PTC: Performance Tuning Component for Auto-Tuning of MapReduce's Configuration

MSc Research Project
Cloud Computing

Ramesh .
x15044815

School of Computing
National College of Ireland

Supervisor: Prof. Victor Del Rosal

National College of Ireland
Project Submission Sheet – 2016/2017
School of Computing

Student Name:	Ramesh .
Student ID:	x15044815
Programme:	Cloud Computing
Year:	2016/2017
Module:	MSc Research Project
Lecturer:	Prof. Victor Del Rosal
Submission Due Date:	19/09/2017
Project Title:	PTC: Performance Tuning Component for Auto-Tuning of MapReduce's Configuration
Word Count:	5,921

Signature:	
Date:	14th September 2017

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

PTC: Performance Tuning Component for Auto-Tuning of MapReduce's Configuration

Ramesh .
x15044815
MSc Research Project in Cloud Computing

14th September 2017

Abstract

Hadoop's MapReduce framework was developed to process large datasets in a distributed environment. Performance of MapReduce job is driven by large number of settings and configuration parameters. Manual configuration of these parameters and identification of optimal values is an error prone and tedious task. Improving Performance of MapReduce framework is important in order to effectively utilize the resource. In this work, existing research methodologies has been evaluated to understand the impact of these configuration parameters and approaches to identify their optimal values. In this research, we propose Performance Tuning Component for Auto-Tuning of configuration parameter with optimal values of *io.sort.factor* and *mapreduce.job.reduces* parameters. Prediction model has been developed in order to find the optimal values of these parameters using Ridge Regression Algorithm. Prediction model has **accuracy of 91.26%** for predicting values of configuration parameters. *TeraSort* and *WordCount* have been used as benchmark for evaluation of performance for MapReduce job. Experiments results show that the proposed solution provided about **6-48%** improvement in execution time for different MapReduce jobs executed. Cost of MapReduce in cloud is associated with the time spent and resources utilized by MapReduce job. Hence, our proposed solution will not only save time, but also cost associated by reducing the execution time of the MapReduce job.

Keywords: MapReduce, Hadoop, Configuration Parameters, Ridge Regression

1 Introduction

MapReduce is distributed programming framework which was initially developed by Google, allowed processing of huge datasets in parallel and distributed mode. Apache provided open-source implementation of MapReduce framework available for commercial use. As unstructured data has increased significantly over last few years, new technical implementation has come into existence to perform analytics on these large datasets and MapReduce has been one of the popular frameworks for processing these large datasets.

MapReduce implementation has been deployed in various environmental settings such as datacentres and cloud to be more cost effective. MapReduce cluster have performance impact due to various settings, data placement and configuration parameters as explained by Rabkin and Katz (2013). Identification of these parameters requires expertise in MapReduce framework and configuration of them with optimal values is error prone task because values are driven by the type and data size of the job executed in MapReduce framework.

The research problem: *What is the impact of Auto-tuning of configuration parameters on execution time of MapReduce job using Ridge Regression ?*

MapReduce framework consists of more than 200 configuration parameters on Job-level and Task-level combined. Job-level parameters affect the performance of overall cluster and require restarting for new parameter settings to take its effect. Task-level parameters can be modified without impacting other nodes within the cluster and this provide an opportunity for improving performance of MapReduce framework using configuration parameters which will be focus of our paper.

In this work, we propose a Performance Tuning Component that auto-configures the parameter of MapReduce framework as predicted by the Ridge Regression model. Tuning the configuration parameters is accompanied with challenges which will be address in our work. Major challenge of adopting this approach is determining optimal configuration which is daunting and error prone task. Previous work has used various approaches to be as efficient as possible. We will use regression based approach to predict optimal parameters and choose the configuration parameter values which has minimal makespan. In this work we hypothesize, dynamic configuration of MapReduce parameters value predicted by Ridge Regression using Performance Tuning Component will decrease execution time significantly than default parameters.

Paper is organized as follows. Section 2, discusses previous related work done in related areas and provide contrast with our proposed solution. Section 3 presents details about the design of the performance tuning component. Section 4 will provide about configuration and implementation details of PTC tool. In Section 5 we will evaluate performance of MapReduce job using PTC tool and will compare with default configuration settings and finally, in Section 6, we concludes the paper and discuss future work.

2 Related Work

MapReduce implementation provided by Apache assumes homogeneous nodes within the cluster. Lu et al. (2015) proved with experiments that MapReduce have performance implication while performing under different configuration. In recent few years, configuration parameters has been area of interest for researches such as Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014) and Li, Zeng, Meng, Tan, Zhang, Butt and Fuller (2014) to exploit and tune parameters in order to enhance the performance of MapReduce. Dynamic configuration is another challenge which has been addressed in these approaches. In our next sub-sections, we will discuss about different approaches used in previous works for improving performance of MapReduce framework by automatic configuration of parameters.

2.1 Cost-based Approaches for optimising configuration parameters

Cost based performance modelling has been popular since early days and has been implemented across different software applications. Cost based modelling uses white box approach to predict the optimal configuration by using the system internals information. Since it uses analytical approach to find optimal parameters, it performance was better than default configuration. However as technology evolved and new software paradigm came into existence such as machine learning, new algorithms and approaches have been implemented in order to accurately predict optimal configuration parameters.

Sampling is one of the processes used in cost-based approach to predict the performance of job in terms of Execution time, CPU and Disk I/O. Song et al. (2013) used similar approach in its sampling algorithm to predict the execution time. Prediction had huge difference for grep and dedup job but the prediction was under permissible error range. It was also concluded during his experiments that the jobs were sensitive to format of input data used for processing. To improve the approach used by Song et al. (2013), Lu et al. (2015) provided feedback mechanism which enable higher computing node to process more data using configuration parameters. Another approach Wang et al. (2017) used changes in implementation strategy of MapReduce to improve shuffling and sorting process of framework which eventually improved the performance of MapReduce framework.

Few approaches uses combination of adaptive learning and mathematical model to improve the accuracy of tool proposed Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014) . Similar approaches has been contrasted by Kalavri and Vlassov (2013) which consent that in order to develop these mathematical model one require deep understanding of system internals and these models needs to updated if system configuration is changed within the cluster. Hence mathematical model would not be good choice for cluster with frequent on demand scaling.

Another approach Wasi-Ur-Rahman et al. (2016) leverage the advantage file system along with divide and conquer algorithm to improve the performance of job. Optimal configuration for MapReduce framework has been obtained using divide and conquer approach to find the local minima. Sampling data could be explored in various ways to filter relevant information, one such approach Kim et al. (2015) make use of top five parameters which greatly influence the execution time of MapReduce job. Parameter selection criteria might depend on read/write speed which is different from previous approach Wasi-Ur-Rahman et al. (2016) of total execution time. Selecting top five parameters and using analytical approach to find best combination of parameters was able to reduce upto 32% job execution time. However selection and identification of sampling parameters used in Kim et al. (2015) approach requires expertise and in-depth knowledge of Hadoop framework to filter irrelevant parameters such as *dnsname* and *datanode.dir* etc.

Sequential Sampling and random sampling are two primary approaches which have been majorly used across different methodologies Kim et al. (2015) and Song et al. (2013). Random sampling with Monte Carlo simulation is another approach Kim and Park (2015) for sampling to reduce the noise and improve prediction model developed. During the analysis Monte Carlo sampling proved to come out with best configuration parameters across Sequential sampling Kim et al. (2015) and Correlation Coefficient approach. Figure 1 summarises different approaches.

	Approach	Possible Weakness
Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014)	Combination of adaptive learning and mathematical model	Mathematical model requires deep understanding of system internals such as Java Runtime Environment and Disk I/O speed etc.
Wasi-Ur-Rahman et al. (2016)	Divide and Conquer Algorithm using system and user parameters	Requires deep understanding of user-space and system-space parameters
Kim et al. (2015)	Feature selection based on expertise	Based on experience, require framework expertise. Not good for Novice Users.
Song et al. (2013)	Propose custom sampling algorithm based on cost based model	Parameter value predicted will always be same for homogeneouse node.

Figure 1: Comparison between different approaches

2.2 Machine learning Approaches for optimising configuration parameters

Machine learning is classified into supervised and unsupervised learning. Regression is supervised learning based approach used for prediction of feature or a variable. Multiple Linear Regression has been widely used for creating learning models Yigitbasi et al. (2013). While analysing among different regression algorithms such as MLR, ANN, M5Tree and SVR etc. in Yigitbasi et al. (2013) paper, SVR outperformed the with 39% improvement in execution time than the default parameters.

KMeans clustering algorithm allow segregation of data into clusters and then try to search for optimal parameter within the cluster. One of the drawback of this approach Wu and Gokhale (2013) is that the result could provide features which could not be mapped into MapReduce framework, hence it requires to modified to correctly map to range to values permissible within the MapReduce framework. Other approaches Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014) follow same methodology to extract data from different sources and then develop learning model. However in our approach we have filtered sample data hence we don't require analyser component. Testing against different types of job is import aspect is another aspect which has been tested in Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014) and Wu and Gokhale (2013) approaches.

