Job Execution Prediction on Hadoop Cluster using Artificial Neural Network

Mingyu Zeng
College of Computer Science
Zhejiang University
Hangzhou, China
21221233@zju.edu.cn

Huajun Chen
College of Computer Science
Zhejiang University
Hangzhou, China
huajunsir@zju.edu.cn

Jiaoyan Chen
College of Computer Science
Zhejiang University
Hangzhou, China
jiaoyanchen@zju.edu.cn

ABSTRACT

MapReduce programming paradigm and its open source implementation hadoop is becoming a major force in the big data scene. The primary reason for this situation is that hadoop makes data storage far less costly than prior methods of data storage simply by adding more commodity servers to a hadoop cluster. Hadoop presents a new software framework to manage large clusters of commodity hardware to run applications with petabytes of data. Researchers and enterprise users employ it for a variety of application domains including business data processing, text analysis, natural language processing, Web graph and social network analysis, and computational science. Efficient scheduling is essential for better resource utilization and maximal performance of hadoop clusters. Job execution prediction based on performance modeling is effective and accurate for scheduling applications, also a key aspect of efficient scheduling. This paper presents an artificial neural network based job execution prediction model for hadoop. We employ artificial neural network to model hadoop cluster performance by using a set of hadoop job performance characteristics to train artificial neural network and predict job execution times using the trained artificial neural network.

1. INTRODUCTION

The era of big data is approaching as data being collected by internet companies such as Facebook, yahoo, amazon, google, etc and gathered by ubiquitous information sensing mobile devices, remote sensing technologies, software logs, wireless sensor networks etc grows in size exponentially. Large and complex data sets are being collected for diverse reasons through all kinds of technologies. This situation poses a significant challenge on traditional data processing applications and on-hand data management tools. MapReduce developed by google and its open source implementation Apache Hadoop rises up as a competent data processing framework for huge data sets in cloud computing environments. Mapreduce reduces the complexity and difficulty of writing parallel distributed program which has been proven to be a massive and challenging undertaking requiring specialized skills. By taking care of many underlying tasks automatically such as parallelizing the jobs across a cluster of computing nodes, transferring data between various parts of the system, managing communication and synchronization, providing for redundancy and failure, it frees up the programmer from the burden of parallelization control and harnesses the potential of cloud computing for their data-intensive jobs . In this way, the programmer only need to focus on the logics of the problems and write a simple map and reduce function.

As the size of computing cloud expands and more and more jobs are submitted to the cloud, an issue becomes more and more evident: how to optimize scheduling? The undesirable performance is magnified by the large amount of simultaneous jobs and the quantity of data being processed. By maximizing the utilization of cluster resources, reducing resource contention, optimized scheduling should maximize the throughput, decrease waiting time and guarantee fairness between jobs. A key aspect of optimized scheduling is the ability to estimate job execution times.In this paper we describe and evaluate a new model that compute job execution time using trained artificial neural network using historical job executing metrics. Hadoop generates metrics for every MapReduce job, we use these metrics and job configuration features as job characteristics, the next step is to use historical job metrics to train artificial neural network for modeling the relationship between the characteristics and the job execution time, the final step is to form our estimation using the trained artificial neural network.

This paper is organized as follows: the next section discusses some related work regarding our work. Subsequent section presents our prediction technique. In section 4 we evaluate our approach; finally we summarize and outline the challenges of implementing our approach into real world usage and the future work.

2. RELATED WORK

Computation time estimation is not a new topic. Several studies [8] [3] [7] have explored various approaches regarding this area. Before [5],existing works use manually selected similarity template of job characteristics to determine similar jobs in a history. A similarity template is a set of job characteristics that we use to compare jobs to decide whether they are similar, but this approach has the following limitations: Identifying the best similarity template is a

