PRELIMINARY RESULTS ON MODELING CPU UTILIZATION OF MAPREDUCE PROGRAMS

TECHNICAL REPORT 665

NIKZAD BABAII RIZVANDI, YOUNG CHOON LEE AND ALBERT Y. ZOMAYA

DECEMBER, 2010
Preliminary Results on Modeling CPU Utilization of MapReduce Programs

Nikzad Babaii Rizvandi¹², Young Choon Lee¹, Albert Y. Zomaya¹

¹ Center for Distributed and High Performance Computing
School of Information Technologies, University of Sydney
NSW 2006, Australia

² National ICT Australia (NICTA), Australian Technology Park
Sydney, NSW 1430, Australia

{nikzad, yclee}@it.usyd.edu.au
albert.zomaya@sydney.edu.au

Abstract—In this paper, we propose an approach for predicting the CPU utilization of applications when they are running on MapReduce. Our approach has two key components: a set of application experiments running on MapReduce to profile the CPU utilization of the application on a given platform, and a regression-based model that maps the MapReduce configuration parameters (number of Mappers, number of Reducers, size of File System (HDFS) and the size of input file) into the CPU utilization for the application. This derived model can be used for predicting CPU requirements of the same application running on MapReduce on the same platform. Our approach aims to eliminate error-prone manual processes and presents a fully automated solution. Our evaluation on running three real applications (WordCount and Exim Mainlog parsing) on pseudo-distributed mode MapReduce shows that our automated model generation procedure can effectively characterise the CPU resource of these applications with average prediction error of 3.5% and 2.75%, respectively.

1. Introduction

Nowadays, cloud computing has received a lot of attention from both research and industry due to the deployment and growth of commercial cloud platforms [1]. Such services enable customers to change, or dynamically supplement, their own IT infrastructures with large choices of computational and storage resources that are accessible anytime. From the other side, cloud providers charge customers based on their usage or reservation of datacenter resources (CPU hours, storage capacity, network bandwidth, etc) which results in a typical dependency between service level agreements (SLAs) and resource availability. Therefore, predicting the amount of these resources becomes important for customers to decide how many nodes and for how long they should hire from the cloud providers [2]. Moreover, such a prediction can be used by cloud providers to guide scheduling and resource management decisions, and realistic workload generators to evaluate the choice of policies prior to full production deployment.

Recently, businesses have started using Mapreduce as a popular computation framework for processing large scale of data in both public and private clouds. Especially many Internet endeavours, i.e. private clouds, utilize MapReduce for analysing their core business and mining their produced data. Therefore, there is a significant benefit to application developers understanding performance trade-offs in MapReduce-style computations, so they can customize their design to efficiently whole workload resource utilization.

Generally, the MapReduce users run a few number of applications for a long time. For example, Facebook, which is based on Hadoop (Apache implementation of MapReduce in Java), is using MapReduce to read the daily produced log files and filter database information depending on the incoming queries. Such applications are repeated million times per day in Facebook. Another example is Yahoo where around 80-90% of the jobs is based on Hadoop. The typical applications are searching among large quantities of data, indexing the documents and returning appropriate information regard to the incoming queries. The same as Facebook, these applications are run million times per day for different purposes. One of the major problems with direct influence on Mapreduce performance is tuning the effective configuration parameters for effective running of a program. Generally, Mapreduce
users face with performance problem because of how to properly set these configuration parameters [3]. Therefore, it becomes important for cloud users to predict a Mapreduce application performance based on different values of configuration parameters before running of the application on actual system. In addition, the outcome of this prediction can be utilized by cloud providers to make better scheduling policies.

In this paper, we propose a general methodology to model CPU utilization of a Mapreduce application on a given platform. Here, CPU utilization is defined as the average percentage of CPU usage during executing the Mapreduce application multiply by the time of executing the application. Among effective Mapreduce configuration parameters, we choose four major configuration parameters which are the number of Mappers, the number of Reducers, the size of File System (HDFS) and the size of input file. Then, we iterate the application for different values of these parameters and capture the value of CPU utilization from the native system. Finally, a linear model is constructed by applying linear regression on the set of configuration parameters values (as input) and the CPU utilization values (as output). Obviously, our modelling technique can be performed for other configuration parameters and also for modelling the other resources such as Disk utility, network utility and memory usage. Although, our modelling technique can be applied for different applications on different platforms, two issues should be concerned: first, the obtained model of an application on a specific platform may not be used for predicting the same application on another platform and second, the modelling of an application on a platform is not applicable to predict other applications on the same platform.

