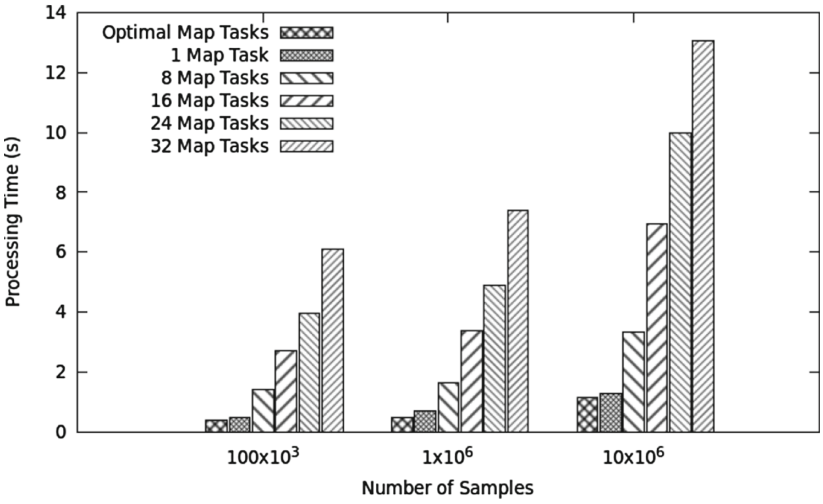


**Fig. 2.** Optimal processing time at the mappers

| Optimal Number of Map Tasks with Various Samples | | |
|---|---|---|
| $10x10^3$ samples | $1x10^6$ samples | $10x10^6$ samples |
| 2 | 2 | 3 |

**Fig. 3.** Optimal number of map tasks

The program is tested for both single node Hadoop cluster setup and multinodes Hadoop cluster setup by varying input data size from $100 \times 10^3$ samples, $1 \times 10^6$ samples to $10 \times 10^6$ samples. Figure 3 shows the experimental result for the optimal number of map tasks when the various number of samples are processed and optimal number of map tasks is automatically figured out by the program. For resource provisioning, customers can choose the optimal number of map tasks to process $100 \times 10^3$ samples to $10 \times 10^6$ to estimate the value of $\pi$. The processing time of the optimal map tasks are significantly faster than that of the chosen number of map tasks in existing approach as shown in Fig. 2.

## 4 Related Works

The existing researches on MapReduce focused on improving the performance of MapReduce in a Hadoop cluster based on profiling and sampling [6–10,12,16] and collecting properties of codes and data on the execution of jobs [11]. Profiling approaches use job profiles to optimize MapReduce resource provisioning. A framework for the resource provisioning and failures estimation in Hadoop cluster by profiling characteristics of MapReduce jobs was introduced in [3]. MapReduce profiling techniques were developed to optimize the resource provisioning and minimize the cost of computing in the cloud [3,12–15]. Automatic resource prediction tool based on job profiling and estimation models was developed to provision the best cluster size to meet the requirements of jobs [17]. CRESP [2] developed a MapReduce resource provisioning method by analyzing the cost for map and reduce tasks to find optimal setting for resources. Bazaar [18] was developed to predict the job performance in data centers using gray box approach with MapReduce resource provisioning as an example of the data analytic. Cura [4] addressed the global resource optimization and scheduling for the cloud provider to minimize the customer costs in MapReduce jobs. In existing researches, profiling and sampling approaches were used to estimate the optimal number of resources with test runs. Profiling and sampling can incur overheads and increase processing time depending on the time for generating job profiles, the characteristics of data and resource specification. In our research, a mathematical model is derived to estimate the number of map tasks for both customer-managed and cloud-managed environment without profiling or sampling. Our research is to optimize resource provisioning in Hadoop MapReduce for both customers and cloud service providers in private and public cloud environments.

