Predicting Misconfiguration-induced Unsuccessful Executions of Jobs in Big Data System

TANG Hongyan[1], LI Ying[2,*], WANG Long[3], GU Jing[1], WU Zhonghai[1,2]
[1] School of Software and Microelectronics, Peking University, Beijing, China
[2] National Engineering Center of Software Engineering, Peking University, China
[3] T. J. Watson Research Center, IBM, USA

***Abstract*—As the complex workload scheduling and resource allocating mechanism in big data system, programmers' configuration error is one of the most typical root causes of unsuccessful termination of jobs, which can result in performance deterioration, availability degradation, resource inefficiency and user unsatisfactory. In this paper, we propose an approach called SD-Predictor, to predict misconfiguration-induced unsuccessful executions of jobs combining static job configurations and dynamic runtime system state before scheduling and execution, so as to save computing resource and scheduling overheads in big data system. We implement and incorporate SD-Predictor with a popular scheduling framework YARN to optimize job scheduling so as to avoid negative impacts by misconfigured jobs. Moreover, we explore correlations between configurations and termination status of jobs and provide some recommendations for configuration optimization. The experiment results show that our approach performs at 78% of precision, 52% of recall and 2% of false positive rate in unsuccessful job prediction, with significantly better recall and false positive rate than related works.**

Keywords- misconfiguration; unsuccessful execution; prediction; big data; OpenCloud

I. INTRODUCTION

Analytic applications running on big data systems such as Hadoop and Spark may result in unsuccessful executions (failed, killed, etc.) due to different errors/failures, including crash of nodes (VMs/containers), hardware failure, software bug and misconfiguration. Particularly, misconfiguration is quite common for execution of analytics applications because

i) Performance (success or unsuccess) of such Hadoop and Spark applications has a large dependency on the configurations of how many mapper tasks and how many reducer tasks an application job is split into, how much memory is requested by the individual tasks, etc.

ii) Many users of big data systems and analytic application developers lack the expertise on setting the configurations properly. For example, a large number of questions are asked in online forums on configuration understanding, configuration tuning guidance, and configuration fixing on Hadoop, Spark and other big data systems [19].

Therefore, it is essential to predict the termination results (success or unsuccess) of analytic application jobs with given configurations before execution. Aided by accurate prediction, users and developers on big data systems can greatly reduce the probability of unsuccessful execution and minimize the time cost and resource cost due to unsuccessful execution.

Prior studies on termination status prediction tend to predict job failures during runtime after the job executions have started [12][2][16][17][11][7][13]. But these prediction techniques still incur a large amount of time cost and resource cost as job scheduling and part of job execution are done before the prediction.

To minimize the time cost and resource cost caused by the unsuccessful job execution, in this paper we propose an approach called SD-Predictor to predict job execution results before job scheduling. Specifically, we predict job execution results by combining static job configurations and dynamic runtime system metrics that are post job submission and prior to job scheduling. Experiment results show that, by properly using the post-submission prior-scheduling runtime system metrics, we can predict unsuccessful job executions with better prediction performance, compared with state-of-the-art technologies, while our solution eliminates the time cost and resource cost for scheduling and executing those jobs with unsuccessful results, which are incurred in these technologies.

Here is an outline of our approach. There are two phases, training phase and prediction phase, involved in the approach. During the training phase, we feed our prediction mechanism the job configurations, collected post-submission prior-scheduling runtime system metrics, and the job execution results from history of the analytic application job executions. The prediction mechanism constructs a binary decision tree based on runtime system metrics, and establishes correlations among the binary decision tree output (called *regions*), job configurations, and job execution results. During the prediction phase, we capture the runtime system metrics immediately after job submissions as input of the binary decision tree, and obtain its output region. Then we compute the job execution result (success or unsuccess) by combining the output region and the established correlations among the regions, configurations, and job execution results. As our prediction is performed quickly (*less than 3 milliseconds*), users and developers stop the submitted job and resubmit it with adjusted configurations if the job is predicted to be unsuccessful. Classical binary decision tree model CART [25] and prior-art classification technologies of correlating job configurations and job execution results, such as GBDT [20], KNN [21], RF [22] and LR [23], are leveraged in our prediction mechanism.