Apart from linear regression, tree based regression has also been used Chen et al. (2015) to classify and predict the optimal configuration parameters for MapReduce framework. Hill climbing is another approach which could be used for search for best possible parameters. However one might stuck to local minima, if data has more than one local minima. Using tree based regression and combination of random sampling along with Hill climbing algorithm reducecd the execution time around 20% of execution time. Hill climbing algorithm has been popular in Chen et al. (2015), Li, Zeng, Meng, Tan, Zhang, Butt and Fuller (2014) approaches to search for optimal parameters. Along with Hill climbing algorithm Ding et al. (2015) approach uses tuning and resource allocation techniques to improve the performance of the MapReduce framework. Elastic container has been used for resource allocation mechanism which allows dynamic scalling of resources.

It has been majorly divided into two process just like Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014), however in this approach second step is used for resource allocation mechanism to effectively distribute these resources. Another distinguish feature of this approach is that it make use of real-time statistics Ren et al. (2012) which is collected from MapReduce framework and used for predicting tuning parameters. Implementation of Li, Zeng, Meng, Tan, Zhang, Butt and Fuller (2014) uses configuration parameters of YARN to improve the performance of MapReduce framework with heterogeneous nodes. Parameters have been broadly classified into memory and CPU which are optimised using HillClimbing search for desirable configuration.

Cheng et al. (2017) approach complements Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014) self adaptive tuning of MapReduce configuration parameters. However Cheng et al. (2017) and Zhang et al. (2015) approach considers both homogeneous and heterogeneous clusters for parameter optimisation. Experiments demonstrated that the approach is effective on both homogeneous and heterogeneous clusters.

MapReduce framework performance significantly degrades in heterogeneous cluster. AROMA Lama and Zhou (2012) address this issue with automatic allocation of resources along with configuration parameters. SVM machine learning algorithm similar to Wasi-Ur-Rahman et al. (2016) approach has been used to construct model for automatic job provisioning. Data collection technique makes use of historical logs generated by MapReduce framework similar to Yigitbasi et al. (2013) approach. However, it uses *dstat* (third party tool) to extract relevant information such as execution time, data size and resource allocation etc. while other approaches uses custom implementation for data extraction.

MapReduce job executed creates log entries about the execution of job and relates information such as execution time, no of mapper tasks and total bytes transferred etc. Data about the jobs executed within the MapReduce framework could be exacted from different sources Yigitbasi et al. (2013). Data extracted from log entries (Historical data) is useful when we don't want to sample the data or cluster is already into production. Historical data about Job performance has been prominent source in Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014) approach for modelling. Historical data might not be useful if cluster used performs similar task multiple times. Sampling method allow to execute task with variable file size and parameters.

2.3 Need for Auto-Tuning of configuration parameters

MapReduce framework is widespread adoption of distributed data processing during past few years. But since MapReduce consists of large number of configuration parameters, parameter selection is quite a daunting task for a novice. Moreover, configuration parameters might have impact on other parameter as well which night either significantly improve or degrade the effectiveness of related configuration parameter as demonstrated in Lu et al. (2015) and Rabkin and Katz (2013) approaches. For dynamic tuning of configuration, Yigitbasi et al. (2013) paper provides machine learning approach to get optimal parameters for MapReduce job. It was found that machine learning approach has better results in predicting these parameters.

Experiments performed by Cheng et al. (2017) proved significant improvement in MapReduce framework performance by tuning the task-level parameters. However, identification of optimal parameter is error prone task and moreover manual tuning of these parameters over large cluster is time consuming process and impractical. Hence, we require to dynamically configuring these parameter to values for which MapReduce framework could provide optimal performance. Wasi-Ur-Rahman et al. (2016) also agrees to the fact that configuration parameters of MapReduce framework have different optimal values under different HPC cluster. Hence, set of values for configuration parameters cannot be generalized to provide optimal parameters and we need to find the optimal values of configuration for each cluster deployed. Table 2 provide summary of closest approaches to our proposed solution.

Auto-Tuning Approaches	Similarity	Difference
By Babu (2010)	Similar configuration parameter has been chosen to get data about MapReduce job	Approach uses Database Query Optimization technique, while PTC uses Regression based prediction model.
By Wu (2015)	Search Algorithm to search for optimal parameters has been inspired by this research	Hadoop version used in Wu (2015) experiment is too old (1.0.0), most of the parameters used in research are now obsolete (in 2.7.2).
By Bei et al. (2016)	Similar approach used in PTC for data collection	Random Forest Algorithm has been used for creation of Prediction model while PTC uses Ridge Regression Algorithm for prediction.

Figure 2: Similarity and Differences between PTC and previous works in same direction

3 Methodology

Configuration parameters provide numerous numbers of parameters which could be configured by system administrator. However finding effective value of them could be error prone task which might negatively affect the performance of MapReduce job. A performance tuning component has been proposed, to efficiently find optimal values of these parameters and dynamically configure these parameters. Section 1 will describe details about environment details and tools required to implement solution. Section 2 will provide glimpse of Overall Architecture of Performance tuning component. Furthermore, pseudo algorithm to find optimal parameters is proposed. Section 3,4 would provide details of software diagrams such as class integration diagram and sequence diagram respectively.

3.1 Environment Setup

In order to measure the performance our proposed solution we have constructed MapReduce Cluster Environment as per the steps defined in configuration manual available as a part of Appendix A

Cluster consists of 3 Nodes. One of them is *MasterNode* running *NameNode* and *ResourceManager* processes and other two are slaves running *DataNode* and *NodeManager* processes. All DataNode servers have the same hardware configuration, while NameNode has been better RAM of 2048. Each Node has 1-Core CPU processor running *Ubuntu*

16.04 LTS 64-bit Linux. RAM of each DataNode is 1 GB. NameNode is given higer memory due to the reason that all the job are executed on NameNode and requires seamless interaction. Cluster is deployed with *Java Development Kit 1.8.0_131* and *Hadoop 2.7.2* . Hadoop configuration parameter has been kept at their default configuration settings.

Node Type	Machine Name	IP Address	CPU Processor Count	Memory(MB)
Master Node	hadoopmaster	192.168.0.12	1	2048
Data Node	hadoopslave1	192.168.0.13	1	1024
Data Node	hadoopslave2	192.168.0.14	1	1024

Figure 3: Cluster Configuration

Moreover, below tools has been used in order to develop and evaluate the solution:

1. Oracle Virtual Box 5.1.26
2. Spyder (Python 3.6)
3. IBM SPSS statistics
4. dstat (Resource Montioring Tool)

3.2 Proposed Performance Tuning Component Architecture

The proposed solution within this research works on finding optimal configuration parameter for MapReduce framework and dynamically update them to the MapReduce framework. Overall Architecture of Performance Tuning Component is represented in Figure 4 . Performance Tuning Component is configured on NameNode. PTC is configured to be called from MapReduce Job Tracker. PTC is subdivided into two components: Analyser and Tuner. We will discuss more about subcomponents of PTC in upcoming sections.

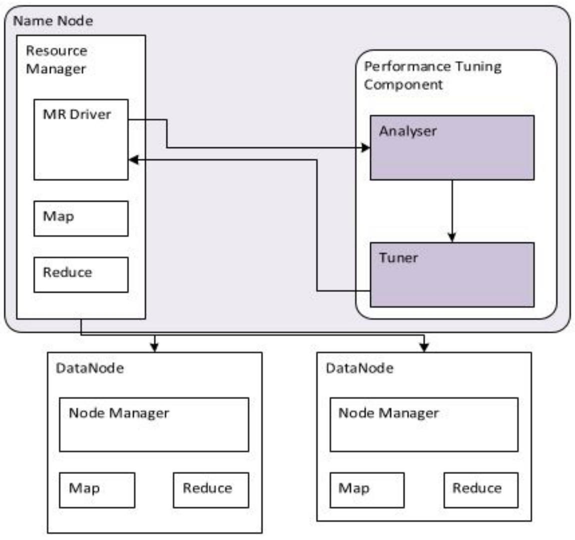


Figure 4: Overall Architecture of Performance Tuning Component (PTC)

3.2.1 Analyser

Analyser is subcomponent of PTC and is responsible for finding optimal parameter values. Analyser uses Ridge Regression algorithm to predict the values of configuration parameters. Aim of Analyser is to find the parameter values which results into least execution time of MapReduce job. Analyser is prediction model is developed from Training dataset as shown is figure 5.

