IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 22, 2018, accepted December 25, 2018, date of publication January 8, 2019, date of current version January 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2891001

Time Estimation and Resource Minimization Scheme for Apache Spark and Hadoop Big Data Systems With Failures

JINBAE LEE[1,2], BOBAE KIM[1,3], AND JONG-MOON CHUNG[1], (Senior Member, IEEE)
[1]School of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea
[2]Samsung Electronics, Co., Ltd., Gyeonggi-do 16677, South Korea
[3]Republic of Korea Army, Seoul 04383, South Korea

Corresponding author: Jong-Moon Chung (jmc@yonsei.ac.kr)

This research was supported by the grant MOIS-DP-2015-10 through the Disaster and Safety Management Institute, which is funded by the Ministry of Interior and Safety of the Republic of Korea Government.

ABSTRACT Apache Spark and Hadoop are open source frameworks for big data processing, which have been adopted by many companies. In order to implement a reliable big data system that can satisfy processing target completion times, accurate resource provisioning and job execution time estimations are needed. In this paper, time estimation and resource minimization schemes for Spark and Hadoop systems are presented. The proposed models use the probability of failure in the estimations to more accurately formulate the characteristics of real big data operations. The experimental results show that the proposed Spark adaptive failure-compensation and Hadoop adaptive failure-compensation schemes improve the accuracy of resource provisions by considering failure events, which improves the scheduling success rate of big data processing tasks.

INDEX TERMS Big data, failure probability, Apache Spark, resilient distributed dataset (RDD), Apache Hadoop, MapReduce, cloud computing, job estimation, resource provisioning, performance optimization.

I. INTRODUCTION

Spark and Hadoop of the Apache Software Foundation are the most widely used open-source parallel distribution platforms for big data processing. As big data processing is used in various fields, studies of platforms for big data processing are actively being carried out. Spark uses resilient distributed datasets (RDDs) to handle distributed processing on multiple nodes based on a directed acyclic graph (DAG), in which RDDs are effective in overcoming failures in the process. Spark can create various types of RDDs (e.g., HadoopRDD, MapRDD, ReduceRDD, JsonRDD, PythonRDD, etc.) according to each data type by loading file contents from databases, such as, Hadoop Distributed File System (HDFS), HBase, MySQL, and CSV. Therefore, Spark can process data from various types of already built databases. Spark transforms the RDD data using distributed processing cores. All of the processes conducted in Spark are recorded in the lineage, which is used in the recovery of the RDD in the case of a failure. Due to the fact that Spark uses RDDs, in-memory, and flexible multi-core processing, a RDD can be recovered from failure more quickly compared to the recovery time consumed in Hadoop. Spark is also very efficient in streaming and conducting iterations of jobs. Because many existing databases are configured in HDFS, many Spark systems have been analyzing data in HDFSs using the Yet Another Resource Negotiator (YARN) resource management system.

Hadoop is a parallel distributed processing platform that has been widely used in the original Google search engines. Hadoop has been widely used due to its fault tolerance and scalability provided by MapReduce technology, and is still used by many companies, such as, Yahoo, Facebook, and several cloud computing service providers. Hadoop systems majorly consist of HDFS, MapReduce data processing functions, and the resource manager YARN.

In order to implement a big data analysis system that can satisfy performance target times, Spark and Hadoop systems need accurate resource provisioning and job execution time estimations. However, due to unexpected errors and failures, making an accurate estimation is very challenging, since one will need to consider various changes that may occur in the dataset processing jobs and tasks, and also make accurate time consumption predictions.

Although many studies have been carried out on Spark and Hadoop systems, analysis of the existing prediction models reveals that there are a couple of important yet unsolved issues. First, prediction models are mostly based on

environments where failures did not occur [3], [15], [16], [17], and second, even for the models that considered failures, a probability of failure occurrence (HP model) was not included into the resource provisioning process [3]. Because failures do occur in real world systems, when applying existing time prediction models in resource provisioning, target completion time violations frequently occur. In particular, large systems, such as, big data systems should consider failure times because they are more prone to experience failure events [8]–[14].

Therefore, in this paper, a Spark adaptive failure-compensation (SAF) time estimation method, Hadoop adaptive failure-compensation (HAF) time estimation method, and resource minimization schemes based on SAF and HAF are proposed to effectively deal with failures that occur in Spark and Hadoop systems.

This paper is organized as follows. The background and related work on Hadoop and Spark are summarized in section 2, the HAF time estimation method is presented in section 3, the SAF time estimation method descriptions are provided in section 4, the experimental environments and results are presented in section 5, followed by the conclusion in section 6.

II. BACKGROUND AND RELATED WORK

Reliability and speed enhancement of big data processes are becoming more important due to various new real-time and near real-time applications. Due to this reason, many models on Spark and Hadoop performance and resource provisioning have been proposed. In the following subsections, the Hadoop system model is introduced first followed by the Spark system model descriptions.

A. APACHE HADOOP SYSTEM MODEL

Apache Hadoop is a big data open source parallel distributed processing platform that was initially released in 2011. Apache Hadoop works on HDFS with multiple servers as shown in Fig. 1. HDFS can process big data without costly

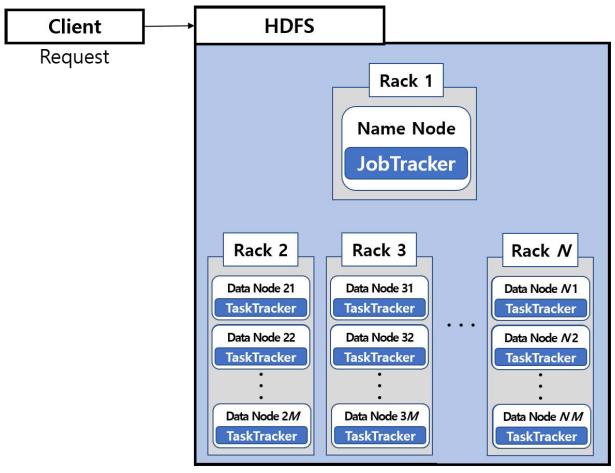


FIGURE 1. Apache Hadoop system architecture.

servers by performing parallel processing on multiple low cost servers. Hadoop is also robust against failures because it makes multiple copies of the data file and processes the data copies on multiple servers.