massive undertaking and may not always be possible; There is no generic similarity template. Different approaches have been introduced to address this issue including using genetic algorithm or greedy-search techniques to automate this process. In [5] it proposes using rough set theory to automatically select similarity template to identify similar jobs ,then computing the prediction using statistical measures. Characteristic selection is to mitigate impact of the unrelated attributes by applying rough set theory to identify a set of characteristics that most relate to job runtime. Formalizing an information system using historical job execution is the first step. Attributes are categorized as the runtime being the decision attributes and all the other attributes being the condition attributes. Every set of condition attributes has some degree of dependency with the decision attribute, this degree measures the extent to which this set of features affect the runtime. The degree of dependency varies in the range [0, 1] with 1 indicating the total dependency and 0 indicating independency, the task is to identify the set of characteristic with the strongest degree of dependency with the runtime. Attribute significance indicates how much this attribute affects the dependency between a set of condition attributes and the decision attribute. First, computing the dependency between a set of condition attributes without the attribute which we want to calculate the significance and the decision attribute ,then computing the dependency between this same set of condition attributes but including the attribute and the decision attribute, the difference between these two dependencies demonstrates how this attribute affects the dependency and the attribute significance. After identifying the strongest degree of dependency between a set of condition attributes and the decision attribute , In[5] it select the minimal set of condition attributes that has the same degree of dependency as the strongest ,this set of condition attributes are referred as a reduct. Then it [5] uses this reduct as a similarity template to determine similar applications and compute the mean of the runtimes of these similar applications as the estimated runtimes.

In [7], It formalizes the historical job executions as the experience base, an experience represents all the input and output features of a job executed before, a prediction query is an experience with only input features to predict its output features. When a query is presented, several similar experiences are selected using the Heterogeneous Euclidean Overlap Metric as the distance function to choose the N nearest neighbors to the query as the relevant experiences within some distance of the query, then using a kernel regression function to compute the estimated value and confidence interval of every output feature. In order to choose optimal values of necessary parameters, It employs the genetic algorithm to search over training data to try different configuration in [7]. Several studies [2], [4] have applied instance-based learning techniques to predict runtimes of serial application for a relatively small set of applications. The instance-based learning algorithm employed by [4] works as follows: the first step is to find similar jobs using hadoop-specific input parameters, it used the Heterogeneous Euclidean Overlap Metric to quantify the distance between jobs and locate K-Nearest-Neighbors ; the second step is to predict job completion time using either the distance weighted average algorithm or the locally-weighted linear regression algorithm. Another early work of predicting job execution times of serial applications using statistical analysis and modeling an application with a state machine was [1].

Another approach proposed by researchers worth mentioning is to develop application and system models with much more detailed metrics and then use these models to predict job runtime, so this approach tends to be much more accurate. However this technique suffers from a fatal flaw that there is no performance model for an application running on hadoop with different configuration and cluster resources. An early exploration on this subject was in [8], It builds a job profile that compactly summarizes critical performance characteristics of a production hadoop job during map and reduce stages and design a MapReduce performance model based on the job profile and the performance bounds of completion time of different job phases. This model is used to predict resource requirement for a given job with a known profile and its soft deadline to complete within the deadline. They implement this performance model in hadoop for job scheduling and resource allocation which later are proven to be quite effective to meet job soft deadline until job demands exceed the cluster resources.Other researchers have used traditional query progress heuristics to predict the execution time of pig queries [6]. They estimate the remaining query execution time by comparing the number of tuples processed to the number of remaining tuples. But this technique is only taking effects halfway through the execution. Another disadvantage is that it cannot adapt to changing workloads.

3. PREDICTION TECHNIQUE

We form predictions using trained artificial neural network. Hadoop job histories contain a wealth of information about every aspect of the executed job, what happened during the execution and under what circumstances? How many map slots and how many reduce slots are there in the cluster and how many available for the job? How many map tasks and reduce tasks completed for the job to finish? How many bytes did the job read from the hadoop distributed file system? And how many did the job write to hadoop distributed file system? What is the map input records number and how about the map output records? All this job characteristics form a holistic profile for the job during every phase of its execution: map, shuffle, sort, and reduce. With all this data siting there gathering dust, why don‘t we put them into better use? So switch to a different perspective on how we perceive this information, it not only can tell us what happened to a job in the past but also we can use these information to predict what will happen to a job under the same circumstances. We know there is some sort of correlation between the job history metrics and the execution time. But the hard part lies in how to extract this correlation under multi-dimensionality. Artificial neural network is designed to tackle this kind of problem for modeling relationships between inputs and outputs.