In summary, we make the following contributions:

- We propose an approach to predict misconfiguration-induced unsuccessful executions of jobs before the job scheduling and job execution start. This eliminates the time cost and resource cost wasted by

*corresponding author. The work is supported by Key Program of National Natural Science Foundation of China (Grant No. 61232005).

0730-3157/17 $31.00 © 2017 IEEE
DOI 10.1109/COMPSAC.2017.191

unsuccessful job executions and greatly improves the throughput of job execution on big data systems.

- We implement our approach in a popular scheduling framework YARN and evaluate it via experiments on OpenCloud dataset [1] and simulated dataset. Experimental results show that our approach achieves 78% of precision, 52% of recall and 2% of false positive rate in unsuccessful job prediction. Compared with related works, our approach's performance is much better in terms of recall and false positive rate, and is slightly better in terms of precision.

- We analyze correlations between configurations and job termination status in OpenCloud to provide insights and suggestions for configuration tuning and optimization.

II. Related Work

A. Failure Prediction

Failure prediction is an important research area for proactive failure management. As symptoms of system, logs and monitoring metrics are essential sources to predict system failure. Watanabe et al. [16] extract message patterns from system logs and apply Bayesian method to correlate message patterns and system failures for failure prediction. Chalermarrewong el al. [17] employ ARMA model to predict resource usage in the future, then construct a Fault Tree to utilize predicted resource usage to predict system availability.

In addition to system failure, lots of works focus on predicting job failure. Fadishei et al. [11] analyze correlations between job failures and job attributes such as resource usage based on Grid Workload Trace. They then utilize a Bayesian network to predict job failures in grid computing system. Chen et al. consider task attributes and resource usage of tasks as features, then employ RNN model to predict task failures, and apply the prediction results to predict job failure [7]. Rosa et al. [13] combine static features e.g. resource request and priority with dynamic system loads to predict failures of jobs after submission. EI et al. predict job failure based on job configurations and job running status, they find that knowing whether at least one task has failed in a job can improve accuracy significantly [2]. In contrast to these works, we aim to predict unsuccessful jobs combining job configurations and instant system state before job execution in big data system.

B. Failure Analysis on Public Dataset

Many researchers try to analyze public dataset to better understand failures in distributed systems. Chen et al. [5] analyze job failure in terms of scheduling constraints, resource usage and user attributes in Google cluster[4]. El-Sayed et al. [2] explore and compare characterization of failed, killed and finished jobs in Google cluster, OpenCloud and a HPC cluster. Rosa et al. [9] analyze performance impacts including machine time and resource waste of unsuccessful executions at the levels of job, task and event. They also try to disclose patterns and root causes of unsuccessful executions. Garraghan et al. [6] explore statistical distribution of server failures and task failures. They also study MTTF and MTTR of servers and tasks respectively. Birke el al. [10] capture and compare failure patterns including failure rate, time between failure and repair time of virtual machine and physical machine. Moreover, they uncover dependency of failures on resource capacity and usage. Tang et al. identify killer tasks which fail frequently and continuously from large amount of running tasks to notify administrator to take proactive task maintenance actions, so as to save computing resources and avoid scheduling overhead [24]. Different from these works, we mainly focus on correlations between configurations and termination status of jobs to prediction job failure before their execution, and provide guidance for configuration tuning.