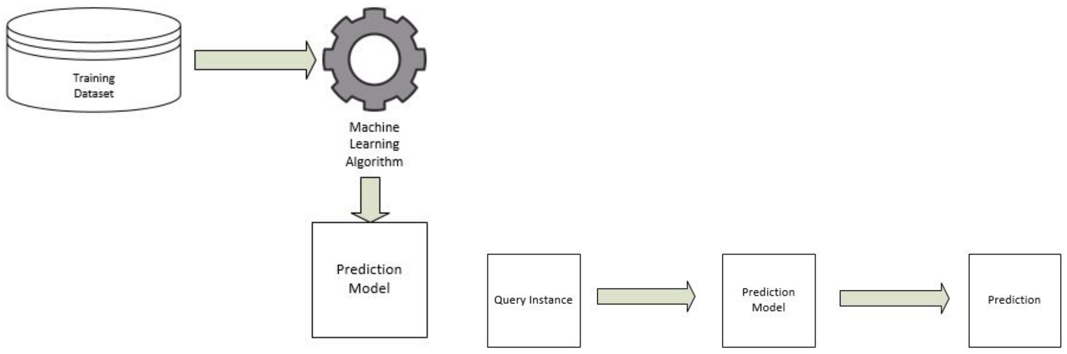


Figure 5: Steps of developing prediction model

Previous methodologies emphasised on Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014) , Li, Zeng, Meng, Tan, Zhang, Butt and Fuller (2014) and Babu (2010) uses different parameter in order to optimize the MapReduce job. Similar to Babu (2010) approach, in our research we have used two configuration parameters : *io.sort.factor* and *mapreduce.job.reduces* which are dependent on inputDataSize of MapReduce job. Default Configuration value of MapReduce framework has been described in Figure 6 .

Parameter Name	Description	Default Value	Values Considered
mapreduce.task.io.sort.factor	The number of streams to be merged at once while sorting files.	10	[10,500]
mapreduce.job.reduces	The default number of reducer tasks per job.	1	[2,300]

Figure 6: Job Configuration parameters in Hadoop

Information about the MapReduce job is gather using sampling which uses different combination of MapReduce parameters in order to train our prediction model using bash script. Secondly we train our prediction model, as per the standards 80 percent of sampling data is used to train the prediction model and accuracy of the model is tested against the 20 percent.

Once we have our prediction model capable of predicting the configuration parameter values, we can use search algorithm to find the configuration values with least execution time. Search algorithm is inspired by the Wu (2015) approach to find the optimal values. Pseudo Alogirthm has described in Figure 7 .

In this research, we have selected three parameters for regression i.e. the input data size (denoted as *is*), the number of reducers (denotes as *r*) and the io.sort factor (denoted as *sf*). Regression equation is represented as *et=PredictionModel(is, sf, r)*, where

```
m_time=intMax;
        for(sf in 10..m_sortfactor)
                for(r in  2..m_reduce)
                        if(m_time>PredictionModel(is,sf,r))
                                m_time=PredictionModel(is,sf,r);
                                O_sf=sf;
                                O_r=r;
                        end if;
                end for;
        end for;
```

Figure 7: Pseudo code for searching optimal parameters

et represents the execution time. Based on the predicted value we will choose the parameter values of io.sort factor and number of reducers which are able to provide least job execution time.

Algorithm uses range of values from *[1,m_sortfactor]* and *[1,m_reduce]* in order to find optimal parameters *O_sf* and *O_r* (where *m_sortfactor* and *m_reduce* are maximum values of sort factor and number of reducers respectively). In line 5, *m_time* is compares with predicted values against different parameter values and if we found parameter configuration resulting into less execution time, it consequently becomes our optimal configuration which configuration values as *O_sf* and *O_r*.

3.2.2 Tuner

Tuner, a subcomponent of PTC updates the optimal parameter provided by the Analyser to the MapReduce framework. All the details about the parameter and their value are provided by the Analyser to the Tuner. Once the parameters have been updated, Job Tracker takes the ownership of the job and distributes the MapReduce job across the cluster with the configured parameters.

3.3 Class Interaction Diagram of Proposed Solution

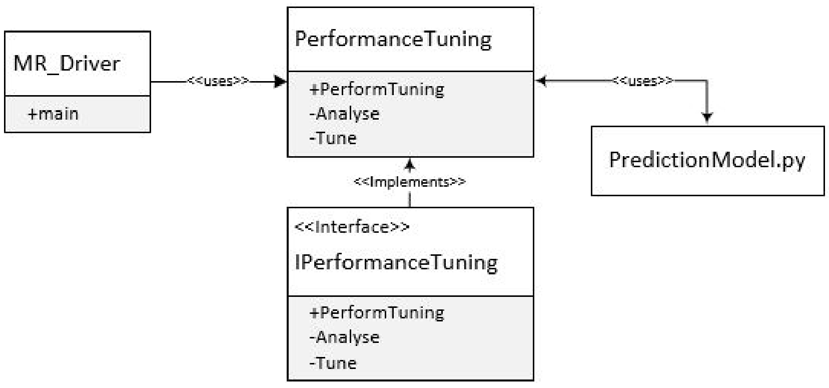


Figure 8: Class Diagram with dependencies for Proposed Solution

MapReduce framework consists of Driver, Mapper and Reducer classes to executed

MapReduce job. MR_Driver (MapReduce Driver) class is responsible for initial setup of MapReduce job such as Data Input Source, configure parameters and provide details about Mapper and Reducer Classes. MapReduce Driver will be interacting classes in order effectively call PTC component. IPerformTuningComponent provide skeleton for implementing classes. Analyse method is responsible for analysing the optimal parameter by executing PredictionModel.py script. PredictionModel.py is dependent module which returns optimal parameter values after performing Ridge regression as shown in Figure 8 . Tune method would also be responsible to update these parameters to the MapReduce framework.

3.4 Sequence Diagram with Data Flow of the Proposed Solution

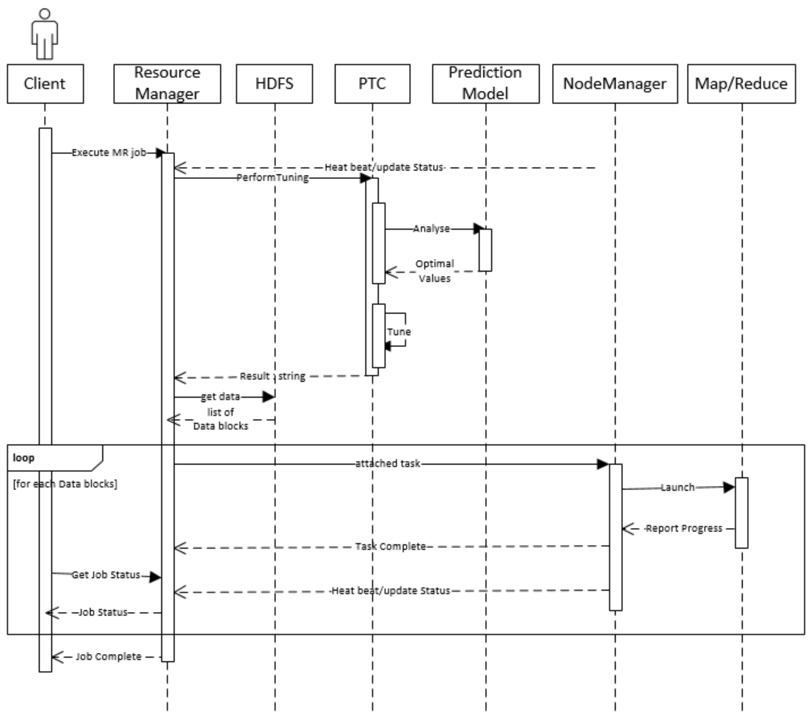


Figure 9: Sequence Diagram of Proposed Solution

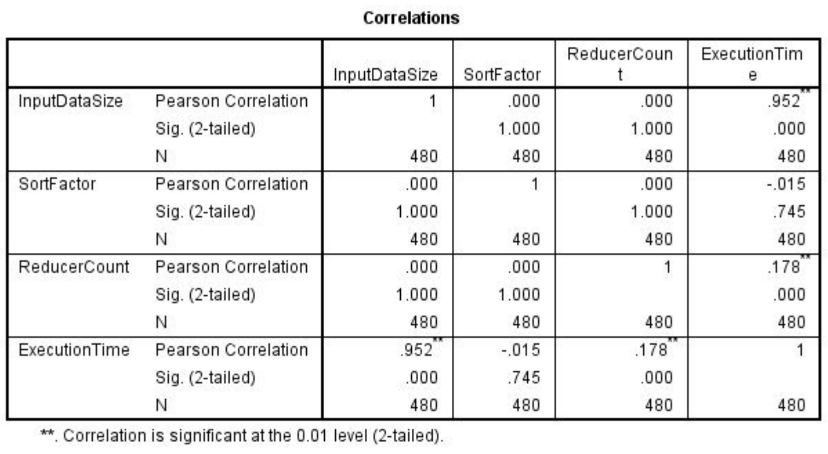
The main aim of this diagram is to capture the sequence of events and interaction of proposed solution with MapReduce framework. PTC component is installed and configured on Namenode. When a client requests for job execution, Resource Manager perform initial setup for job. Our poposed solution would be called during this phase to help in configuration with optimal parameters values. Hence, Resource Manager calls our component to perform tuning. PTC internally calls Analyser to predict paramaters values and tuner to update these value to MapReduce framework. After parameters has been updated, MapReduce traditional workflow continues to fetch the data from HDFS and execute the MapReduce job on Datanode with the parameters configured using PTC component as shown in figure 9.