## 5 Conclusions and Future Work

This paper presents a mathematical MapReduce performance model for applying in both cloud-managed environment and customer-managed environment. In contrast to existing services, this approach can estimate the number of map tasks for resource provisioning in big data computing without test runs. Profiling or sampling is not necessary to find the optimal number of map tasks in

this research. The required number of map tasks can be calculated based on the specification of resources and the size of input data. This research can improve performance of Hadoop MapReduce environment for cloud service provider and users. This research paper is on the mapper side to estimate the optimal number of map tasks. Future work and the extension of this research paper are on both map and reduce sides to estimate optimal number of mappers and reducers for the resource provisioning using MapReduce benchmarks with overall evaluation.

## References


1. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**, 107–113 (2008)
2. Chen, K., Powers, J., Guo, S., Tian, F.: CRESP towards optimal resource provisioning for MapReduce computing in public clouds. IEEE Trans. Parallel Distrib. Syst. **25**(6), 1403–1412 (2014)
3. Verma, A., Cherkasova, L., Campbell, R.H.: Resource provisioning framework for MapReduce jobs with performance goals. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 165–186. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25821-3_9
4. Palanisamy, B., Singh, A., Liu, L.: Cost-effective resource provisioning for MapReduce in a cloud. IEEE Trans. Parallel Distrib. Syst. **26**(5), 1265–1279 (2015)
5. Sotomayor, B., Keahey, K., Foster, I.: Combining batch execution and leasing using virtual machines. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing, pp. 87–96 (2008)
6. Jiang, D., Ooi, B.C., Shi, L., Wu, S.: The performance of MapReduce: an in-depth study. Proc. VLDB Endow. **3**(1–2), 472–483 (2010)
7. Babu, S., et al.: Towards automatic optimization of MapReduce programs. In: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 137–142 (2010)
8. Herodotou, H., Babu, S.: Profiling, what-if analysis, cost-based optimization of MapReduce programs. Proc. VLDB Endow. **4**(11), 1111–1122 (2011)
9. Wang, G., et al.: A simulation approach to evaluating design decisions in MapReduce setups. In: Proceedings of the IEEE/ACM International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems, pp. 1–11 (2009)
10. Herodotou, H.: Hadoop Performance Models, Technical teport, CS-2011-05 (2011)
11. Agarwal, S., Kandula, S., Bruno, N., Wu, M.-C., Stoica, I., Zhou, J.: Re-optimizing data-parallel computing. In: Proceedings of the 9th USENIX Conference on NSDI, p. 21 (2012)
12. Kambatla, K., Pathak, A., Pucha, H.: Towards optimizing Hadoop provisioning in the cloud. In: Proceedings of the Conference on Hot Topics in Cloud Computing, pp. 156–172 (2009)
13. Morton, K., Friesen, A., Balazinska, M., Grossman, D.: Estimating the progress of MapReduce pipelines. In: Proceedings of the IEEE 26th International Conference on Data Engineering, pp. 681–684 (2010)
14. Tian, F., Chen, K.: Towards optimal resource provisioning for running MapReduce programs in public clouds. In: Proceedings of the IEEE 4th International Conference on Cloud Computing, pp. 155–162 (2011)

15. Popescu, A., Ercegovac, V., Balmin, A., Branco, M., Ailamaki, A.: Same queries, different data: can we predict runtime performance? In: Proceedings of the 3rd International Workshop on Self-Managing Database Systems, pp. 275–280 (2012)
16. Herodotou, H., et al.: Starfish: a self-tuning system for big data analytics. In: CIDR 2011, pp. 261–272 (2011)
17. Herodotou, H., Dong, F., Babu, S.: No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In: Proceedings of the 2nd ACM Symposium on Cloud Computing, pp. 1–14 (2011)
18. Jalaparti, V., Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.: Bazaar: enabling predictable performance in datacenters, Microsoft Res., Cambridge, U.K., Technical report MSR-TR-2012-38 (2012)
19. Amazon Elastic Compute Cloud (2018). https://aws.amazon.com/ec2/
20. Amazon Elastic MapReduce (2018). https://aws.amazon.com/emr/
21. Apache Hadoop (2018). http://hadoop.apache.org