III. Methodology

SUCCESS, FAILED and KILLED are three termination status of jobs in big data system[1]. Jobs which experience failure during runtime will terminate with FAILED, while KILLED indicates that the job was killed by administrator or user. In contrast to SUCCESS, we consider FAILED and KILLED as unsuccessful execution of jobs. In this section, we propose an approach called SD-Predictor to predict termination status of jobs (success or unsuccess) combining static job configurations and dynamic runtime system state. We first present feature sets and then elaborate framework of SD-Predictor.

TABLE I. Features Of SD-Predictor

Feature Set	Feature	Type	Description
Configuration Parameters	JobId	Continuous	The ID of the job assigned by the system.
	NumberOfMapper	Continuous	The number of mapper tasks of the job.
	NumberOfReducer	Continuous	The number of reducer tasks of the job.
	MaximalAttemptOfMapper	Continuous	The maximal attempts of a mapper task before the system give up it.
	MaximalAttemptOfReducer	Continuous	The maximal attempts of a reducer task before the system give up it.
	RequestMemoryOfMapper	Continuous	Memory request of each mapper task of the job.
	RequestMemoryOfReducer	Continuous	Memory request of each reducer task of the job.
	SpeculativeOfMapper	Categorical	Whether enable speculative execution of mapper tasks.
	SpeculativeOfReducer	Categorical	Whether enable speculative execution of reducer tasks.
	CompressOfMapper	Categorical	Whether compress the output of mapper tasks.
Runtime System Metrics	NumberOfJobs	Continuous	The number of jobs in the system when the job is submitted.
	NumberOfTasks	Continuous	The number of tasks in the system when the job is submitted.
	ReservedMemory	Continuous	Size of reserved memory of the system when the job is submitted.

A. Features

We mainly consider the following two feature sets in SD-Predictor.

1) Configuration Parameters. Jobs are submitted with lots of configurations such as the number of tasks and requested resource in big data systems. Correlated with task scheduling, resource allocation and fault tolerance, configurations influence the performance and even execution results of jobs. Thus configuration parameters are important indicators of unsuccessful job prediction. In this paper, we develop our approach based on two Hadoop clusters (OpenCloud and an experimental cluster) as case studies, and we select 10 configuration parameters of Hadoop system as features based on EI's works [8] and experience knowledge. Table I illustrate details of these 10 configuration parameters.

2) Runtime System Metrics. Instant system state after job submission also have impacts on termination status of jobs. For instance, a job may fail as out of resource if free resource is not enough. Moreover, a job may be killed if current workload exceeds system capacity. Thus system state such as available resource and system workload is essential to predict termination status of jobs. According to Table I, we consider three metrics, e.g. the number of jobs, the number of tasks and the amount of reserved resource in the system to measure system state. Due to limitation of the dataset, we only consider reserved memory as reserved resource in our experiment. We may include more available metrics to enrich features in general systems.

B. Framework of SD-Predictor

The basic idea of SD-Predictor is to split system state into several regions through a binary decision tree, then classify jobs based on their configurations in each region. We describe components of SD-Predictor shown in Figure 1 as follows.

1) Parameter Extraction

The responsibility here is to extract configuration parameters of jobs from configuration files and process them to generate feature vectors. For configurations decided by multiple factors, we should transform them based on corresponding calculation rules. For categorical configuration with discrete values, we encode it as one-hot feature. To keep same weight for all configuration parameters in prediction model, we scale all features into [0,1] through Min-Max scaler [14]. Finally, we organize all extracted configurations in certain sequence to generate a feature vector.