Apache Hadoop operates on HDFS, with one NameNode and one or more DataNodes in the Hadoop cluster. The NameNode is a node that manages the entire cluster, and the DataNodes are nodes that store data and process the jobs. The JobTracker in the NameNode manages job scheduling, and TaskTracker in the DataNodes processes distributed job tasks. In the case of Hadoop, previous research has been conducted to predict the execution time of a job in Hadoop. Subbulakshmi and Manjaly [22] and Hou *et al.* [23] show the performance improvement of Hadoop based on job scheduling using Hadoop YARN. However, this study did not attempt to optimize the performance using the developed mathematical model.

The process of Hadoop MapReduce follows the order of data loading, map phase, shuffle phase, and reduce phase. In [5], a mathematical Hadoop performance multi-step model was proposed. In addition, in [3], the HP model was proposed, which is based on a job execution time prediction method. In the HP model, the number of reduce tasks is fixed and the impact of the number of reduce tasks on the performance was not considered. In addition, in the first wave of the shuffle phase, the non-overlapping stage with the map phase is not mathematically expressed. Verma *et al.* [3] explain the impact of failure on the execution time, but the probability of failure occurrence and resource provisioning effects were not included. To deal with some of these issues, the improved HP (IHP) scheme proposed in [4] divides MapReduce into map, shuffle, and reduce phases and mathematically models the non-overlapping stage with the map phase in the first wave of the shuffle phase. Moreover, the IHP scheme uses locally weighted linear regression (LWLR) and shows a 95% accuracy under varying conditions in the number of tasks. However, since the model does not consider the occurrence of errors and failures, there is a difference in the performance when compared to a real HDFS system. In reality, Hadoop systems experience various types of failures due to user code mistakes, processor crashes, and equipment problems, which have different degrees of a degrading influence on the performance depending on the situation.

In [6], models that predict the execution time with five machine learning (ML) algorithms are proposed. However, a mathematical model is not provided and the performance analysis is based on the measurement of the total execution time. Since the total time has a wider variation range compared to the variations of the map task time and reduce task times, the scheme needs to create a new ML model even when slight changes are made. Yang *et al.* [7] measured the correlation between the total execution time and the input data size, file size, number of map and reduce tasks, and apply principal component analysis (PCA) in an attempt to optimize the performance by adjusting the variables that have the greatest impact. The proposed scheme tries to find

the factor with the biggest influence and use support vector machine (SVM) algorithms to predict the execution time. But the multiple waves model does not consider the overlap of the first wave and second wave, nor the probability of failure.

B. APACHE SPARK SYSTEM MODEL

Apache Spark is a big data open source parallel distributed processing platform that was initially released in 2014. As shown in Fig. 2, Apache Spark distributes several virtual machines (VM) in the Spark Executor. Apache Spark can use a Spark Standalone scheduler, and also can work with various types of databases because it loads data by copying the necessary data in RDD format from an already created database (e.g., HDFS or SQL). Spark will commonly use a YARN cluster manager in case of accessing a HDFS. In each executor, RDDs will be used in a sequence of 'transformation' operations based on the execution of an 'action.' RDD transformations and actions are conducted in a distributed processing manner after being scheduled by the DAG scheduler.

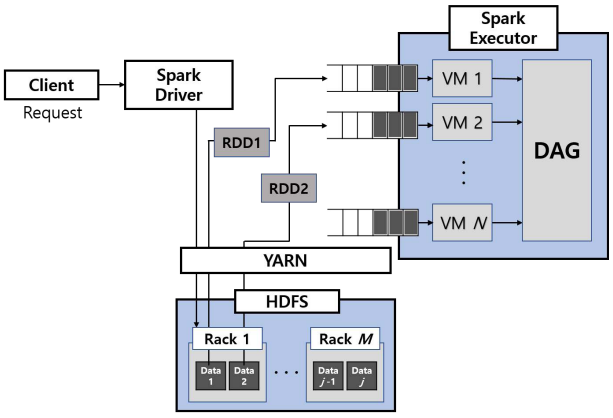


FIGURE 2. Apache Spark system architecture.

In the case of Spark, most research on Spark has been on predicting a job's execution time. Xia *et al.* [20] and Hu *et al.* [21] show the Spark performance improvement by job scheduling using Spark YARN or Spark Standalone. However, these papers do not conduct any optimization using the developed mathematical models.

Spark creates a RDD in the form of HadoopRDD from HDFS, then proceeds to MapReduce in the Spark executor. RDD is a distributed collection of data as elements. RDD is made up of multiple partitions, with data elements in them. Spark distributes each partition of the RDD. Spark processes the data while transforming the RDD to another RDD. In Hadoop's map, shuffle, and reduce steps, Spark performs map transformation of the HadoopRDD to generate a MappedRDD. Subsequent work will proceed using the ShuffledRDD and UnionRDD, which is similar to Hadoop MapReduce [19]. This work uses a different mathematical modeling technique compared to the methods used in Hadoop, as it is based on processing of RDDs, which is different to Hadoop processing procedures. In [15] a mathematical model of Spark is proposed, where the execution time of Spark is determined by a stage and partition unit. Sidhanta *et al.* [16] and Islam *et al.* [17] mathematically model the Spark execution time by proposing OptEx [16] and dSpark [17], which are resource allocation policies that find the lowest cost while satisfying the deadline. OptEx [16] and dSpark [17] show an accuracy in the range of 95-97%. However, these models do not take into account the influence of errors and failures, which can lead to a timeout that may result in deadline violations. Spark analysis needs to consider failures and errors and corresponding delays, because it is evident that servers, computers, and networking equipment show a growing probability of failure and errors as they age [16].