2. Related work

MapReduce: Almost one of the early work on analysing/improving MapReduce performance goes back to 2009 in the work done by Zaharia et al [4], which pointed the problem of improving the performance of hadoop for Heterogeneous environments. The critical assumptions in Hadoop are that the cluster nodes are homogeneous and as well the tasks progress linearly. Hadoop utilizes these assumptions in order to schedule tasks and re-execute the stragglers. This paper designs a new scheduling policy to overcome these assumptions. In addition to this paper, there are a quiet number of publications to enhance or analysis the performance of different parts of MapReduce framework particularly in scheduling [5], energy efficiency [2, 6-7] and workload optimization[8]. A statistics-driven workload modelling was introduced in [7] in order to effectively evaluate design decisions in scaling, configuration and scheduling. The framework in this paper is utilized to make appropriate suggestions to improve the energy efficiency of MapReduce. A modelling method was proposed in [9] for finding the total execution time of a MapReduce application. It uses Kernel Canonical Correlation Analysis to obtain the correlation between the performance feature vectors extracted from MapReduce job logs, and map time, reduce time, and total execution time. These features are critical for establishing any scheduling decisions. Recent works in [10-11] reported a basic model for MapReduce computation utilizations. At first, the map and reduce phases are modelled using dynamic linear programming independently. Then these phases are combined to build a global optimal strategy for MapReduce scheduling and resource allocation. The effective configuration parameters in Mapreduce framework and their influences on the Mapreduce performance have been studied in [3]. Also, it was anticipated a possibility of developing a cost model in order to optimally parameter settings for Mapreduce configuration parameters. Our modelling technique which verifies this possibility can be utilized to obtain the approximate relation between these parameters and the CPU utilization. The outcome of our model then can be used to effectively find the best combination of configuration parameters in order to execute a MapReduce application before running on the native system.

System Profiling and modelling: system profiling is a mechanism to gather information from both software and hardware of a system in order to model the system. In high performance computing systems, profiling system specification is utilized for modelling different parts of the system. For example, in [12] a combination of performance modelling and prediction was applied to reduce execution times with respect to their predefined energy usage upper limit. After creating models for both execution time and energy consumption, key parameters of models are estimated by executing a program for a small number of times and then regressing the estimated parameters. In a recent published work from Microsoft researchers, the idea of profiling and modelling was used for power metering in Virtual Machines (VMs) [13].The major components in hardware such as CPU, Memory and Disk are modelled regard to some unknown coefficients. Then linear regression is utilized to train these coefficients due to the profiling of thousands of traces. Among methods in literature, there is a
work done by Wood et al in [14] which is closely related to our idea and the method we proposed in the paper. This work proposed a linear regression-based model to predict the CPU resource overhead between running an application in Virtual Machine (VM) and the native system. The CPU usage of the application in VM was extracted by XenTop and XenMon tools and in the same time the CPU usage in native system was captured by SysStat package [15]. Then a linear regression model was applied to form a linear relation between the CPU usage in VM and native system. In our design we follow the similar logic: we use SysStat package to capture the CPU utilization of an application in MapReduce. Then using regression, we model the relation between the MapReduce configuration parameters and the CPU utilization extracted from SysStat. Later, we apply the derived model to the same application but with different MapReduce configuration parameters and use it for predicting the CPU resource requirements of the application before its actual running.

3. Application modelling in MapReduce

3.1. Problem definition

MapReduce, introduced by Google in 2004 [16], is a framework for processing the large quantities of data on distributed systems. The computation of this framework has two major phases (Figure 1):

- “Map” phase: after copying the input file to the MapReduce file system and split the file into smaller files, the data inside the files are converted into <key,values> format (e.g. key can be the line number and values are the data in the lines). These <key,values> pairs are entered to the mappers and the first part of processing are applied on them. One of the major advantageous of MapReduce is that the mappers are independent. Therefore in theory, it gives a good opportunity for parallelization. However, this parallelization can be bounded because of the data source and/or the numbers of CPUs close that data.
- “Reduce” phase: After finishing the Map phase, a network intensive job starts in order to send the intermediate <key,values> coming from mappers to the reducers. Then, depending on the MapReduce configuration, a sort/shuffle stage may be applied. After that, the map operations with the same key will be presented to the same reducer, at the same time. The result then be written in output files (typically one output file) in the file system.
The process of converting an algorithm into independent mappers and reducers causes MapReduce to be inefficient for algorithms with sequential nature. Generally, MapReduce has been designed for computing on significantly large quantities of data instead of making complicated computation on a small amount of data [17]. Due to its simple structure, MapReduce is suffering from some serious issues especially in scheduling, energy efficiency and resource allocation. Therefore, predicting an application resource requirement before running on a native system can make a big contribution on modifying MapReduce scheduling and resource allocation.