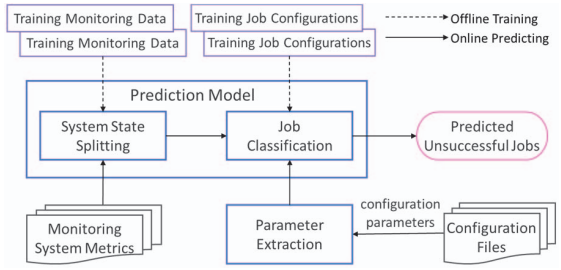

Figure 1 Framework of SD-Predictor

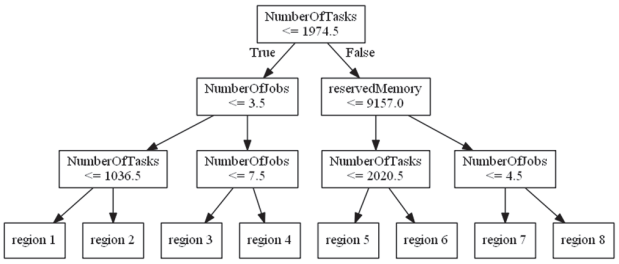

Figure 2 Example of System State Splitting Model

2) System State Splitting

As aforementioned, runtime system state is influential on termination status of jobs. Here we aim to employ monitoring metrics to split system state into several regions, where each region represents different conditions of the system. In offline training phase, we apply monitoring data of training jobs to train a binary decision tree model to split system state. While in predicting phase, we assign the job into a region based on the trained splitting model. Height of the decision tree is configurable as a parameter. In our case study, we set it as 3 to split system state into 8 regions and choose a classical binary decision tree model CART, which choose best split based on GINI impurity between features and classes, to split system state. Figure 2 is an example of system state splitting model based on OpenCloud dataset. There are 7 internal nodes decided by monitoring metrics and 8 leaf nodes which represent 8 regions in the decision tree. Based on the decision tree, we can find the corresponding region with monitoring results of these 3 metrics provided.

3) Job Classification

In this step, we aim to employ classification technologies to correlate binary decision tree output (regions) in last step and job configurations with execution results of jobs. As correlations between configurations and termination status of jobs may be different with different system states, we separate classification processes in different regions of system state. In training phase, we train job classification models based on configuration vectors in each system state region. For a newly submitted job, we first choose the corresponding classification model based on its system state region, then feed extracted configuration vectors into the classification model to classify the job as successful or unsuccessful. Note that the prediction framework is flexible, many proper classification algorithms can be plugged in to classify jobs. In our case study on OpenCloud dataset, we choose 4 classical classification algorithms, GBDT, KNN, RF, LR, as classifier for job classification.

IV. Implementation And Evaluation

In this section, we implement SD-Predictor to incorporate it with a scheduling framework YARN and evaluate it with experiments. We first elaborate implementation of the optimized scheduler, then analyze experiment results.

A. YARN with SD-Predictor

We implement our prediction approach and integrate it with a popular distributed resource management framework,

YARN [15] to optimize job scheduling. Figure 3 shows the architecture of the optimized scheduler. The Resource Manager is a core component of YARN, which plays a role as a negotiator and is responsible for resource tracing and task scheduling. While the client works as a controller to connect users and Resource Manager. According to design of YARN, user submits job through client, then Resource Manager receives information of the job from client, and allocates resource and schedules it onto a container. However, in the architecture of optimized scheduler, SD-Predictor obtains job configurations and monitoring results from client and monitoring system respectively after job submission, then feed configuration vector and runtime system metrics to prediction model to predict termination status of the newly submitted job. If the job is predicted to be successful, client pass it to Resource Manager to schedule it directly. Otherwise, client will alarm with misconfiguration and notify the job owner to optimize its configurations.

To achieve flexibility of optimized scheduler, the SD-Predictor is designed to be switchable through a configuration parameter. For instance, the SD-Predictor can be disabled in the beginning period of a big data cluster, as data is not enough to train a prediction model. After collecting sufficient data, we can train a prediction model and evaluate it. If the model achieves satisfactory performance, we then enable the SD-Predictor, otherwise we should keep collecting data and updating the prediction model. Moreover, the SD-Predictor works in a self-learning fashion to keep collecting termination status of jobs as feedback of prediction, then update and optimize the model based on misclassified jobs iteratively.