3.1 The Basic Component of an Artificial Neural Network

A typical artificial neural network consists of three layers: the input layer comprised of input neurons represents all the predictive variables, the hidden layer, the output layer comprised of output neurons represents all the target variables. Neurons of the previous layer send data to neurons of the next layer via synapses-the connection between two different neurons. Every synapse stores a parameter

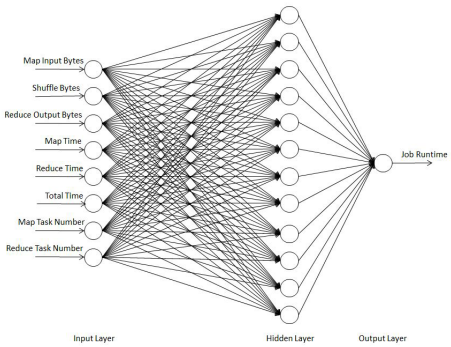

Figure 1: Artificial neural network model we used in our experiment

called weight representing the weight of signal transmitting through this connection. Every neuron defines an activation function which take the weighted input output by neurons of former layer connected to it and convert it to its output activation. Different neural network topologies distinguish each other from the number of hidden layers and neurons and the connections between neurons. A neural network learning process uses a set of observations to adjust weights between different neurons and find the optimal solution for a specific task. This process is controlled by a cost function which measures how far away a particular solution is from an optimal solution to a specific task by preventing from overtraining and controlling the training speed. For every record in the training set, it trains a neural network based on feedback method if the output is wrong. A training cycle is complete only when all the records in the training set have been used. A training process can have multiple training cycles or even hundreds of training cycles.

3.2 Training the Artificial Neural Network

We used a three layer BP-back propagation neural network, for activation function, we choose logarithmic sigmoid transfer function for the hidden layer and the linear transfer function for the output layer. The input layer has eight neurons, the Hidden layer has twelve neurons, and the output layer has one neurons. For the training function, we choose the gradient descent with momentum and adaptive learning rate back propagation. Figure 1 shows the artificial neural network model we used in our experiment,it demonstrates how input features enter the neural network and how neural network outputs output feature. We need at least set the following training parameters for the training function to begin: (1) the maximum number of epochs to train the network; (2) the performance goal; (3) the learning rate.

BP neural network training process is divided into two phases: propagation and weight update. In propagation phase, neurons from the input layer receive input vector from outside, pass it to neurons from the hidden layer. After receiving input vector, neurons of the hidden layer first calculate the inner product of the input vector and its weight vector, then the sum of this product and the neuron threshold is passed to the activation function of the hidden layer - logarithmic sigmoid transfer function as the input variable to calculate a scalar result. This result is passed to neurons of the output layer as a component of the output layer‘s input vector. Neurons from the output layer calculate the inner product of its input vector and weight vector, then the output layer activation function- linear transfer function computes a scalar result using the sum of the inner product and neuron threshold, this result is a component of the final network output vector, thus this completes the forward propagation. After this process, backward propagation begins, it first computes the deltas between teacher output and network output, using these deltas it calculates deltas of the hidden layer, so the error propagates backward from the output layer to the hidden layer.

In weight updating phase, training function updates weight and bias value according to gradient descent momentum and an adaptive learning rate. Weight gradient are calculated by multiplying output deltas and input activation passed from the previous layer. Weights are updated in the opposite direction by subtracting the product of the weight gradient and the learning rate, threshold is also updated. If the delta between the teacher output and network output is decreased, then the learning rate is adjusted by a ratio to increase learning rate. If the delta is increased by more than the max performance increase factor, the learning rate is adjusted by the ratio to decrease learning rate and the change that increase the performance is not made. The training process stops if any one of the following conditions occurs:(1) The maximum number of epochs is reached; (2) The maximum amount of time is exceeded; (3) The performance goal is met.

After the training process, we have a trained BP neural network. Artificial neural networks functionality lies in the fact that it can infer a function from observed data between inputs and outputs, when the complexity between inputs and outputs outweigh our ability to design such a function by hand. So this trained BP neural network embodies this function. Just like how a function works, we can use this trained BP neural network to produce output when an input is given.

4. PREDICTION RESULTS

In this section, we perform an initial performance experiment to justify and validate our proposed job execution prediction approach based on artificial neural network. We extracted data from hadoop job history logs which are collected by the cluster‘s jobtracker node to track every detail of a job‘s execution on the cluster, such as the errors, the occurrences of certain events, or performance data. The hadoop performs tasks on a cluster of eight virtual machine equipped with 1024 MB Random access Memory and 20 GB hard disks. The virtual machines are set up on a sever with 16 Intel Xeon processors, each processor has 4 2.4 GHZ cores , 16 GB RAM and 1 TB hard disks. We used hadoop 1.0.4 with one virtual machine being the job-tracker and the name-node, the rest of the seven virtual machines being the task-trackers and the data-nodes. The hadoop distributed file system blocksize is set to 64 MB. The replication level is set to 3, and we enabled the speculative execution. Our original extracted job metrics from the hadoop job history logs include the following: (1) the job id which uniquely identifies the job; (2) the job name; (3) the map input bytes which are the input bytes read by all the map tasks ; (4) the shuffle bytes ; (5) the reduce output bytes which are the output bytes output by all the reduce tasks; (6) the submit time of the job in seconds; (7) the duration time of the job