If the time prediction models in the resource scheduler cannot properly provision failures and errors, performance target time violations may occur when a processing failure or error occurs. In particular, in the case of a large-scale big data system, the size of the server becomes large, which can reach up to a warehouse-scale machine size formed through a large-scale cluster, and the probability of a processing failure or error event occurring increases [8]–[14]. Barroso *et al.* [11] noted that the annual failure rate of warehouse-scale machines is 4%. In [12] and [13], failures and resulting delays in large-scale data processing systems are discussed, and [9] shows that the annualized failure rates of the drives rise to 8% after two years. In addition, based on the MapReduce job failure rates reported in [8], disk drive failure rates reported by Google in [9], and server crash and disk drive failure rates reported in [10], it is evident that the failure and error rates are not negligible, thus dealing with failures is important. Therefore, the SAF and HAF schemes, proposed in this paper, attempt to overcome the lack of accuracy by modeling the influence of failures into the performance analysis of Spark and Hadoop systems, respectively. The modeling methods applied in the SAF and HAF scheme are presented below.

III. HADOOP ADAPTIVE FAILURE COMPENSATION METHOD

The proposed HAF scheme models the influence of failures into the IHP [4] model based on variables that express the influence of failure types. In the HAF scheme, MapReduce is divided into the map phase, shuffle phase, and reduce phase, same as in the IHP model. The parameters used in this paper are summarized in TABLE 1, where $N_{m.tf}$, $N_{sh.2.tf}$, and $N_{r.tf}$ were newly defined in this paper to model the failure probabilities that will be used to conduct optimal control in the HAF scheme.

A. JOB EXECUTION TIME ESTIMATION WITH FAILURES

The prediction time calculation differs depending on the stage at which the failure occurred, because the number of failed tasks that need to be reexecuted are different based on the operation stage. If a failure occurs during a map task in progress, it will be reexecuted only for the map task that has already been executed as well as the running map task in

TABLE 1. HADOOP parameters.

Variables	Expressions
$N_{m.1}$	Number of map tasks that complete in the first wave of the map phase
N_m^{slot}	Total number of map slots
N_r^{slot}	Total number of reduce slots
W	Total number of worker nodes
N_r	Total number of reduce tasks of a job
T_m	Average execution time of average and maximum time of a map task
t_f	Time of failure occurrence
$N_{m.done}$	Number of map tasks completed of a job
N_m	Total number of map tasks of a job
$N_{sh.1}$	Number of shuffle tasks that complete in the first wave of the shuffle phase of a job
$N_{sh.2}$	Number of shuffle tasks that complete in other waves of the shuffle phase of a job
$N_{m.2}$	Number of map tasks that complete in other waves of the map phase of a job
T_j^{total}	Total execution time of a job
$N_{m.fail.m}$	Number of map tasks completed on a failed node with a map failure
$N_{m.tf}$	Total number of map tasks included failed tasks
$N_{r.tf}$	Total number of reduce tasks included failed tasks
$T_{sh.1}$	Average execution time and maximum time of a shuffle task that completes in the first wave of the shuffle phase
$T_{sh.2}$	Average execution time and maximum time of a shuffle task that completes in other waves of the shuffle phase
T_r	Average execution time and maximum time of a reduce task
$N_{r.fail}$	Number of reduce tasks completed on a failed node with a reduce failure
$N_{r.done}$	Number of reduce tasks completed of a job
$N_{sh.2.tf}$	Number of shuffle tasks included failed tasks in other waves of the shuffle phase
$N_{m.fail.r}$	Number of map tasks completed on a failed node with a reduce failure

the failed worker node, but this does not affect the reduce tasks. However, when a failure occurs during a reduce task in progress, both the map tasks and reduce task on the failed node must be reexecuted.

1) CASE 1 : MAP TASK FAILURE ($t_f \le T_m + T_{sh.1}$)

The estimation time of the map phase is $(T_m N_{m.1})/N_m^{slot}$. From this, we can compute the number of map tasks completed ($N_{m.done}$) as $N_{m.done} = \lfloor (t_f N_m^{slot})/T_m \rfloor$. If there are W nodes, the number of failed tasks in the failed node is calculated as $N_{m.fail.m} = \lfloor N_{m.done}/W \rfloor$. The total execution time of the map phase, including the increased time due to failure, is equal to the time to execute the map tasks, which includes the failed map tasks ($N_{m.fail.m}$) and the currently processing map tasks ($N_{m.1}$). Thus, the total number of map tasks can be expressed as $N_{m.tf} = N_{m.1} + N_{m.fail.m}$. A failure at the map stage does not affect the reduce tasks, and therefore, $N_{r.tf} = N_r$.

2) CASE 2 : REDUCE TASK FAILURE

The expression of the job execution time is $T_j^{total} = (T_m N_{m.1})/N_m^{slot} + (T_{sh.1} N_{sh.1})/N_r^{slot} + (T_{sh.2} N_{sh.2})/N_r^{slot} + (T_r N_r)/N_r^{slot}$. From this, we can compute the number of reduce tasks completed ($N_{r.done}$) as follow: $(T_m N_{m.1})/N_m^{slot} + (T_{sh.1} N_{sh.1})/N_r^{slot} + (T_{sh.2} N_{sh.2})/N_r^{slot} + (T_r N_{r.done})/N_r^{slot} = t_f$. If there are W nodes, the number of failed tasks in the failed node is calculated as $N_{r.fail} = \lfloor N_{r.done}/W \rfloor$ and $N_{m.fail.r} = \lfloor N_m/W \rfloor$. The total execution time of each phase, including the increased time due to failure, is equal to the time to execute all involved tasks, which include the failed tasks and the initial configured tasks. Thus, the total number of tasks are expressed as $N_{m.tf} = N_{m.1} + N_{m.fail.r}$, $N_{sh.2.tf} = N_{sh.2} + N_{r.fail}$, and $N_{r.tf} = N_r + N_{r.fail}$. Therefore, the total execution time in the HAF model can be calculated as follows.