4 Implementation

In this section we would be discuss about the environment configuration used for research. We would also discuss about the Data Collection and implementation used for creation of prediction model. Furthermore, the implementation details about the components of PTC would be explained at the end of this section.

4.1 DataCollection

We have collected **480 samples** of the execution information as the training set. The input dataset has been generated using *teragen* program provided by the Hadoop framework. It was ensured that hadoop is configured with default parameters and during the sample collection no other program was executed in background apart from *dfs* and *yarn* services. Moreover, updates were turned off to ensure all resources are fairly avaiable for MapReduce job only.



Correlations

		InputDataSize	SortFactor	ReducerCount	ExecutionTime
InputDataSize	Pearson Correlation	1	.000	.000	.952**
	Sig. (2-tailed)		1.000	1.000	.000
	N	480	480	480	480
SortFactor	Pearson Correlation	.000	1	.000	-.015
	Sig. (2-tailed)	1.000		1.000	.745
	N	480	480	480	480
ReducerCount	Pearson Correlation	.000	.000	1	.178**
	Sig. (2-tailed)	1.000	1.000		.000
	N	480	480	480	480
ExecutionTime	Pearson Correlation	.952**	-.015	.178**	1
	Sig. (2-tailed)	.000	.745	.000	
	N	480	480	480	480

**. Correlation is significant at the 0.01 level (2-tailed).

Figure 10: Correlation Matrix provides details about the correlation across different features

Bash script has been created to get the details about the MapReduce job under different parameter settings. All the results are collated into the text file and after each job execution, its output file is deleted to keep hdfs clean. The size of the input data varies from 5 MB to 1GB. The regression model has been developed in Python using this training dataset.
As shown in Figure 10 Execution Time is highly dependent on InputDataSize and ReducerCount. However as per the results, SortFactor doesnot influence ExecutionTime. To validate this we will dig deeper into the data. As shown in Figure 11 over certain value of number of reducer tasks, changing io.sort factor has least impact on job execution time. Therefore we get non-significant value between Execution Time and SortFactor in Correlation matrix. It also proves that the parameter *io.sort.factor* and *mapreduce.job.reduces* are inter-related thus resulting into multicollinearity. *Multicollinearity* is another reason for choosing Ridge Regression as it supports multicollinearity features.
From Figure 12 it could be concluded that these features has great impact on jobs with larger data size and since MapReduce framework is mostly used for BigData analytics it provide us opportunity to optimize these parameters with minimal execution time.

It allow supports our hypothesis, to find optimal parameter values in order to reduce execution time.

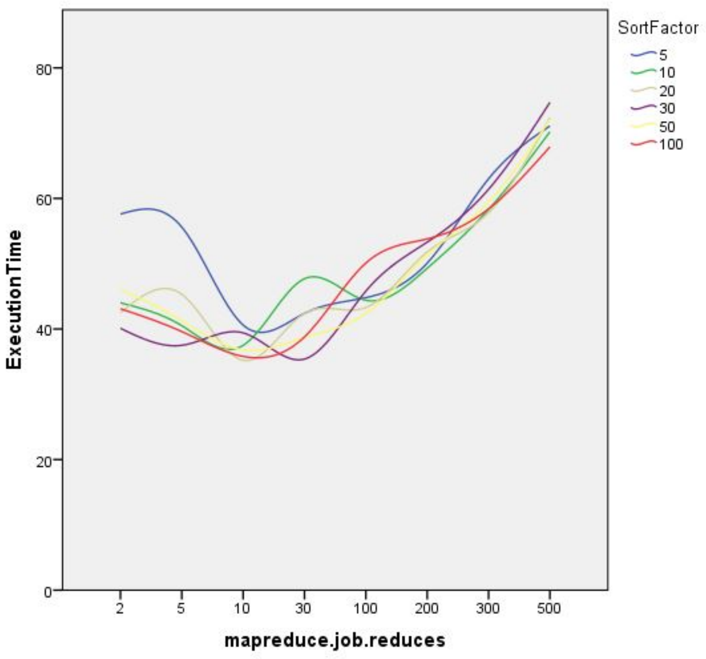


Figure 11: Visualization of MapReduce job for impact of io.sort.factor on Reducer Tasks count.

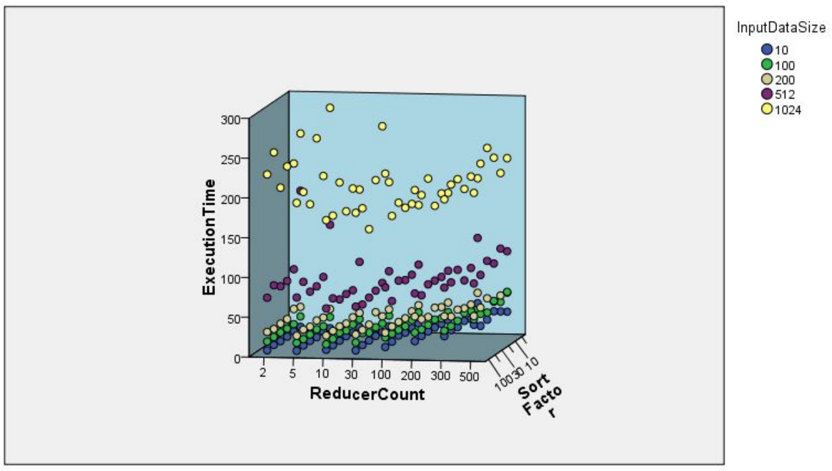


Figure 12: Scatter Plot for io.sort.factor and mapreduce.job.reduces under different data size.

4.2 Implementation of Performance Tuning Component

Performance Tuning Component has been written in Java programming language. Regression Model has been developed in Python programming language. Python provides open source libraries which helps in building and testing the Ridge Regression Model. Regression Model act as a Analyser of PTC, which is responsible for predicting the optimal values of configuration parameters. Java component act as Tuner of PTC, which is responsible for fetching the results from Analyser and update the parameters to MapReduce job execution environment.

4.2.1 Analyser

Training Model
Analyser uses Python programming language to train the prediction model. Sampling data set has been divided into two subsets. First subset is 80% of total sampling data which is used to train the model. Rest 20% of data set would be used to check the accuracy of the model, which has been performed using below code :

```
...
X_train , X_test , Y_train , Y_test = train_test_split(X, Y,
                                        test_size=0.20, random_state=27)
lm = linear_model.Ridge(alpha=1.0)
lm.fit(X_train,Y_train)
...
```

Once that the data has been splitted, 80% of the data could be used to train the model. *X_Train* and *Y_Train* have been used as training data in our case. In our next section we will try to find best value of *alpha*.

Finding Best fit
Ridge Regression Model provide straight line which best represents the data points in the training set. Position of straight line is driven by tuning knobs (value of *alpha* in our proposed model). We have used *GridSearchCV* as cross validation utility which predict the best possible fit over the training set. Once we have identified the best fit, we can use the value of parameters provided by *GridSearchCV* results and test the model for its accuracy. *GridSearchCV* provided ***best_score*** **as 94.21%** under alpha=1.0 and thats the reason for choosing value of *alpha=1.0* during initialization of Ridge Regression.

Ridge Regression has been chosen among various regression algorithms because sampling data provide evidence that the features are multicollinearity. Ridge Regression provides coefficient estimates as non-zero while in other approaches such as LASSO one of the coefficients will be equal to zero. Another reason for not choosing LASSO is that it recommended for strong correlated predictors. Since, our predictors are correlated after large values of reducers hence Ridge Regression perfectly aligns with our requirements for prediction.

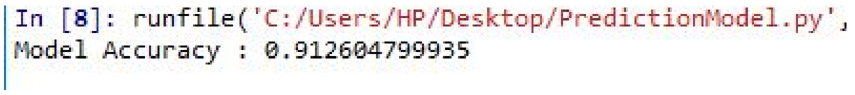
Testing Model
Accuracy is tested against the remaining 20% of data from the original dataset. Accuracy provides an idea about how accurately model is able to predict the value of test data set. Accuracy of the model is tested using below code:.

Once the model is efficient enough to predict the optimal parameters, We created a pickle file which would be used to predict the parameters.

Pickle file will remove the necessity to train the model again and again. It will also allow prediction over the scaled environment, since it would only requires changing of trained pickle file.

```
        ...
        accuracyRidge=lm.score(X_test, Y_test)
        print("Model Accuracy :",accuracyRidge)
        ...

```



```
In [8]: runfile('C:/Users/HP/Desktop/PredictionModel.py',
Model Accuracy : 0.912604799935
```

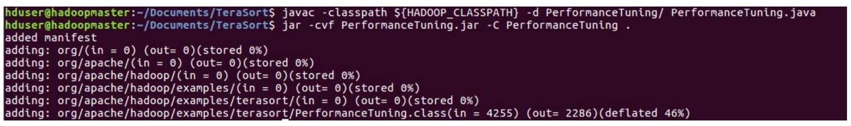
Figure 13: Accuracy of Prediction Model (91.26 %)

```
...
with open('linearregression.pickle','wb') as f:
        pickle.dump(lm,f)
...
```

4.2.2 Tuner

Tuner subcomponent of PTC is developed to update the optimal parameters into the MapReduce framework. Tuner component is installed and configured under hadoop libraries. It will allow every MapReduce job to access the PTC component whenever required.