Table 1: Job Features We used for Our Approach

Input Features		
Feature	Type	Description
Map input bytes	bytes	Bytes input to all the map tasks
Shuffle bytes	bytes	Bytes shuffled after map phase before reduce phase
Reduce output bytes	bytes	Bytes output by all the reduce tasks
Map time	seconds	Addition of all the map task time
Reduce time	seconds	Addition of all the reduce task time
Total time	seconds	Addition of all the task time
Number of maps	integer	Number of all the map tasks
Number of reduces	integer	Number of all the reduce tasks
Output Feature		
Feature	Type	Description
Duration time	seconds	The time from the first launched task until it finishes

Table 2: Impact Factor of Every Input Feature

Input Features	Map input bytes	Shuffle bytes	Reduce output bytes	Map time
Impact Factor	-0.30	-0.21	0.1391	0.7020
Input Features	Reduce time	Total time	Number of Maps	Number of reduces
Impact Factor	-0.2172	-0.4859	1.1333	0.25

from the first launched task until the job finishes in seconds ; (8) the total time of all the map task added in seconds; (9) the total time of all the reduce task added in seconds; (10) the total time of all the map and reduce task added in seconds; (11) the number of map task ; (12) the number of reduce task ; (13) the hadoop distributed file system path of the input data; (14) the hadoop distributed file system path of the output data. For all this information paints a basic picture of a successful executed job. However in our experiment, Table 1 shows our choice of job characteristics as input and output features, for these job characteristics demonstrate the whole process of job execution.

We create two workloads from the extracted job metrics: A training workload and an evaluation workload. We trained the artificial neural network using the training workload and evaluated the performance of the trained artificial neural network by submitting the evaluation workload to the trained network to predict the runtime. In our experiment, we differentiated the evaluation workload from the training workload by removing the job output feature. Thus a test case consists of all the job input features except the runtime. The idea was to determine an estimated runtime using our prediction technique and compare it with the task's actual runtime.

In our experiment, we analyzed four aspects of our approach. We first demonstrate that our approach is valid by performing an initial test applying our technique. In the experiment, we use the training workload to train the artificial neural network, simulate predicted runtime using the evaluation workload, and calculate the mean relative prediction error which is 31%. Figure 2 illustrates the actual and predicted runtimes from one of our experimental runs. This demonstrates the validity of our approach. Figure 3 shows the relative prediction errors of each predicted job. Figure 2 shows that most of the predicted runtimes match its corresponding actual runtimes, while there are only a few with high prediction error, which we will explain why this

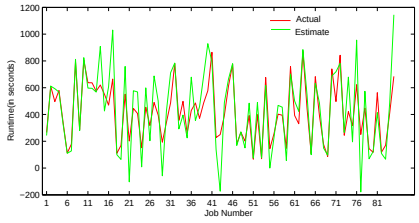

Figure 2: Actual versus estimated runtimes from our experiment

happens. We manually inspected the top-10 jobs with the largest prediction error, from our observation we conclude that these errors occurred mainly on the jobs with short execution times. These jobs are inherently difficult to model and predict. For one reason, the setup time for these jobs is the same time scale with the jobs with long runtimes; for another reason, jobs with short runtimes is more sensitive to workload changes of the hadoop cluster and performance problems such as configuration problem and process failures. Due to these circumstances, their runtimes will vary significantly and present a real challenge on modeling and predicting their runtimes. So from our initial test, we can safely conclude that our approach is valid, and the prediction error is under certain acceptable level.

In our second experiment, we analyzed how the job input features we used affect the prediction results. This process is carried out in three steps: first we train the network using the training workload with all the job input features and simulate results A using the evaluation workload with all the job input features ; second we train the network using the training workload but excluding the input feature which we want to analyze and simulate results B using the evaluation workload without the input feature we are analyzing; finally using results A and B , we compute the mean relative pre-

Table 3: Mean Relative Prediction Error of Different Input Feature Selections

Selection cases	case 1	case 2	case 3	case 4	case 5
Mean Error	0.3037	0.3454	0.3953	0.3543	0.3162