$$T_i^{total} = T_m \left\lceil \frac{N_{m.tf}}{N_m^{slot}} \right\rceil + T_{sh.1} \frac{N_{sh.1}}{N_r^{slot}} + T_{sh.2} \left\lceil \frac{N_{sh.2.tf}}{N_r^{slot}} \right\rceil + T_r \left\lceil \frac{N_{r.tf}}{N_r^{slot}} \right\rceil \quad (1)$$

B. TIME ESTIMATION WITH FAILURES PROBABILITY

The probability that a failure occurs in the map and the reduce phase is denoted by P_m and P_r, respectively. Then, the mean of $N_{m.tf}$, $N_{sh.2.tf}$, and $N_{r.tf}$ can be obtained as follows.

$$\begin{aligned} E(N_{m.tf}) &= N_{m.1}(1-P_m)(1-P_r) \\ &\quad + (N_{m.1} + N_{m.fail.m})P_m(1-P_r) \\ &\quad + (N_{m.1} + N_{m.fail.r})(1-P_m)P_r \\ &\quad + (N_{m.1} + N_{m.fail.m} + N_{m.fail.r})P_m P_r \quad (2) \\ E(N_{sh.2.tf}) &= N_{sh.2}(1-P_r) + (N_{sh.2} + N_{r.fail})P_r \quad (3) \\ E(N_{r.tf}) &= N_r(1-P_r) + (N_r + N_{r.fail})P_r \quad (4) \end{aligned}$$

Therefore, the total execution time based on (1) becomes

$$T_j^{total} = T_m \left\lceil \frac{E(N_{m.tf})}{N_m^{slot}} \right\rceil + T_{sh.1} \frac{N_{sh.1}}{N_r^{slot}} + T_{sh.2} \left\lceil \frac{E(N_{sh.2.tf})}{N_r^{slot}} \right\rceil + T_r \left\lceil \frac{E(N_{r.tf})}{N_r^{slot}} \right\rceil. \quad (5)$$

C. RESOURCE PROVISIONING

The deadline for the predicted job execution time is t, and in order to predict the optimal amount of resources, the first wave and the other waves (overlapped with the shuffle stage) of the map phase, the first wave and the other waves of the shuffle phase, and the reduce phase need to be considered in order to consider the influence of all slots. Then, (5) can be transformed into the following form of $a/m + b/(m+r) + c/r + d/r = t$ based on the transformation substitutions of $a = T_m E(N_{m.tf})$, $b = (T_m N_{m.2}) + (T_{sh.1} N_{sh.1})$, $c = T_{sh.2} E(N_{sh.2.tf})$, $d = T_r E(N_{r.tf})$, $m = N_m^{slot}$, $r = N_r^{slot}$, and $t = T_j^{total}$. Then the Lagrange multiplier method is used to conduct the resource optimization. Related equations are expressed as $f(m,r) = m + r$ and $g(m,r) = a/m + b/(m+r) + c/r + d/r - t$. The Lagrangian function is expressed as $L(m,r,\lambda) = f(m,r) + \lambda(m,r)$. The solutions satisfying the equations $\partial L/\partial m = 0$, $\partial L/\partial r = 0$, and $\partial L/\partial \lambda = 0$ are

$$m = \frac{\lambda}{(x+1)} \sqrt{a(x+1)^2 + y} \quad (6)$$

$$r = \frac{\lambda}{x(x+1)} \sqrt{a(x+1)^2 + y} \quad (7)$$

where the variables λ, x, and y are defined as $\lambda = 1/t\sqrt{(x+1)a+y}\lceil a(x+1)+bx+(c+d)(x(x+1))\rceil$, $x=\sqrt{a/(c+d)}$, and $y=ab/(c+d)$.

D. HAF SCHEME PROCESS

In the work process of HAF, (6) and (7) are used to find the optimal resource allocation size. The system periodically follows the processing of Algorithm 1.

Algorithm 1 HAF Scheme

```
if job request received then
    while job incomplete do
        if detect changes in error rate then
            UPDATE recalculate P_m,P_r
        end
        RECEIVE job execution request from the client
        CHECK job completion target time
        CHECK information in the job execution
              request message
        SET UP parameter values
              (T_m, T_sh.1, T_sh.2, T_r, P_m, P_r)
        LOAD parameters according to job type
        CALCULATE resource allocation size for
                  the map phase resource m
                  using (6)
        CALCULATE resource allocation size for
                  the map phase resource r
                  using (7)
        ALLOCATE resources according to optimal
                  resource values m and r
        EXECUTE job
    end
end
```

IV. SPARK ADAPTIVE FAILURES-COMPENSATION

The proposed SAF scheme models the influence of failures into the OptEx [16] model based on variables that express the influence of failure types. In the SAF scheme, Spark processing is divided into the Initialization phase, Preparation phase, Variable Sharing phase, and Computation phase as in the OptEx model [16]. The parameters used in this paper are summarized in TABLE 1, where the probability of failure in the data processing within a batch P_e and the probability of failure in data processing of the RDD process P_{RDD} was newly defined (i.e., presented in (9)) and applied to the optimal control process of the SAF scheme.

A. SPARK JOB EXECUTION TIME ESTIMATION WITH FAILURES

The difference between Hadoop and Spark is that Spark rebuilds the RDD at the point where the failure occurs at any stage, and then recalculates starting at the failed location. Therefore, no matter which type of error occurs at any stage, the total calculation time (T_{Est}) does not change, but instead, the failure influences the time to rebuild the RDD, which is why the variable sharing phase and the communication phase of the computation phase are added. If the RDD error probability of Spark is P_{RDD}, the execution time of Spark can be expressed as

$$T_{Est} = T_{Init} + T_{prep} + T_{vs} + T_{comp} + P_{RDD}(T_{vs} + T_{commn}) \quad (8)$$

where, T_{init} and T_{prep} are not affected by the input data size and iterations. In addition, other values are obtain by $T_{vs} = \varepsilon_{vs} T_{vs}^{baseline} in$ and $T_{comp} = T_{commn} + T_{exec} = (\varepsilon_{commn} T_{commn}^{baseline}/s_{baseline})s/n + (i\sum_{k=1}^{n_{unit}} M_a^k)/n$ as in the OptEx model [16], based on the parameter definitions in TABLE 2.