Tuner fetches the result from Analyser component and parses the result into the configuration parameter values. Once the result has been parsed, these parameter values are assigned to Job class of MapReduce framework for performing subsequent tasks. Tuner component is compiled using *javac* compiler and jar is created as shown in figure.

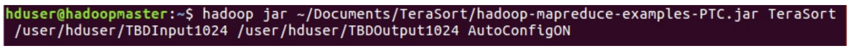


```
hduser@hadoopmaster:~/Documents/TeraSort$ javac -classpath ${HADOOP_CLASSPATH} -d PerformanceTuning/ PerformanceTuning.java
hduser@hadoopmaster:~/Documents/TeraSort$ jar -cvf PerformanceTuning.jar -C PerformanceTuning .
added manifest
adding: org/(in = 0) (out= 0)(stored 0%)
adding: org/apache/(in = 0) (out= 0)(stored 0%)
adding: org/apache/hadoop/(in = 0) (out= 0)(stored 0%)
adding: org/apache/hadoop/examples/(in = 0) (out= 0)(stored 0%)
adding: org/apache/hadoop/examples/terasort/(in = 0) (out= 0)(stored 0%)
adding: org/apache/hadoop/examples/terasort/PerformanceTuning.class(in = 4255) (out= 2286)(deflated 46%)
```

Jar file is configured into hadoop environment and placed under hadoop installation director i.e. $HADOOP_DIR/share/hadoop/mapreduce/lib . It allows every MapReduce job to have accessibility over PTC component and PTC component could be integrated with any MapReduce job whenever required.

4.3 Execution of Performance Tuning Component

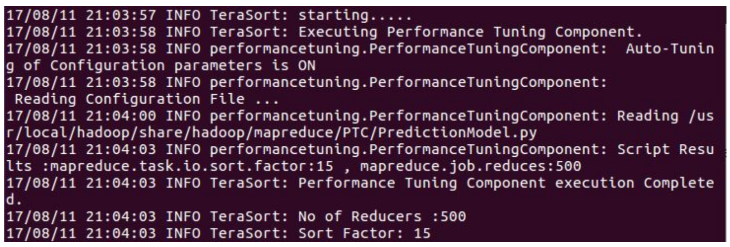
PTC component has been designed hassle freeway in order to effective use the PTC in user friendly manner. PTC component simply uses one tag AutoConfigON to use the component features of job optimization and AutoConfigOFF to execute job with default configuration settings. PTC component can be turned ON as below for TeraSort job :



```
hduser@hadoopmaster:~$ hadoop jar ~/Documents/TeraSort/hadoop-mapreduce-examples-PTC.jar TeraSort
 /user/hduser/TBDInput1024 /user/hduser/TBDOutput1024 AutoConfigON
```

PTC component also provide logs for easy debugging and allowing user to track the steps performed by the PTC component as shown below :

Also, MapReduce could be executed without the optimization under default settings using *AutoConfigOFF*.



```
17/08/11 21:03:57 INFO TeraSort: starting.....
17/08/11 21:03:58 INFO TeraSort: Executing Performance Tuning Component.
17/08/11 21:03:58 INFO performancetuning.PerformanceTuningComponent:  Auto-Tunin
g of Configuration parameters is ON
17/08/11 21:03:58 INFO performancetuning.PerformanceTuningComponent:
 Reading Configuration File ...
17/08/11 21:04:00 INFO performancetuning.PerformanceTuningComponent: Reading /us
r/local/hadoop/share/hadoop/mapreduce/PTC/PredictionModel.py
17/08/11 21:04:03 INFO performancetuning.PerformanceTuningComponent: Script Resu
lts :mapreduce.task.io.sort.factor:15 , mapreduce.job.reduces:500
17/08/11 21:04:03 INFO TeraSort: Performance Tuning Component execution Complete
d.
17/08/11 21:04:03 INFO TeraSort: No of Reducers :500
17/08/11 21:04:03 INFO TeraSort: Sort Factor: 15
```

5 Evaluation

TeraSort and *WordCount* are two major benchmarks provided by Hadoop framework, in order to evaluate the performance of MapReduce job. Performance of proposed solution has been evaluated against the traditional system with default configuration parameters. *TeraSort* program measures the amount of taken to sort large data set. *TeraGen* is another program to generate large data set. WordCount counts the words over the large data set and both of these benchmarks uses both Mapper and Reducer phases of MapReduce framework. In this section we will evaluate the performance of these two benchmarks after integration with PTC component.

5.1 Evaluation of TeraSort performance

Performance of TeraSort job has been evaluated against different data sizes ranging from 512 MB to 1.25 GB. Table 14 presents the results against different data size and under PTC and default configuration settings. The results has been collated across three Tera-

	PTC AutoConfigOn	Default
512 MB	0:01:32	0:02:01
1 GB	0:03:44	0:04:49
1.25 GB	0:04:58	0:07:20

Figure 14: Camparison of TeraSort job's execution time under PTC AutoConfigON and Default configuration settings

Sort job execution i.e. for 512 MB we ran three execution under default settings and calculated the average execution time and similarly the process was repeated for different data size and configuration settings. Main idea behind doing average is to get *un-biased* result. Figure 15 provide visualization of the results obtained. We can conclude from the figure that there is significant reduction of execution time under PTC AutoConfigON configuration. Another interpretation of the result could be increase in CPU utilization which has been proved as shown in figure 16.

5.2 Evaluation of WordCount performance

Similar to TeraSort performance evaluation, we have collected the execution time of WorCount job over three execution and calculated the average execution time of the job for same settings. Results have been compiled under Table 17. Figure 18 and 19 provide visualization of the results obtained. In our next section 5.3 we will discuss the results in detail.

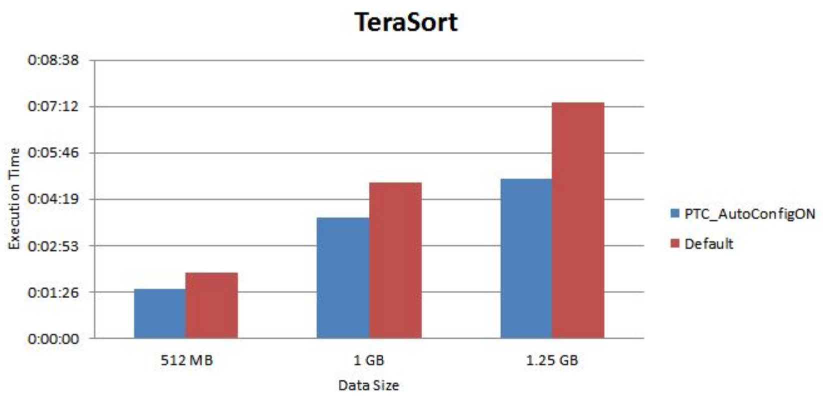


Figure 15: Visualization of TeraSort performance under different Data Input.

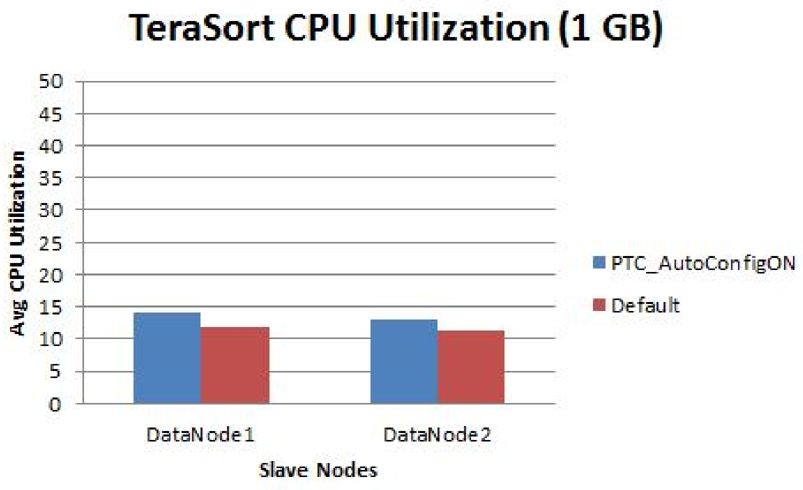


Figure 16: CPU Utilization of TeraSort

	PTC_AutoConfigON	Default
512 MB	0:02:19	0:02:28
1 GB	0:05:49	0:06:24
1.25 GB	0:07:04	0:07:59

Figure 17: Camparison of WordCount job's execution time under PTC AutoConfigON and Default configuration settings

5.3 Discussion

As seen in TeraSort experiment, PTC was able to reduce execution time up to 48 percent (for 1.25 GB). Also, it could be interpreted from the experiments that PTC has less impact for job with smaller data set (31.5% improvement) as compare to larger data set (48% improvement).