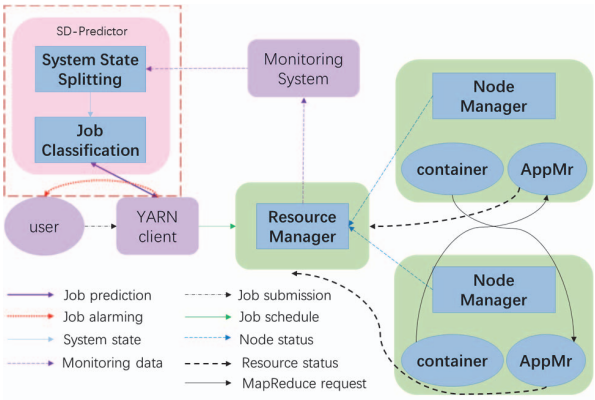

Figure 3 YARN with SD-Predictor

B. *Experiment Results*

We conduct 2 experiments to evaluate SD-Predictor. Experiment 1 is based on OpenCloud dataset, while Experiment 2 develops a case study with simulated jobs in an experimental Hadoop cluster. We assess prediction results based on 4 commonly used metrics in machine learning, e.g. precision, recall, accuracy and false positive rate (FPR). We describe these two experiments in detail as following.

1) *Experiment 1*

OpenCloud dataset contains 30-month log files collected in a 64-node Hadoop cluster, including 60,463 successful jobs and 8,445 unsuccessful jobs. Here we conduct 10-fold cross validation [3] based on OpenCloud dataset to evaluate SD-Predictor comprehensively. As aforementioned, we consider and compare 4 different classification algorithms, e.g. LR, RF, KNN, and Gradient Boosting Decision Tree (GBDT) to classify jobs. The experiment results are shown in Figure 4.

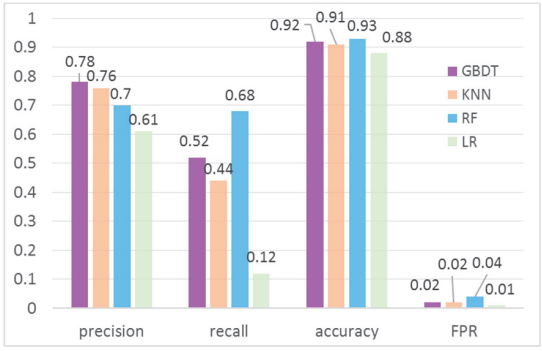

Figure 4 Prediction Results of SD-Predictor

According to Figure 4, SD-Predictor achieves 78% of precision, 52% of recall, 92% of accuracy and 2% of false positive rate in the best case with GBDT as job classifier. In particular, RF tends to predict more jobs as unsuccessful, which is aggressive with highest recall and worst false positive rate. On the contrary, KNN performs conservatively and predicts most jobs as successful to achieve lowest recall and false positive rate. Except for best false postive rate, LR performs with worst precision, recall and accuracy, which proves unsuccessful job prediction is not a linear problem.

Figure 5 compares our experiment results with state-of-the-art related works. We choose the best results under similar prediction circumstance from their works and make some transformations on evaluation metrics to guarantee unification. For instance, both EI et al. [2] and Chen et al. [7] consider specificity as evaluation metric, which equals to 1 minus false positive rate. In addition, Rosa et al. [13] evaluate their approach with misclassification rate, which equals to 1 minus accuracy. According to Figure 5, SD-Predictor outperforms related works with better precision, recall, and false positive rate. Specifically, SD-Predictor achieves significantly higher recall and lower false positive rate than related works. What's more, our approach needs less features