TABLE 2. SPARK parameters.

Variables	Expressions
i	Number of job iterations
n	Number of nodes
s	Input data set size
T_{Est}	Total calculation estimation time
T_{Init}	Time duration of the initialization phase
T_{Prep}	Time duration of the preparation phase
T_{vs}	Time duration of the variable sharing phase
ε_{vs}	Coefficient of the variable sharing phase
$T_{vs}^{baseline}$	Base value of the variable sharing phase time
T_{commn}	Time duration of the communication phase in the computation phase
ε_{commn}	Coefficient of the communication phase
$T_{commn}^{baseline}$	Base value of T_{commn}
T_{exec}	Time duration of the execution phase in the computation phase
M_a^k	Processing time of the kth RDD
n_{unit}	Number of RDD files
P_e	Probability of failure during the data processing
P_{RDD}	Probability of failure during the RDD process
T_{object}	Target job completion time duration

B. SPARK FAILURES PROBABILITY

Since the probability of error of Spark is not divided into map and reduce stages, but depends on the number of errors that occur in the processing of a data batch, the Spark RDD error probability P_{RDD} can be expressed as (9).

$$P_{RDD} = P_e + P_e^2 + P_e^3 + + P_e^i = \sum_{k=1}^{i} P_e^k \quad (9)$$

Therefore, the execution time of Spark can be expressed as (10).

$$\begin{aligned} T_{Est} &= T_{Init} + T_{prep} \\ &\quad + in\varepsilon_{vs} T_{vs}^{baseline} \\ &\quad + i\frac{1}{n}\sum_{k=1}^{n_{unit}} M_a^k \\ &\quad + \varepsilon_{commn} T_{commn}^{baseline} \frac{1}{S_{baseline}} \frac{s}{n} \\ &\quad + in\varepsilon_{vs} T_{vs}^{baseline} \sum_{k=1}^{i} P_e^k \\ &\quad + \varepsilon_{commn} T_{commn}^{baseline} \frac{1}{S_{baseline}} \frac{s}{n} i \sum_{k=1}^{i} P_e^k \end{aligned} \quad (10)$$

C. SPARK RESOURCE PROVISIONING

The optimal amount of required resources that need to be allocated to match Spark's job execution time T_{object} to the performance target time is derived in this subsection. For this purpose, using the performance target time upperbound T_{object}, (10) can be reformulated into (11), which can be reorganized into (12),

$$a + bn + \frac{c}{n} \leqq T_{object} \tag{11}$$

$$bn^2 + (a - T_{object})n + c \leqq 0 \tag{12}$$

based on the transformation substitutions of $a = T_{Init} + T_{Prep}$,$b = i\varepsilon_{vs}T_{vs}^{baseline} + i\varepsilon_{vs}T_{vs}^{baseline}\sum_{k=1}^{i} P_e^k$, and $c = i\sum_{k=1}^{n_{unit}} M_a^k + \varepsilon_{commn}T_{commn}^{baseline} s/s_{baseline} + i\varepsilon_{commn}T_{commn}^{baseline} s\sum_{k=1}^{i} P_e^k/s_{baseline}$. Based on the variable n in (12), the roots of the quadratic inequality are obtained as n_1 and n_2 below.

$$n_1 = \left\lceil \frac{(a - T_{object}) + \sqrt{(a - T_{object})^2 - 4bc}}{2bc} \right\rceil$$

$$n_2 = \left\lceil \frac{(a - T_{object}) - \sqrt{(a - T_{object})^2 - 4bc}}{2bc} \right\rceil$$

Since n represents the number of nodes, it has to be a positive integer. Therefore, if there is a negative number among n_1 and n_2, the value is excluded. Therefore the optimal n value (i.e., n^*) can be obtained from (13).

$$n^* = \{min(n_1, n_2) | n_1 > 0, n_2 > 0\} \tag{13}$$

D. SAF SCHEME PROCESS

In the work process of SAF, (13) is used to find the optimal resource allocation size. The system periodically follows the processing of Algorithm 2.

Algorithm 2 SAF Scheme

```
if job request received then
    while job incomplete do
        if detect change in error rate then
            UPDATE recalculate P_e,P_RDD
        end
        RECEIVE job execution request from the client
        CHECK job completion target time
        CHECK information in the job execution
              request message
        SET UP parameter values
              (T_Init, T_Prep, T_vs, T_exec, T_commn, P_e)
        LOAD parameters according to job type
        CALCULATE resource allocation size range
              [n_1, n_2] for the Spark resource n
              using (12)
        DETERMINE optimal n* using (13)
        ALLOCATE resources according to optimal
              resource values n*
        EXECUTE job
    end
end
```

V. EXPERIMENTAL ENVIRONMENT AND RESULTS

In this section, the setup of the simulation experiments and the performance results are provided. Considering the system job failure rates reported in [8], large disk drive failure rates reported by Google in [9], and server crash rates reported in [11]–[13], the failure rate of 2%, 5%, and 8% were tested in the simulation experiments, based on a random failure event occurrence model.

A. HADOOP EXPERIMENTAL ENVIRONMENT

Simulation experiments were based on the Hadoop profile for the Wordcount application in an EC2 environment with an input data size of 50 GB, with the number of map and reduce slots set at 20 each [4].

B. HADOOP JOB EXECUTION TIME ESTIMATION

First, the time predicted of a failure-less environment with a failure existing environment is compared, in order to check the effect of failures on the job execution time. In the failure-less environment, the IHP model is applied, and the time t_f at which the failure occurred is assumed to be immediately after the completion of the entire map and reduce tasks. In addition, in the event of a failure, it is assumed that rescheduling for a failed task and replenishment of a failed worker node occurred immediately. Fig. 3 shows the increased time due to failures. When a failure occurs in the map phase, the time consumed increases by 20∼30 seconds, but in the case a failure occurs in the reduce phase, the consumed time increased by 60∼110 seconds according to the input data size [4]. This is because when a failure occurs in the reduce phase, not only the failed reduce tasks but also the previously executed map tasks on the failed worker node have to be re-executed.