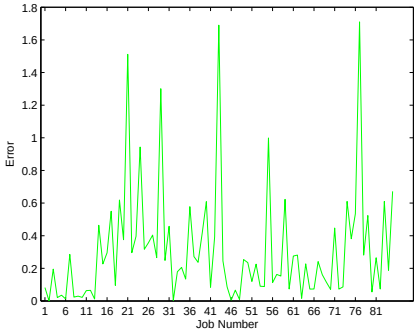

Figure 3: Execution time prediction error from our experiment

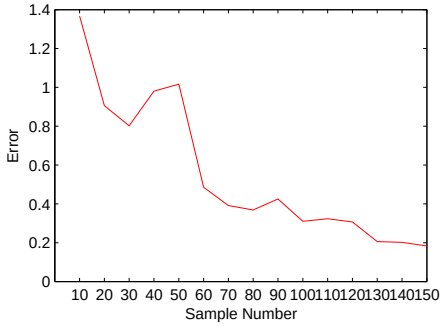

Figure 4: Errors of different amounts of training workload

diction errors E and J for both A and B, the impact factor can be calculated using function (J-E)/E. We can deduce from the results that If the impact factor is zero, so the input feature has no impact on the prediction; if the impact factor is negative, the input feature has negative impact on the prediction results which means The mean relative error will increase if the training workload include this input feature; if the impact factor is positive , the larger the value , the greater the impact, which means including this input feature in the training workload will decrease the mean relative error. Table 2 summarizes the impact factor of the eight input job feature. From the table we know the input feature with the largest impact factor is number of maps, this result is consistent with our workloads. Most of the jobs of the training workload and evaluation workload are map mostly jobs, these jobs have tens or even hundreds of map tasks, but they only have one reduce task. This could also explain why the input feature with the second largest impact factor is map time. Because most of the jobs tasks are map tasks, so map task time contributes significantly on the runtime. This results teach us when we pick job input features, There exists a need for us to understand what types of jobs we are dealing with, which input feature determines the execution of the jobs, if necessary, the jobs‘ running process.

The third aspect of our approach we analyzed is how different amounts of training workload affect prediction results. For this analysis, we devised a series of experiments to test different amounts of training workload and their estimation. Figure 4 summarized the mean relative prediction errors of different amounts of training workload. From the figure, we can see that when we use the training workload with 10 samples, the mean relative prediction error is 136%, which means the simulation results tend to be much larger than the actual runtimes. When we gradually add up the amount of the training workload, the mean relative prediction error slightly decreases, but still remains at a high level. This situation is easy to understand, when we initialize the artificial neural network, we set two most important training parameters for our experiment, the performance goal and the training epochs. The network stops training if the performance goal is met or the maximum training epochs are reached. So when the amount of training samples is small, this causes the performance goal can be met easily in a very small amount of training epochs. So when the training stops, the network is not adequately trained which leads to high relative prediction error when we employ the network to simulate estimation. There is a dramatic decrease when the training samples reach 60. This result reveals two facts regarding our approach. The first fact is that we need only a small amount of training samples to train an artificial neural network adequately to simulate results with acceptable prediction error. The second one is that when training stops because the maximum number of epochs is reached before the performance goal is met, the neural network is adequately trained for our approach to apply. But for this condition to apply, we have to consider an appropriate maximum number of epochs. If the maximum number is too small, then this could lead to an inadequate trained neural network; if the maximum number is too big, this could waste unnecessary training or even overtrain the neural network. When the training samples reach a certain amount, the maximum training epochs required to meet the performance goal increase exponentially. These two facts work as great advantages for our approach, for our approach to apply, we dont need very large amount of training samples to train the neural network, we just need a small amount of the newest executed jobs metric to represent clusters current workload and status, this eliminates the interference of history jobs executed under a very different cluster workload and status from the jobs whose runtimes we want to estimate. Another benefit of our approach is that since the training required to apply our approach is small because we dont need the network to achieve its performance goal plus the training samples are small too, our approach can respond quickly for each prediction request. The mean relative prediction error remains at this level for the following several training workloads with different amounts until the slight decrease achieved when the amount of training workload reaches 100, we can conclude that the mean rela-

tive prediction error can be further decreased if the training workload is large enough, but this comes at a huge price of a disturbing amount of unnecessary training for a very slight dispensable decrease.