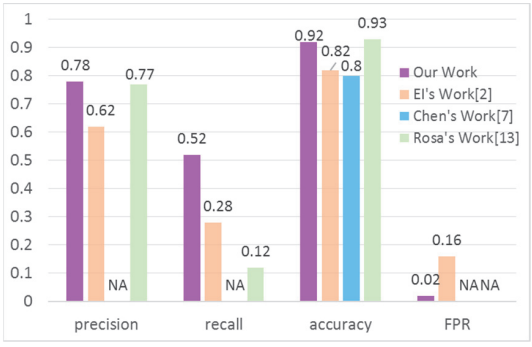

Figure 5 Prediction Comparison with Related Works

TABLE II. SIMULATION OF CONFIGURATION PARAMETERS

Configuration	Type	Range
NumberOfMap	Continuous	[0, 6]; [7, 63]; [64, 366]; [367, 908]
NumberOfReducer	Continuous	[1,1]; [1,1]; [1,49]; [50,75]
MaximalAttemptOfMap	Continuous	[1, 4]
MaximalAttemptOfReducer	Continuous	[1, 4]
RequestMemoryOfMap	Continuous	[1G, 2G]
RequestMemoryOfReducer	Continuous	[1G, 2G]
SpeculativeOfMap	Categorical	True / False
SpeculativeOfReducer	Categorical	True / False
CompressOfMap	Categorical	True / False

and achieves better results with same dataset, compared to EI et al.'s work whose prediction scenario is most similar to us.

2) Experiment 2

In addition to public dataset, we also build a 3-node Hadoop cluster to evaluate SD-Predictor comprehensively. To simulate typical usage of Hadoop cluster, we implement a job simulator to submit 4 classical MapReduce application jobs (e.g. WordCount, KMeans, TopK and InvertedIndex) following Poisson distribution. The simulator simulates configurations based on OpenCloud dataset as it represents typical scientific usage of Hadoop cluster. It is natural to enumerate true or false to simulate categorical configurations. While for continuous configurations, we only consider integer values to avoid explosive simulation space. According to Table II, we ignore statistical outliers and split it into 4 intervals according to 25% quantile, 50% quantile and 75% quantile to simulate configurations spanning in a wide range. The simulator enumerates all combinations of all intervals of all configurations to cover different configurations of jobs in real system. After 26 days' simulation, we collect data of 12,000 jobs, with 7368 successful and 4632 unsuccessful jobs.

We choose GBDT which performs best in Experiment 1 as classifier and conduct 10-fold cross validation with simulated jobs. Results show that SD-Predictor achieves 80% of precision, 95% of recall, 89% of accuracy and 14% of false positive rate in the experimental Hadoop cluster, which proves good performance and applicability in general big data system of our approach further. Moreover, we monitor time consumption of prediction and results show that it takes less than 2.2 milliseconds to predict termination status of a job on average, which is negligible in the execution process of jobs.

V. CONFIGURATION TUNNING

In this section, we analyze correlations between configurations and termination status based on OpenCloud dataset to provide some coarse-grained recommendations for configuration tuning. We mainly focus on configurations of memory request, speculative execution and task attempt here.

A. Memory Request

A job should declare the size of memory each map task and reduce task request in Hadoop. According to scheduling principal of YARN, a job will be killed directly if the actual memory usage exceeds memory request [18]. In addition, a job will terminate with failure due to out of memory if available memory is not enough. Therefore, the ratio of memory request to available memory is an important factor of termination status of jobs. Lacking enough information in the dataset, we only consider a similar metric, which is the ratio of reserved memory to memory request, to uncover correlation between memory request and termination status. Figure 6 indicates that successful and unsuccessful job distributes differently in terms of the metric, the ratio of successful job is less than unsuccessful job, especially in reducers. Therefore, reducing memory request of mappers or reducers is an option to decrease the ratio of reserved memory to memory request, which means decrease the ratio of memory request to available memory, so as to avoid unsuccessful execution induced by out of memory.