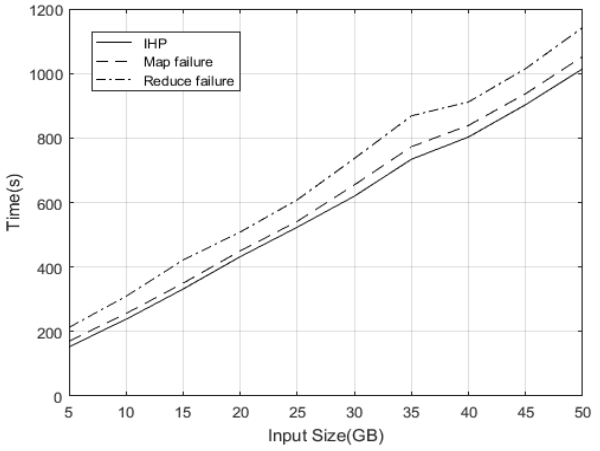
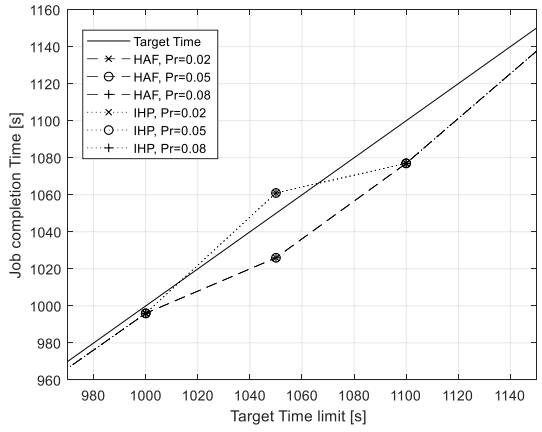
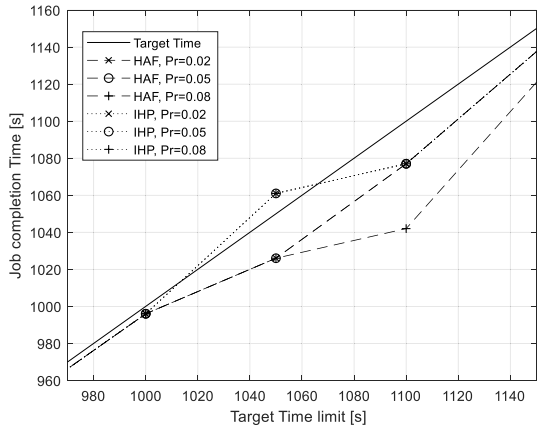


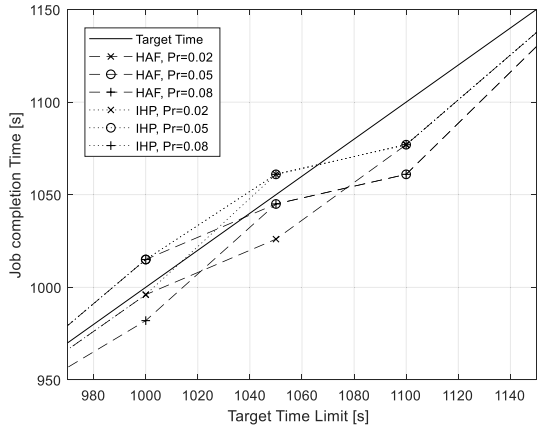
FIGURE 3. Comparison of no-failure and failure environments.



(a)



(b)



(c)

FIGURE 4. Hadoop job execution time estimation with failures. (a) $P_m = 0.02$. (b) $P_m = 0.05$. (c) $P_m = 0.08$.

C. HADOOP RESOURCE PROVISIONING

In the simulation experiments, it will be checked if the target times can be satisfied in an environment where actual failures occur. The probability of failure (P) to be applied to the map and the reduce phases is 0.02, 0.05, and 0.08, and the target times are 905, 1000, 1050, 1100, and 1195 seconds. Resource provisioning is conducted through a prediction model that is based on a dataset size of 50 GB, and it is assumed that failures occur randomly, with probability P_m and P_r at any time t_f. The results in Fig. 4 show that the target time is satisfied when resource provisioning for each model is set. Resources were provisioned according to the time and probability of failures, respectively, where the graphs present the mean performance values. The HAF model satisfied the target time in almost all cases tested. In comparison, as the probability of failure increased, and as the target time is reduced, the number of times the IHP model exceeds the target time increases. Overall, the IHP scheme achieved a 64% success rate for the high failure rate cases of ($P_m = 0.05, P_r = 0.08$) and ($P_m = 0.08, P_r = 0.08$), and otherwise maintained a 84% success rate for all other cases tested. Performance degradation occurred due to not being able to account for potential failures in the resource provisioning procedures, which confirm that reduce failures have a critical influence on the job complete time. In addition, the HAF achieved a 84% success rate for the high failure rate cases of ($P_m = 0.05, P_r = 0.08$) and otherwise maintained a 100% success rate for all other cases tested.

D. SPARK EXPERIMENTAL ENVIRONMENT

Simulation experiments on Spark were based on [18] where the Spark profile for the k-mean application is based on a 75 node cluster with iterations of 400 tasks working on 100 GB of data.