In distributed computing systems, MapReduce has been known as a large-scale data processing or CPU intensive job [3, 18-20] which implies that CPU utilization is the most important part of running an application on MapReduce. Therefore, predicting the amount of CPU an application needs becomes important for customers to hire enough CPU resources from cloud providers and for cloud providers to schedule incoming jobs properly. Among all of the parameters influencing MapReduce performance, in this paper we are going to study the dependency between MapReduce configuration parameters and the CPU utilization of the system. We expect that CPU utilization of an application in MapReduce is highly correlated and proportional to the MapReduce configuration parameters as shown in Figure 2. The diagonal line pink bar shows running an application with the same configuration parameters except the number of mappers while the vertical line green bar shows the same application for fixed values of configuration parameters except the number of reducers. The x-axis in the figure addresses the experiment index (e.g. the i\textsuperscript{th} index represents the actual and predicted values of i\textsuperscript{th} experiment with a set of values for configuration parameters) while the y-axis is the percentage CPU utilization of the experiment during its total running. In another word, if the experiment starts at $t_1$ and finishes at $t_2$ and the average percentage CPU utilization of this experiment in this time interval is $\%CPU$, then total $\%CPU$ utilization of the application is calculated as: $\%CPU \times (t_2 - t_1)$. Therefore, it is expected the $\%CPU$ utilization in y-axis be more than 100%. As can be seen there is high dependency between the amount of CPU usage and changing these configuration parameters. The straightforward benefit of finding a model between the MapReduce configuration parameters and CPU utilization is that one can find the best values of the parameters to optimize the amount of CPU utilization of his application. This means if somebody wants to run his application of a cloud (like Amazon EC2), by modelling he can approximately find how much virtual nodes his application needs and for how long.

### 3.2. Profiling the CPU utilization of applications

For each application, we generate a set of experiments with different values of four MapReduce configuration parameters on the given platforms. While running each experiment, the CPU utilization of the experiment is gathered to build a trace for future using as training data of the model (this statistic can be gathered easily in Linux with the SysStat monitoring package which has low overhead). Within the system, we sample the CPU usage of the experiment in native system from starting mappers till

![Figure 2. The relation between number of mappers/reducers and an application CPU utilization. The x-axis is the index of the experiment while the y-axis is %CPU utilization of the application during its execution on Mapreduce framework.](image-url)
finishing reducers with time interval of one second. Then the value of total CPU usage of this experiment is a summation of all CPU usage samples during this time (Figure 3). Because of the temporal changes, it is expected that several running of an experiment with the same configuration parameters results in slightly different total CPU utilization. Therefore, utilizing a mechanism to prune the unsuitable gathered data from the training dataset will improve the modelling accuracy. In [14], Robust Stepwise Linear Regression was used as a post processing stage to refine the outcome of the model by giving weights to data points with high error. However, we used a simple technique to prune the data set as follows:

Run N instances of an experiment with the same values of four configuration parameters.
1) For the instances of the experiment:
   2) Calculate the mean
   3) Find the instances outside of ±10% of the mean value
   4) Exclude these instances
   5) Repeat steps 3 and 4 till all instances be inside the range
Then the final value of total CPU usage of this experiment is the final calculated mean in step 2. The same procedure must be followed for other experiments.

### 3.3. Model creation

This section describes how to create model of a running application on MapReduce framework by characterizing the relationship between a set of Mapreduce configuration parameters and CPU resource utilization metric. The problem of modeling an application in MapReduce based on linear regression involves choosing the suitable coefficients of the modeling such that the model’s response well approximates the real system response[14, 21].

Consider the linear algebraic equations for all experiments of an application for four configuration parameters:

\[
\begin{align*}
CPU^{(1)} &= a_0 + a_1M^{(1)} + a_2R^{(1)} + a_3F^{(1)} + a_4I^{(1)} \\
CPU^{(2)} &= a_0 + a_1M^{(2)} + a_2R^{(2)} + a_3F^{(2)} + a_4I^{(2)} \\
&\vdots \\
CPU^{(N)} &= a_0 + a_1M^{(N)} + a_2R^{(N)} + a_3F^{(N)} + a_4I^{(N)}
\end{align*}
\]

where \(CPU^{(j)}\) is the value of total CPU utilization of an application in \(j^{th}\) experiment on MapReduce and \(M^{(j)}\), \(R^{(j)}\), \(F^{(j)}\) and \(I^{(j)}\) are the number of Mappers, the number of Reducers, the size of