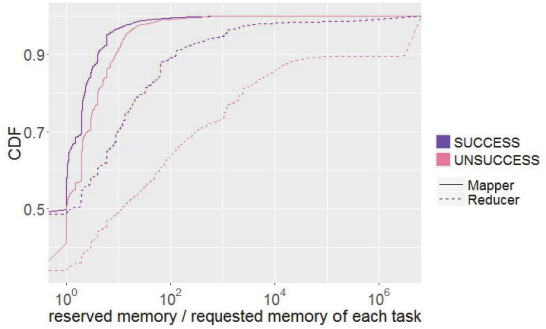

Figure 6 Memory Request and Termination Status

B. Speculative Execution

Speculative Execution is an optimization strategy to deal with stragglers in Hadoop. If the speculative execution is switched on, the scheduler will schedule an equivalent task as a backup when it detects straggling task that progresses slowly. If the backup task completes faster, performance of the job will be improved. Therefore, speculative execution is an important configuration of jobs to achieve fault tolerance. According to correlations between speculative execution and termination status shown in Figure 7, speculative execution of some jobs are configured to switch off though it is default to be true. However, jobs are more likely to be terminate unsuccessfully if speculative execution of mapper or reducer is disabled. Therefore, we suggest that switching on speculative execution of mapper and reducer can improve fault tolerance and successful probability of job.

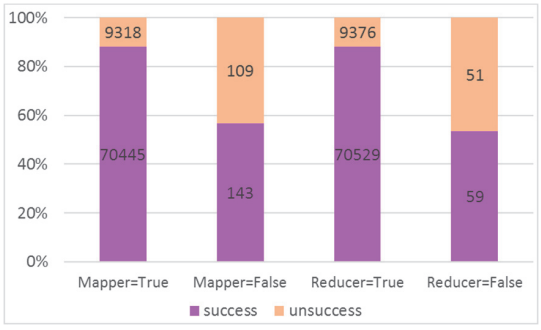

Figure 7 Speculative Execution and Termination Status

C. Task Attempt

Scheduler will restart a task attempt if it detects unsuccessful execution of a task. A job can set the maximal attempts in configuration files, which is default to be 4. According to the correlation between maximal task attempts and termination status shown in Figure 8, successful jobs tend to have larger attempts of mappers and reducers compared to unsuccessful jobs. An intuitive explanation is that larger attempts imply more opportunities to reschedule unsuccessful tasks, which will reduce the probability of unsuccessful execution. Thus increasing the maximal attempts of mappers and reducers is another configuration tuning option.

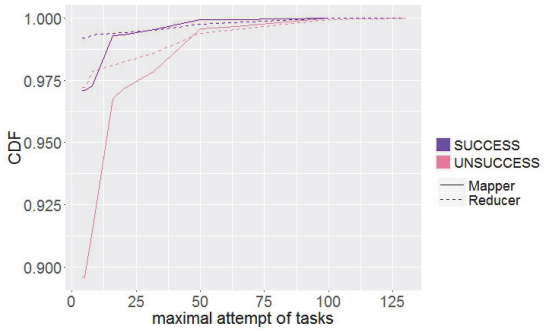

Figure 8 Task Attempt and Termination Status

VI. Conclusion

Misconfiguration-induced unsuccessful executions of jobs prevail in big data system. This paper proposes an approach called SD-Predictor to predict termination status of jobs before scheduling based on configurations and system state metrics. We implement SD-Predictor and incorporate it with YARN to optimize job scheduling. We evaluate our approach with OpenCloud dataset and simulated jobs in an experiment Hadoop cluster, results show that SD-Predictor outperforms than related works with significantly higher recall and lower false positive rate in an extremely short period. Moreover, we provide some configuration tuning recommends to help achieve successful executions through exploring correlations between configurations and termination status of jobs. As future work, we plan to improve prediction precision further by considering techniques dealing with unbalanced data, e.g. oversampling and SMOTE. In addition, we'd like to explore dataset thoroughly to provide finer-grained job configuration tuning recommendations to achieve successful executions.

References

[1] CMU Parallel Data Lab Project. http://www.pdl.cmu.edu/HLA.