E. SPARK JOB EXECUTION TIME ESTIMATION

First, the time predicted of a no-failure environment with a failure existing environment is compared in order to check the effect of failures on the job execution time. In the no-failure environment, the OptEx [16] model is applied. In addition, in the event of a failure, it is assumed that rescheduling for a failed task and replenishment of a failed worker node occurred immediately. Fig. 5 shows the increased time due to failures. When failures occurred in the job, the time consumed

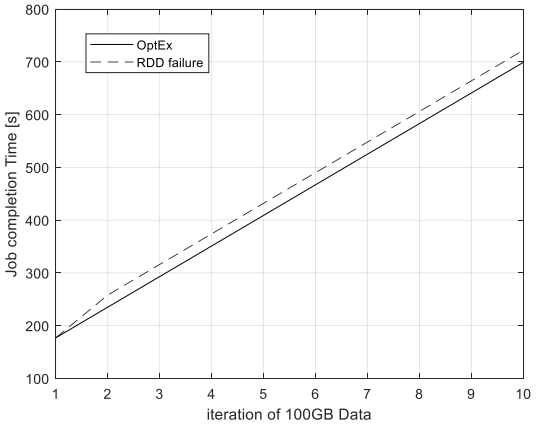
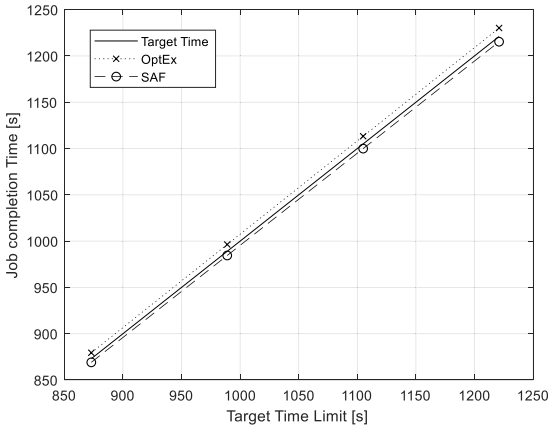
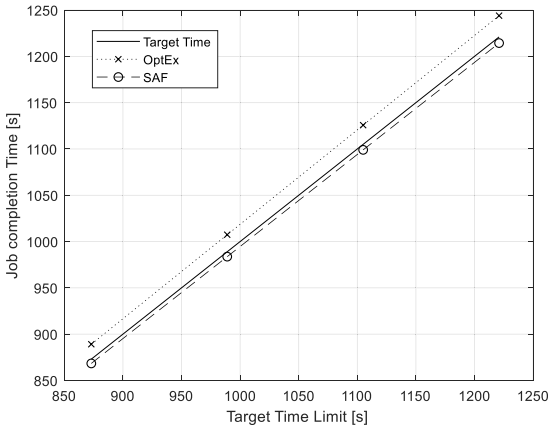


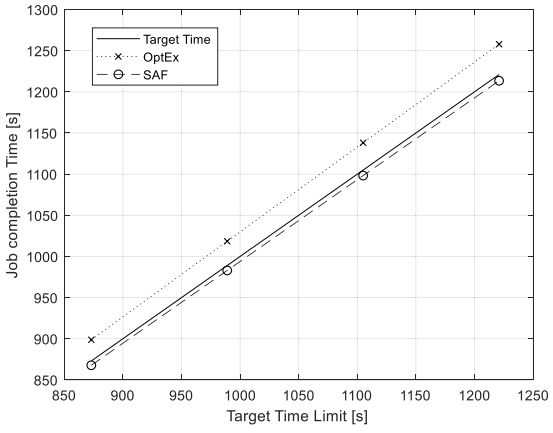
FIGURE 5. Spark resource provisioning and applying in case of failure.



(a)



(b)



(c)

FIGURE 6. **Spark job execution time estimation with failures. (a) $P_e = 0.02$. (b) $P_e = 0.05$. (c) $P_e = 0.08$.**

increases by 58 seconds, regardless of which part the failure had occurred in.

F. SPARK RESOURCE PROVISIONING

In the simulation experiments, it will be checked if the target times can be satisfied in an environment where actual failures occur. The probability of failure (P_e) to be applied to the RDD process is 0.02, 0.05, and 0.08, and the target times are 873, 989, 1105, and 1221 seconds [9], [11]. Resource provisioning is conducted through a prediction model that is based on a dataset size of 100 GB, and it is assumed that failures occur randomly, with the probability P_{RDD}. The results in Fig. 6 show that the target time is met when resource provisioning for each model is set. Resources were provisioned according to the time and probability of failures, respectively, where the graphs present the mean performance values. As the probability of failure increases, the OptEx model's [16] job complete time exceeds the target time for all cases tested (i.e., the success rate is 0%). In comparison, the SAF scheme achieved a 100% success rate for all cases of P_e tested. Overall, the results show that the OptEx scheme will experience performance degradation due to not accounting for potential failures in the resource provisioning procedures. However, since the proposed SAF scheme executes jobs based on optimally provisioned resources considering possible failures, it is able to satisfy the target times for the range of interest.

G. CONCLUSION

In this paper, the Spark and Hadoop job execution time estimation and resource provisioning schemes respectively named SAF and HAF have been proposed. SAF and HAF are based on generalized mathematical models that include the failure variables so they can adjust their resource distributions according to the phase (or progress of the failure) to complete the task within the target time. Since the SAF and HAF schemes compute the required optimal resource size according to the failure probabilities considering the cluster and network conditions, the system performance and failure probability as well as the deadline can be accurately predicted for RT and NRT applications, and can also be used for synchronized large scale coordinated clustered datasets analysis.

In future research, the influence of variating processing issues, such as individual CPU core adaptive operational frequency changes due to the on-demand governor, real-time governor, and other power control governors, needs to be investigated.

REFERENCES

[1] J. Dean, "Experiences with MapReduce, an abstraction for large-scale computation," in *Proc. PACT*, Seattle, WA, USA, Sep. 2006, p. 1.
[2] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta, "On availability of intermediate data in cloud computations," in *Proc. HotOS*, Monte Verità, Switzerland, May 2009, p. 6.
[3] A. Verma, L. Cherkasova, and R. H. Campbell, "Resource provisioning framework for mapreduce jobs with performance goals," in *Proc. 12th ACM/IFIP/USENIX Int. Conf. Middleware*, Lisboa, Portugal, Dec. 2011, pp. 165–186.
[4] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, "Hadoop performance modeling for job estimation and resource provisioning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 441–454, Feb. 2016.
[5] H. Herodotou. (2011). *Hadoop Performance Models*. [Online]. Available: http://www.cs.duke.edu/starfish/files/hadoop-models.pdf
[6] S. Kadirvel and J. A. B. Fortes, "Grey-box approach for performance prediction in map-reduce based platforms," in *Proc. 21st Int. Conf. Comput. Commun. Netw.*, Munich, Germany, Jul./Aug. 2012, pp. 1–9.