![Figure 3. the flow of the Mapreduce application(left) and %CPU utilization extracted from actual system (right)](image)

<table>
<thead>
<tr>
<th>Number of Mapped</th>
<th>Number of Reducer</th>
<th>Size of Input</th>
<th>Total CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>29</td>
<td>624</td>
<td>100%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Mapped</th>
<th>Number of Reducer</th>
<th>Size of Input</th>
<th>Total CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>29</td>
<td>624</td>
<td>100%</td>
</tr>
</tbody>
</table>

\[
100\% \text{CPU Utilization} = \frac{\%CPU^{(j)} + \%CPU^{(j+1)}}{2} = 61.4782\% 
\]
MapReduce File System and the size of input file for this experiment, respectively. Using the above definition, the approximation problem becomes to estimate the values of $\hat{\alpha}_0, \hat{\alpha}_1, \hat{\alpha}_2, \hat{\alpha}_3, \hat{\alpha}_4$ to optimize a cost function between the approximation values and the real values of CPU utilization. Then, an approximated CPU utilization ($CPU^{(r)}$) of the application for the $r^{th}$ experiment is predicted as:

$$CPU^{(r)} = \hat{\alpha}_0 + \hat{\alpha}_1 M^{(r)} + \hat{\alpha}_2 R^{(r)} + \hat{\alpha}_3 F^{(r)} + \hat{\alpha}_4 I^{(r)}$$

There are a variety of well-known mathematical methods in literature to calculate the variables $\hat{\alpha}_0, \hat{\alpha}_1, \hat{\alpha}_2, \hat{\alpha}_3, \hat{\alpha}_4$. One of these methods used widely in computer science and finance is Least Square Regression which calculates the parameters in Eqn.2 by minimizing the error:

$$error = \frac{1}{N} \sum_{j=1}^{N} (\hat{CPU}^{(j)} - CPU^{(j)})^2$$

The set of coefficients $\hat{\alpha}_0, \hat{\alpha}_1, \hat{\alpha}_2, \hat{\alpha}_3, \hat{\alpha}_4$ is the model that describes the relationship between total CPU utilization of an application in the real system regard to these four MapReduce configuration parameters. In another word, the relation between CPU utilization of a Mapreduce application and the configuration parameters is:

$$CPU = f(M, R, F, I)$$

$$= \hat{\alpha}_0 + \hat{\alpha}_1 M + \hat{\alpha}_2 R + \hat{\alpha}_3 F + \hat{\alpha}_4 I$$

Once a model has been created, it can then be applied to CPU resource utilization traces of the same application to estimate what their CPU requirements will be if the values of Mapreduce configuration parameters change. It also should be considered that the obtained model of an application may be different from another application.

### 4. Experimental results

In this section, we evaluate the effectiveness of our models under several realistic applications on two different hardware platforms.

#### 4.1. Experimental setting

Our modelling procedure has been implemented and evaluated on the pseudo-distributed MapReduce mode on two platforms. In our evaluation, the system runs Hadoop version 0.20.2 which is Apache implementation of Mapreduce developed in Java [22]. Concurrent with running an application in Hadoop environment, in another terminal the resource utilization of the application in the native system is monitored with the SysStat package [15]. For an experiment, statistics are gathered from “running job” stage to the “Job complete” stage (arrows in Figure 3) with sampling time interval of one second. Then all CPU usages between these two times are summed and formed the total CPU utilization of the
To avoid system temporal changes, the experiment is repeated several times with the same values of configuration parameters and the algorithm in section 3.2 is applied.

The evaluation of the proposed modelling technique is based on three realistic Mapreduce applications:

- **WordCount** [23-24]: This application reads data from text files, counts how often words occur and then returns the result to text files. Each line of output files contains a word and the number of its occurrence, separated by TAB. In running a WordCount application on Mapreduce, each mapper picks a line as input and breaks it into words. Then it assigns a <key, value> pair to each word as <word, 1>. In reduce stage, each reducer counts the values of pairs with the same key and returns the number of each word occurrence.

- **Exim mainlog parsing** [25]: Exim is a message transfer agent (MTA) for logging information of sent/received emails on Unix systems. This information is then saved in exim_mainlog file which becomes too large in mail servers. Parsing the data in the exim_mainlog file and therefore organizing all incoming information in this file into individual transactions separated and arranged by their transaction ID is the target of this Mapreduce application.