Under WordCount experiment, PTC was again able to reduce execution time. However, it was not as significant as that of the TeraSort experiment. WordCount experiment provided reduction of up to 12.9% of execution time in job with large data set (1.25 GB) and around 6.4% reduction of execution time for smaller data set (512 MB) .

Based on the trends of the graphs in figure 18 and 15 it could be concluded that

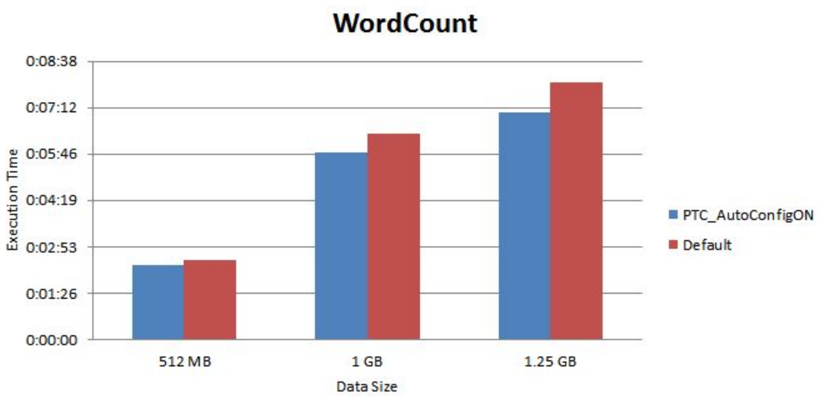


Figure 18: Visualization of WordCount performance under different Data Input.

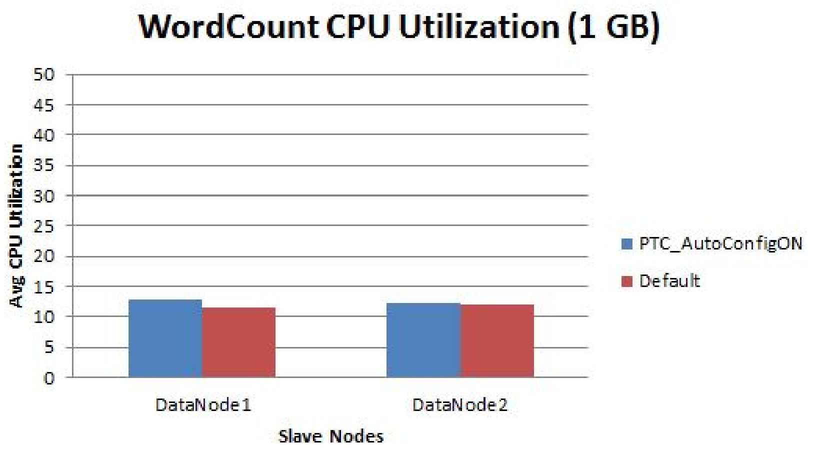


Figure 19: CPU Utilization of WordCount

PTC doesnot have significant improvement on smaller data sets, although MapReduce framework is majorly used for large datasets. Moreover the CPU ultization over proposed solution is higher than that of traditional system. Results could be summarized as below:

1. PTC reduces the execution time of MapReduce job for about **6-48%**.
2. PTC optimization over smaller data set is less significant.
3. PTC improves utilization of CPU resource.

6 Conclusion and Future Work

In this paper we proposed a Performance Tuning Component to improve the performance of MapReduce job using Auto-Tuning of configuration parameters. Results explained in previous section provide evidences that the proposed solution has significant improvement in execution time of MapReduce job. Performance of MapReduce job has been evaluated again the traditional Hadoop system with default parameters.

To evaluate the performance of Hadoop system with proposed solution, two benchmark *TeraSort* and *WordCount* were integrated with Performance Tuning Component. The evaluation section provides elaborated results about the performance of Performance

Tuning Component as compared to traditional Hadoop system.

We can conclude from the results that there is significant improvement in execution time of MapReduce job by implementing Ridge Regression using Performance Tuning Component.

However, the performance of Performance Tuning Component could be improved by using different Machine Learning algorithm apart from the Ridge Regression algorithm. Our work complements the previous work Chen et al. (2015), Li, Zhuang, Lu, Sun, Zhou, Dai and Zhou (2014), Wasi-Ur-Rahman et al. (2016) and Bei et al. (2016) done in similar direction in order to Auto-Tune configuration parameters. One of the limitations of the Performance Tuning Component is that under smaller data set we don't find significant improvements. Although MapReduce framework is mostly used for large dataset but integrating Performance Tuning Computing for processing small data set would not provide significant improvement and would add nuisance to MapReduce job. Hence we would recommend integrating Performance Tuning Component with the MapReduce processing large data sets.

Acknowledgements

I extend my gratitude to our program director Dr. Horacio Gonzalez-Velez for providing all necessary resources and support to accomplish this dissertation work "PTC: Performance Tuning Component for Auto-Tuning of MapReduce's Configuration" for Masters in Cloud Computing at National College of Ireland.

My sincere thanks to my supervisor and guide Prof. Victor Del Rosal for his guidance, suggestions, efforts and discussion during the course work. Every meeting and timely advices helped me to remain focused and motivated during the course work.

Last but not the least, I would like to thank my family and friends for their constant support and co-operation throughout the completion of my master's degree in Cloud Computing.

References

Babu, S. (2010). Towards automatic optimization of mapreduce programs, *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, ACM, New York, NY, USA, pp. 137–142. doi :10.1145/1807128.1807150.
URL: *http://ezproxy.ncirl.ie:3776/10.1145/1807128.1807150*

Bei, Z., Yu, Z., Zhang, H., Xiong, W., Xu, C., Eeckhout, L. and Feng, S. (2016). Rfhoc: A random-forest approach to auto-tuning hadoop's configuration, *IEEE Transactions on Parallel and Distributed Systems* **27**(5): 1470–1483. doi :10.1109/TPDS.2015.2449299.

Chen, C. O., Zhuo, Y. Q., Yeh, C. C., Lin, C. M. and Liao, S. W. (2015). Machine learning-based configuration parameter tuning on hadoop system, *2015 IEEE International Congress on Big Data*, New York, NY, USA, pp. 386–392. doi :10.1109/BigDataCongress.2015.64.

Cheng, D., Rao, J., Guo, Y., Jiang, C. and Zhou, X. (2017). Improving performance of heterogeneous mapreduce clusters with adaptive task tuning, *IEEE Transactions on Parallel and Distributed Systems* **28**(3): 774–786. doi :10.1109/TPDS.2016.2594765.

Ding, X., Liu, Y. and Qian, D. (2015). Jellyfish: Online performance tuning with adaptive configuration and elastic container in hadoop yarn, *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, Melbourne, VIC, Australia, pp. 831–836. doi :10.1109/ICPADS.2015.112.

Kalavri, V. and Vlassov, V. (2013). Mapreduce: Limitations, optimizations and open issues, *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, Melbourne, VIC, Australia, pp. 1031–1038. doi :10.1109/TrustCom.2013.126.

Kim, J., Kumar, T. K. A., George, K. M. and Park, N. (2015). Performance evaluation and tuning for mapreduce computing in hadoop distributed file system, *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, Cambridge, UK, pp. 62–68. doi :10.1109/INDIN.2015.7281711.

Kim, J. and Park, N. (2015). Identification of the optimal hadoop configuration parameters set for mapreduce computing, *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*, Okayama, Japan, pp. 108–112. doi :10.1109/ACIT-CSI.2015.27.

Lama, P. and Zhou, X. (2012). Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud, *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, ACM, New York, NY, USA, pp. 63–72. doi :10.1145/2371536.2371547.
URL: *http://ezproxy.ncirl.ie:3776/10.1145/2371536.2371547*

Li, C., Zhuang, H., Lu, K., Sun, M., Zhou, J., Dai, D. and Zhou, X. (2014). An adaptive auto-configuration tool for hadoop, *2014 19th International Conference on Engineering of Complex Computer Systems*, Tianjin, China, pp. 69–72. doi :10.1109/ICECCS.2014.17.

Li, M., Zeng, L., Meng, S., Tan, J., Zhang, L., Butt, A. R. and Fuller, N. (2014). Mronline: Mapreduce online performance tuning, *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, ACM, New York, NY, USA, pp. 165–176. doi :10.1145/2600212.2600229.
URL: *http://ezproxy.ncirl.ie:3776/10.1145/2600212.2600229*

Lu, Q., Zhu, L., Zhang, H., Wu, D., Li, Z. and Xu, X. (2015). Mapreduce job optimization: A mapping study, *2015 International Conference on Cloud Computing and Big Data (CCBD)*, Shanghai, China, pp. 81–88. doi :10.1109/CCBD.2015.33.

Rabkin, A. and Katz, R. H. (2013). How hadoop clusters break, *IEEE Software* **30**(4): 88–94. doi :10.1109/MS.2012.73.

Ren, Z., Liu, Z., Xu, X., Wan, J., Shi, W. and Zhou, M. (2012). Waxelephant: A realistic hadoop simulator for parameters tuning and scalability analysis, *2012 Seventh ChinaGrid Annual Conference*, Beijing, China, pp. 9–16. doi :10.1109/ChinaGrid.2012.25.