The final aspect of our approach we analyzed is how different kinds of input feature selection impact the prediction results. We implemented our experiments by first selecting four input feature with the largest impact factor calculated from our previous experiments, then using the training workload with these four input features to train the network, simulating results using the evaluation workload with these four input features, finally computing the mean relative prediction error. For the next experiment, we add one input feature with largest impact factor in the remaining features, and run the experiment under the same experimental condition. We repeat this process until there is no remaining input feature. We run each experiment several times and compute an average value of the mean relative prediction errors from the experiments. Table 3 describes the average value of mean relative prediction errors from each set of experiments. From the table, we can see that the average values are very close to each other. We can dram two conclusions from the results. The first one is that training workload and evaluation workload including input features with strong impact factors have the same prediction results as the training workload and evaluation workload including all the input features. This conclusion can guide us when we select our input features, we need only focus on the ones which dominate the jobs‘ execution and exclude others to reduce unnecessary computation and save time. This requires us to have a clear understanding of jobs in the training workload, the jobs execution process. The second one is that we can improve the prediction results by adding more similar jobs to the training workloads.

5. FUTURE WORK

We have demonstrated a new approach for job execution prediction and proven good accuracy on hadoop data analytic and warehousing jobs. We can leverage this new technique for performance optimization on hadoop cluster including job scheduling and resource allocation. Resource allocation is a major issue for hadoop to optimize performance. With the size of hadoop cluster and multiple users grow larger, more and more data are placed on the cluster, a diversity and great amount of application and computation are running on it. So resource allocation and job scheduling is more critical than before in performance optimization. We can apply our approach to resource provisioning, for we can extend our approach to predict resource requirements, if we can predict execution time, we can also predict how man map tasks it will need ,how many reduce tasks it will need. One can evaluate whether there are enough resources in the cluster. Accurate execution time prediction can also be applied to identifying performance problems and lagging tasks. When large prediction error occurs, we can deduce there is a dramatic workload changes or there is a performance problem. Given that our approach accurately identified the hadoop job features which most affect the completion time, we can apply our approach to understand which aspect of hadoop job input features most influences jobs‘ execution.

This paper is just an initial research of employing artificial neural network to predict hadoop job runtimes, we are exploring the plausibility and feasibility of applying machine learning methods into job execution prediction. In our work, we only experimented on our approach with a three layer BP neural network using logarithmic sigmoid transfer function as the hidden layer‘s activation function and linear transfer function as the output layer‘s activation function. However there are a wide range of activation functions, we didn‘t analyze how different activation functions will affect the prediction results. As for artificial neural network, we only experimented on our approach with BP neural network, we didn‘t research the results of our approach on other neural networks. The main purpose of this paper is to demonstrate the validity of our approach, so we select more representative job input features which can only be obtained after the jobs‘ execution. This selection strategy works well in our experiment, we will do more further research on selecting job input feature obtainable prior the execution for implementing our approach into applications.

6. CONCLUSIONS

In this paper, we propose applying artificial neural network to hadoop job computation time estimation. We described our approach in detail and evaluated our approach from four aspects. We first demonstrated the validity of our approach, analyzed how different input features affect the prediction results by calculating the impact factor of every input feature, then discussed how different amounts of training workload influence the prediction results, finally evaluated how different feature selections influence the prediction results. We extracted hadoop job metrics from the job history logs and created a training workload and an evaluation workload. We use the training workload to train the BP neural network and simulate results using the evaluation workload. We analyzed the experiment results and discussed the field we can apply our approach to and the future work we can do further research. We strongly believe our approach can benefit a variety of hadoop in performance optimization. In future, we can do further research on implementing our approach in scheduling algorithm and resource allocation strategies. We will investigate on how and to what level our approach benefits hadoop.

7. REFERENCES

[1] M. V. Devarakonda and R. K. Iyer. Predictability of process resource usage: A measurement-based study on unix. *Software Engineering, IEEE Transactions on*, 15(12):1579–1586, 1989.

[2] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson. Statistics-driven workload modeling for the cloud. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 87–92. IEEE, 2010.

[3] N. H. Kapadia, J. A. Fortes, and C. E. Brodley. Predictive application-performance modeling in a computational grid environment. In *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, pages 47–54. IEEE, 1999.

[4] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production mapreduce cluster. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 94–103. IEEE, 2010.

[5] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky. Estimating computation times of data-intensive applications. *Distributed Systems Online, IEEE*, 5(4), 2004.

[6] K. Morton, A. Friesen, M. Balazinska, and D. Grossman. Estimating the progress of mapreduce pipelines. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 681–684. IEEE, 2010.

[7] W. Smith. Prediction services for distributed computing. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10. IEEE, 2007.

[8] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 235–244. ACM, 2011.