[2] El-Sayed N, Schroeder B. How reliable are large-scale jobs in parallel clusters?[J].

[3] Cross-validation (statistics), https://en.wikipedia.org/wiki/Cross-validation_(statistics)

[4] Reiss C, Wilkes J, Hellerstein J L. Google cluster-usage traces: format+ schema[J]. Google Inc., Mountain View, CA, USA, Technical Report, 2011.

[5] Chen X, Lu C D, Pattabiraman K. Failure analysis of jobs in compute clouds: A google cluster case study[C]//Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on. IEEE, 2014: 167-177.

[6] Garraghan P, Townend P, Xu J. An empirical failure-analysis of a large-scale cloud computing environment[C]//High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on. IEEE, 2014: 113-120.

[7] Chen X, Lu C D, Pattabiraman K. Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study[C]//Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on. IEEE, 2014: 341-346.

[8] Mishra A K, Hellerstein J L, Cirne W, Das C R. Towards characterizing cloud backend workloads: insights from Google compute clusters[J]. Acm Sigmetrics Performance Evaluation Review, 2010, 37(4):34-41.

[9] Rosa A, Chen L Y, Binder W. Understanding the Dark Side of Big Data Clusters: an Analysis beyond Failures[C]//Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on. IEEE, 2015: 207-218.

[10] Birke R, Giurgiu I, Chen L Y, et al. Failure analysis of virtual and physical machines: Patterns, causes and characteristics[C]// Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on. IEEE, 2014: 1-12.

[11] Fadishei H, Saadatfar H, Deldari H. Job failure prediction in grid environment based on workload characteristics[C]//Computer Conference, 2009. CSICC 2009. 14th International CSI. IEEE, 2009: 329-334.

[12] Chen X, Lu C D, Pattabiraman K. Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study[C]//Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on. IEEE, 2014: 341-346.

[13] Rosa A, Chen L Y, Binder W. Predicting and Mitigating Jobs Failures in Big Data Clusters[C]//Cluster, Cloud and Grid Computing, 2015 15th IEEE/ACM International Symposium on. IEEE, 2015: 221-230.

[14] Feature scaling, https://en.wikipedia.org/wiki/Feature_scaling.

[15] Vavilapalli V K, Murthy A C, Douglas C, et al. Apache hadoop yarn: Yet another resource negotiator[C]//Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013: 5.

[16] Watanabe Y, Otsuka H, Sonoda M, Kikuchi S, Matsumoto Y. Online failure prediction in cloud datacenters by real-time message pattern learning[C]//Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, 2012: 504-511.

[17] Chalermarrewong T, Achalakul T, See S C W. Failure Prediction of Data Centers Using Time Series and Fault Tree Analysis[C]//Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on. IEEE, 2012: 794-799.

[18] Best Practices for YARN Resource Management, https://www.mapr.com/blog/best-practices-yarn-resource-management.

[19] Questions in Stack Overflow about mapreduce.map.memory.mb, http://stackoverflow.com/search?q=mapreduce.map.memory.mb.

[20] Friedman J H. Greedy function approximation: a gradient boosting machine[J]. Annals of statistics, 2001: 1189-1232.

[21] Peterson L E. K-nearest neighbor[J]. Scholarpedia, 2009, 4(2): 1883.

[22] Liaw A, Wiener M. Classification and regression by randomForest[J]. R news, 2002, 2(3): 18-22.

[23] Logistic Regression, https://en.wikipedia.org/wiki/Logistic_regression.

[24] Tang H, Li Y, Jia T, et al. Hunting Killer Tasks for Cloud System through Machine Learning: A Google Cluster Case Study[C]//Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference on. IEEE, 2016: 1-12.

[25] Lewis R J. An introduction to classification and regression tree (CART) analysis[C]//Annual meeting of the society for academic emergency medicine in San Francisco, California. 2000: 1-14.