Song, G., Meng, Z., Huet, F., Magoules, F., Yu, L. and Lin, X. (2013). A hadoop mapreduce performance prediction method, *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, Zhangjiajie, China, pp. 820–825. doi :10.1109/HPCC.and.EUC.2013.118.

Wang, J., Qiu, M., Guo, B. and Zong, Z. (2017). Phase reconfigurable shuffle optimization for hadoop mapreduce, *IEEE Transactions on Cloud Computing* **PP**(99): 1–1. doi :10.1109/TCC.2015.2459707.

Wasi-Ur-Rahman, M., Islam, N. S., Lu, X., Shankar, D. and Panda, D. K. (2016). Mr-advisor: A comprehensive tuning tool for advising hpc users to accelerate mapreduce applications on supercomputers, *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Los Angeles, CA, USA, pp. 198–205. doi :10.1109/SBAC-PAD.2016.33.

Wu, D. and Gokhale, A. (2013). A self-tuning system based on application profiling and performance analysis for optimizing hadoop mapreduce cluster configuration, *20th Annual International Conference on High Performance Computing*, Bangalore, India, pp. 89–98. doi :10.1109/HiPC.2013.6799133.

Wu, X. (2015). A mapreduce optimization method on hadoop cluster, *2015 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration*, Wuhan, China, pp. 18–21. doi :10.1109/I-CIICII.2015.92.

Yigitbasi, N., Willke, T. L., Liao, G. and Epema, D. (2013). Towards machine learning-based auto-tuning of mapreduce, *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, San Francisco, CA, USA, pp. 11–20. doi :10.1109/MASCOTS.2013.9.

Zhang, Z., Cherkasova, L. and Loo, B. T. (2015). Exploiting cloud heterogeneity to optimize performance and cost of mapreduce processing, *SIGMETRICS Perform. Eval. Rev.* **42**(4): 38–50. doi :10.1145/2788402.2788409.
URL: *http://ezproxy.ncirl.ie:3776/10.1145/2788402.2788409*

A Appendix Section

A.1 Configuration Manual

A.1.1 Hadoop Cluster setup using Virtual Machine

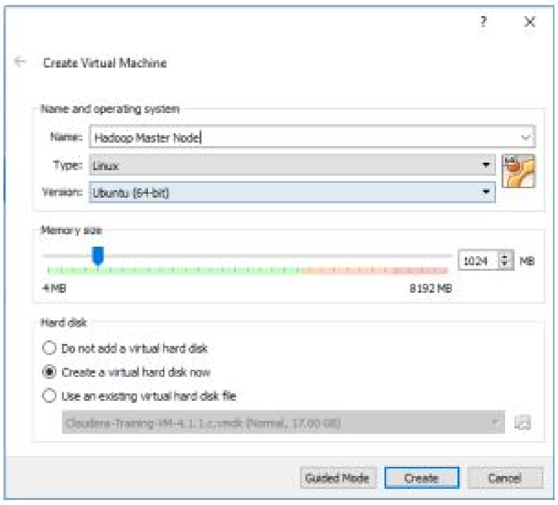
Prerequisites:

- Oracle VirtualBox (Open Source Software), available at `https://www.virtualbox.org/wiki/Downloads`
- Ubuntu 16.04 LTS iso file, available at `http://releases.ubuntu.com/16.04/ubuntu-16.04.3-desktop-amd64.iso`
- At least 150 GB free Hard Disk or more
- At least 6 GB of RAM or more

All above requirement has to be fullfilled in order to setup Hadoop Cluster using Virtual Machine.

Creating Virtual Machine

- Open VirtualBox application, click on "New".
- Create a VM by providing appropriate Name and resources.



Create Virtual Machine

Name and operating system

Name: Hadoop Master Node

Type: Linux

Version: Ubuntu (64-bit)

Memory size

1024 MB

4 MB

8192 MB

Hard disk

( ) Do not add a virtual hard disk

(●) Create a virtual hard disk now

( ) Use an existing virtual hard disk file

Cloudera-Training-VM-4.1.1.c.vmdk (Normal, 17.00 GB)

Figure 20: Create a VM

- Click on "Create", Provide appropriate file storage name and Size.
- New Virtual Machine "Hadoop Master Node" would be created.
- Now to install Ubuntu on this VM, Click on the newly created VM and select "Settings" option from VirtualBox application.
- Go to Storage tab and select Ubuntu .iso file.

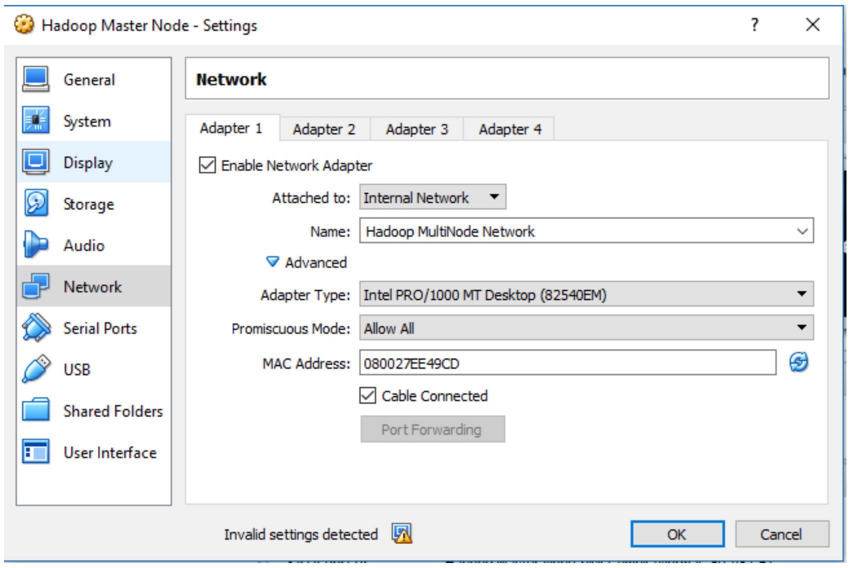
- Click "OK" and Start the VM.
- Install the Ubuntu Linux operating system into the Virtual Machine by providing necessary information during the install.
- After the installation is complete set Storage Controller value to its default value: "IDE Secondary Master".

Creating Cluster

Now we will construct Virtual Machine cluster of two nodes.

Configure Network

1. Stop "Hadoop Master Node" Virtual Machine if it is running.
2. Choose "Settings" option and choose "Internal Network" from Attached to: drop-down list.
3. Provide relevant Name to the network.
4. Choose "Allow All" option in Promiscuous Mode.
5. Network parameters would look like as below :



☑ Enable Network Adapter
Attached to: Internal Network
Name: Hadoop MultiNode Network
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)
Promiscuous Mode: Allow All
MAC Address: 080027EE49CD
☑ Cable Connected
Invalid settings detected

Figure 20: Network Parameters

Clone Virtual Machine

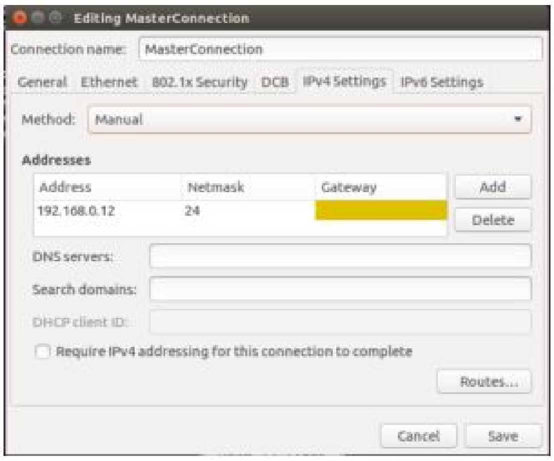
- Right click on "Hadoop Master Node" and choose "Clone" option.
- Provide name "Hadoop Slave1" as name and ensure "Reinitialize the MAC address" is checked.
- Click "Continue".
- Choose "Full Clone" option and click "Clone".
- Machine will be close within few minutes.
- Repeat steps 1-5 for another clone named "Hadoop Slave2"

Configure Node

- Login to "Hadoop Master Node" virtual machine.
- Go to Terminal, Execute below command:

```
sudo addgroup Hadoop
sudo adduser --ingroup hadoop hduser
```

- Edit network connection and choose IPv4 Settings.
- Assign IP address such as 192.168.0.12 to "Hadoop Master Node" VM.