[7] H. Yang, Z. Luan, W. Li, D. Qian, and G. Guan, "Statistics-based workload modeling for MapReduce," in *Proc. 26th Int. Parallel Distrib. Process. Symp. Workshops PhD Forum*, Shanghai, China, May 2012, pp. 2043–2051,
[8] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *Proc. 10th IEEE/ACM CCGrid.*, Melbourne, VIC, Australia, May 2010, pp. 94–103.
[9] E. Pinheiro, W. D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proc. 5th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2007, pp. 17–23.
[10] H. Zhu and H. Chen, "Adaptive failure detection via heartbeat under Hadoop," in *Proc. IEEE Asia–Pacific Services Comput. Conf.*, Jeju Island, South Korea, Dec. 2011, pp. 231–238.
[11] L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. San Rafael, CA, USA: Morgan & Claypool Publishers 2013.
[12] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th ACM Eur. Conf. Comput. Syst. (EuroSys)*, Bordeaux, France, Apr. 2015, pp. 16–22.
[13] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.
[14] G. Wang, L. Zhang, and W. Xu, "What can we learn from four years of data center hardware failures?" in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Denver, CO, USA, Jun. 2017, pp. 25–36.
[15] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *Proc. 17th IEEE Int. Conf. High Perform. Comput. Commun.*, New York, NY, USA, Aug. 2015, pp. 166–173.
[16] S. Sidhanta, W. Golab, and S. Mukhopadhyay, "OptEx: A deadline-aware cost optimization model for spark," in *Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, Cartagena, Colombia, May 2016, pp. 193–202.
[17] M. T. Islam, S. Karunasekera, and R. Buyya, "dSpark: Deadline-based resource allocation for big data applications in apache spark," in *Proc. IEEE 13th Int. Conf. e-Sci. (e-Sci.)*, Auckland, New Zealand, Oct. 2017, pp. 89–98.
[18] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implement*, San Jose, CA, USA, Apr. 2012, p. 2.
[19] S. Gopalani and R. Arora, "Comparing apache spark and map reduce with performance analysis using K-means," *Int. J. Comput. Appl.*, vol. 113, no. 1, pp. 8–11, Jan. 2015.
[20] Y. Xia, R. Ren, H. Cai, A. V. Vasilakos, and Z. Lv, "Daphne: A flexible and hybrid scheduling framework in multi-tenant clusters," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 330–343, Mar. 2018.
[21] Z. Hu, B. Li, Z. Qin, and R. S. M. Goh, "Job scheduling without prior information in big data processing systems," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Atlanta, GA, USA, Jun. 2017, pp. 572–582.
[22] T. Subbulakshmi and J. S. Manjaly, "A comparison study and performance evaluation of schedulers in Hadoop YARN," in *Proc. IEEE ICCES*, Coimbatore, India, Oct. 2017, pp. 78–83.
[23] X. Hou, T. K. A. Kumar, J. P. Thomas, and H. Liu, "Dynamic deadline-constraint scheduler for Hadoop YARN," in *Proc. IEEE SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI*, San Francisco, CA, USA, Aug. 2017, pp. 1–8.

JINBAE LEE received the B.S. and M.S. degrees from the School of Electrical and Electronic Engineering, Yonsei University, South Korea, in 2017 and 2019, respectively, where he was a Research Member with the Communications and Networking Laboratory, from 2017 to 2019. Since 2019, he has been with Samsung Electronics, Co, Ltd. His research interests include Apache Hadoop, Spark, and Storm big data systems and advanced 5G networking technologies.

BOBAE KIM received the M.S. degree from the School of Electrical and Electronic Engineering, Yonsei University, South Korea, in 2018, where she was a Research Member with Communications and Networking Laboratory. She is currently a Captain of the Republic of Korea Army. Her research interests include big data systems and 5G networking technologies.

JONG-MOON CHUNG (SM'04) received the B.S. and M.S. degrees in electronic engineering from Yonsei University, and the Ph.D. degree in electrical engineering from Pennsylvania State University, where he was an Assistant Professor and Instructor, from 1997 to 1999. From 2000 to 2005, he was with Oklahoma State University as a tenured Associate Professor. Since 2005, he has been a tenured Professor with the School of Electrical and Electronic Engineering, Yonsei University. Since 2019, he has been a Vice President of the IEEE Consumer Electronics Society. His research interests include big data, cloud computing, MEC, smartphones, smart cars, smart factory, deep learning, IoT, AR, 5G, 6G, SDN, and NFV. He is a member of the IET and IEICE and a Life Member of the HKN, KSII, IEIE, KIHM, and KICS. In 2000, he received the First Place Outstanding Paper Award at the IEEE EIT 2000 Conference. In 2003 and 2004, respectively, he received the Distinguished Faculty Award and the Technology Innovator Award. In 2005, he received the Regents Distinguished Research Award and the Halliburton Outstanding Young Faculty Award. In 2008 and 2018, he received the Outstanding Accomplishment Professor Awards. In 2012, he received the Defense Acquisition Program Administration Award from the Korean Government. In 2007, 2009, and 2014, he received the Outstanding Teaching Awards. In 2017, he received the KSII Academic Excellence Award. He is currently an Editor of the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the Co-Editor-in-Chief of the *KSII Transactions on Internet and Information Systems*, and a Section Editor of the *ETRI Journal* (Wiley). His courses in the Coursera Specialization Program titled "Emerging Technologies: From Smartphones to IoT to Big Data" and the courses "Deep Learning for Business" and "Introduction to TCP/IP" are among the most popular in the ICT technology field.