Editing MasterConnection

Connection name: MasterConnection

General | Ethernet | 802.1x Security | DCB | IPv4 Settings | IPv6 Settings

Method: Manual

Addresses

Address	Netmask	Gateway
192.168.0.12	24	

DNS servers:

Search domains:

DHCP client ID:

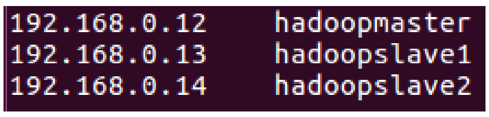
☐ Require IPv4 addressing for this connection to complete

Figure 23: IP Address Configuration

- Edit /etc/hosts using below command:

```
sudo nano /etc/hosts
```

- Update the file with following values :



```
192.168.0.12    hadoopmaster
192.168.0.13    hadoopslave1
192.168.0.14    hadoopslave2
```

- Press Ctrl+O to save changes to file.
- Edit /etc/hostname using below command:

```
sudo nano /etc/hostname
```

- Provide "hadoopmaster" without quotes into the text file and save changes by pressing Ctrl+O.
- Repeat steps 1-9 for "hadoopslave1" and "hadoopslave2" by assigning respective hostname and incremental IPs such as 192.168.0.13 and 192.168.0.14 respectively.
- Reboot all machines.

SSH Access

- Login to "Hadoop Virtual Machine" and use below command to generate ssh keys:

```
ssh-keygen -t rsa -P ""
```

- Add generated keys to authorized key list using below command :

```
cat $HOME/.ssh/id-rsa.pub >> $HOME/.ssh/authorized_keys
```

- Login to "Hadoop Slave1" and copy master node keys to authorize keys to slave1.

```
ssh-copy-id -i $HOME/.ssh/id-ras.pub hduser@hadoopslave1
```

- Repeat step-3 for hadoopslave2.

A.2 Hadoop Environment Setup

A.2.1 Hadoop Installation

1. Login to "Hadoop Master Node".
2. Download Hadoop from Apache release URL : http://hadoop.apache.org/releases.htm
3. Download and unzip the folder content using below command:

```
sudo tar -xvzf hadoop-2.7.2.tar.gz
```

4. Move the folder to '/usr/local' folder using below command:

```
sudo mv /home/hadoopmaster/Downloads/hadoop-2.7.2   \
/usr/local/hadoop
```

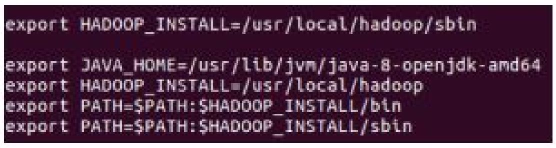
5. Change the ownership of folder using below command:

```
sudo chown -R hduser:hadoop  hadoop
```

6. Update bashrc with java environment path using below command :

```
sudo nano ~/.bashrc
```

7. Add below line into the file and press Ctrl+O to save changes.



```
export HADOOP_INSTALL=/usr/local/hadoop/sbin

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
```

8. Repeat steps 1-7 for hadoopslave1 and hadoopslave2 virtual machines.

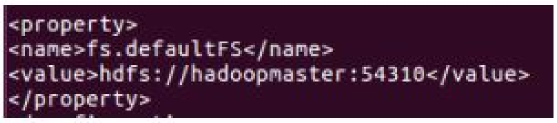
Master Node Configuration

1. Login to "Hadoop Master Node".

2. Edit "hadoop-env.sh" file available at "/usr/local/hadoop/etc/hadoop/" and ensure the JAVA_HOME path is correct.

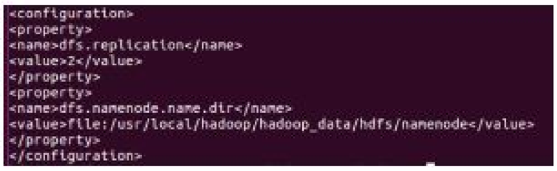
```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-i386
```

3. Edit "core-site.xml" file available at "/usr/local/hadoop/etc/hadoop/" and below entries within ¡configuration¿ tag:



```
<property>
<name>fs.defaultFS</name>
<value>hdfs://hadoopmaster:54310</value>
</property>
```

4. Update "hdfs-site.xml"on master node as below:



```
<configuration>
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop/hadoop_data/hdfs/namenode</value>
</property>
</configuration>
```

5. Update "mapred-site.xml" on master node as below:

6. Format the namenode using below command on Terminal:

```
hdfs namenode -format
```

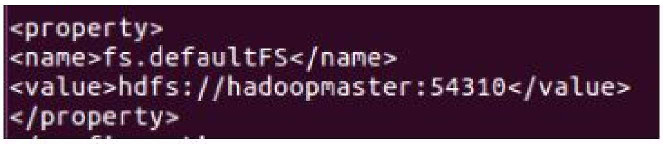
Slave Node Configuration

1. Login to "Hadoop Slave1".

2. Edit "hadoop-env.sh" file available at "/usr/local/hadoop/etc/hadoop/" and ensure the JAVAHOME path is correct.

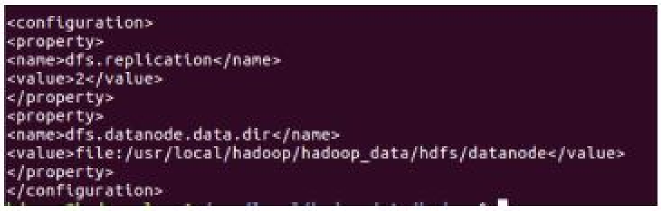
```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-i386
```

3. Edit "core-site.xml" file available at "/usr/local/hadoop /etc/hadoop/" and below entries within ¡configuration¿ tag:



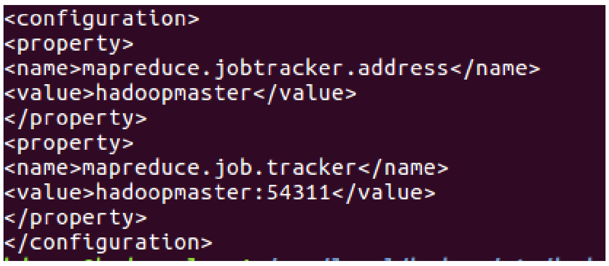
```
<property>
<name>fs.defaultFS</name>
<value>hdfs://hadoopmaster:54310</value>
</property>
```

4. Update "hdfs-site.xml" and "mapred-site.xml"on slave nodes as per above configuration values:



```
<configuration>
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:/usr/local/hadoop/hadoop_data/hdfs/datanode</value>
</property>
</configuration>
```

Figure 24: hdfs-site.xml



```
<configuration>
<property>
<name>mapreduce.jobtracker.address</name>
<value>hadoopmaster</value>
</property>
<property>
<name>mapreduce.job.tracker</name>
<value>hadoopmaster:54311</value>
</property>
</configuration>
```

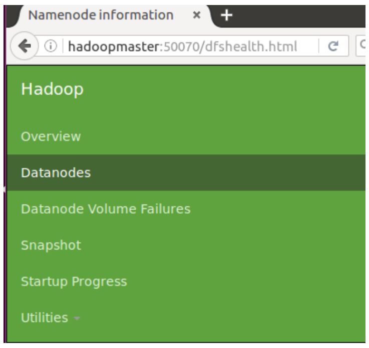
Figure 25: mapred-site.xml

5. Repeat step 1-5 for "Hadoop Slave2" virtual machine.

6. Login to "Hadoop Master Node" and start the all services using below command.

```
sudo bash /usr/local/hadoop/sbin/start-all.sh
```

7. Ensure Hadoop MapReduce is online :



A.2.2 Performance Tuning Component Setup

PerformanceTuning component is installed on NameNode to perform optimization of configuration parameters. Below are the steps required to configure PerformanceTuning-Component successfully :

1. Unzip the PerformanceTuningComponent into directory as shown below:

```
sudo tar -xvzf PTC.zip
```

2. Move the PTC.jar to Hadoop Installation Library,as shown below:

```
sudo mv /home/hduser/Downloads/PTC/PerformanceTuning.jar \
$HOME_INSTALL/share/hadoop/mapreduce/lib
```

3. Move the PerformanceTuning configuration file to Hadoop configuration director using below command :

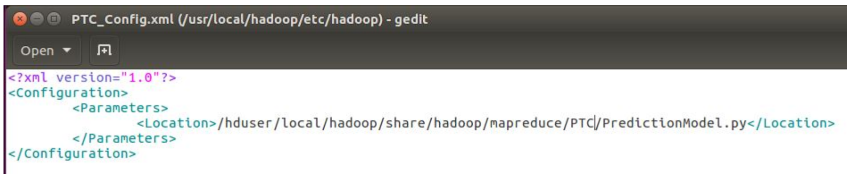
```
sudo mv /home/hduser/Downloads/PTC/PTC_Config.xml \
$HOME_INSTALL/etc/hadoop
```

4. Create folder "PTC" under $HOME_INSTALL/share/hadoop/mapreduce and move the PredictionModel.py and linearregression.pickle file to newly created folder using below commands:

```
sudo mv /home/hduser/Downloads/PTC/PredictionModel.py \
$HOME_INSTALL/share/hadoop/mapreduce/ptc/

sudo mv /home/hduser/Downloads/PTC/linearregression.pickle \
$HOME_INSTALL/share/hadoop/mapreduce/ptc/
```

5. Update the path of PredictionModel.py within PTC_Config.xml file



```
<?xml version="1.0"?>
<Configuration>
        <Parameters>
                <Location>/hduser/local/hadoop/share/hadoop/mapreduce/PTC/PredictionModel.py</Location>
        </Parameters>
</Configuration>
```