We have tested our modelling on two different hardware platforms:

- **Dell Latitude E4300**, two processors: Intel Centrino model 2.26GHz, 64-bit; 2 x 2GB memory; 80GB Disk
- **HP Pavilion**, one processor: AMD Athlon, 1.7 GHz, 32-bit, 2 GB memory, 120GB Disk

For each application there are two phases: (1) training phase, and (2) prediction phase. In the training phase, all gathered information from experiments accompanied by the related values of configuration parameters is utilized to train a specific model for that application. Each application includes 100
different experiments with different values of configuration parameters. Each experiment is repeated eight times to remove the effects of temporal changes (section 3.2). In both training and prediction stages, the number of mappers and reducers are chosen between 1 to 40. While the size of file system and the size of input file vary between 1 Mbyte to 50Mbyte and 10MB to 500MB, respectively. In the prediction phase, new experiments with randomly chosen values for configuration parameters are generated and then applied to the model. The differences between the real CPU utilization extracted by SysStat package and predicted CPU utilization by the model are then compared as shown in Figure 4.

4.2. Results
To test the accuracy of an application’s model, we use it to predict the CPU utilization of some experiments of that application with randomly values of the four configuration parameters in the predefined range. We then run the experiments on the real system and capture the real CPU utilization to determine the prediction error (Figure 4).

In this section we evaluate the obtained models on three real Mapreduce applications: WordCount, Exim_Mainlog parsing and k-mean clustering. Figure 5 shows the prediction accuracy and prediction error of the WordCount application between actual %CPU utilization and its predicted one. The same graph has been shown in figure 6 for Exim_mainlog parsing application, respectively.

We find that the average error between the real CPU utilization and the predicted by models is less than 5% for the tested applications. Obviously, some of the errors come from the model inaccuracy, but it can also be because of temporal changes in system resulting in CPU utilization increase for a short time. As can be seen from figure 5, there are some spikes in prediction errors. These spikes generally happen in the low values of CPU utilization which probably caused by background processes running during executing the applications. For example in hadoop, one of the main background processes comes from streaming when the mapper/reduce are written in another language than Java. As these background processes typically spend low CPU utilization, their influences become significant when CPU utilization of the Mapreduce application is low.

Although the obtained model can successfully predict the amount of CPU resources needs for a Mapreduce application, it can not give information about how application performance, such as response time, changes or how CPU utilization varies during time. Finally, our approach can help cloud customers and providers to estimate the minimum amount of CPU resources which have to be allocated to a Mapreduce application in order to prevent significantly reduced performance because of CPU resource limitation.

5. Discussion and Future work
In this section, we discuss about different behavioural of the modelling for different settings and platforms.

- **Pseudo-distributed mode**
  In our experiments, we used hadoop pseudo-distributed mode for running applications in Mapreduce framework. In this mode, mappers and reducers work as threads independently on a single machine. Generally, behaviors of this mode is close to real distributed mode except in network utilization; therefore, it is expected our modelling technique based on pseudo-distributed mode be still valuable in the real distributed system. The main reason is that the number of required tick clocks, e.g CPU utilization, to process a bunch of mappers/reducers does not change when moving from one mode to another or from one machine to another machine. The major difference between these two modes becomes visible in the stage of sending the output of map phase to the reduce phase. This stage, which is known as network intensive stage in real distributed systems, cause a small CPU overhead because of handling network devices and managing network communications.

- **Language dependency**
  As hadoop implementation of Mapreduce was written in java, writing an application in Java can make a fewer CPU utilization than when it was written in other languages. This is because hadoop uses streaming interface to utilize codes from other languages. This interface, however, makes almost fixed overhead on top of the CPU utilization of an application. Besides the prediction error discussed in section 4.2, this overhead may cause error when a model of an application written in
Java is used to predict the CPU utilization of the same application written in another language like python.

**Energy cost**

If a Mapreduce application spends in average $\%CPU_{active}$ and $\%CPU_{idle}$ in active and idle time during running the application, respectively, and the completion time of the application takes $T$ seconds, the amount of energy consumed for computation can be calculated as:

$$E_{CPU} = E_{CPU, Active} + E_{CPU, Idle} = (\alpha \%CPU_{active} + \beta \%CPU_{idle})T$$

(4)

Where CPU spends $\alpha \frac{Watt}{sec}$ and $\beta \frac{Watt}{sec}$ in active and idle time, respectively. Therefore, our modelling can be indirectly used to predict the CPU energy consumption of an application before running on the native system.

**How to use the modelling:**

Although, the modelling coefficients of a Mapreduce application by our proposed method may change from one platform to another and two different applications may create different models on the same platform, we expect linear regression modelling can model the relation between the Mapreduce configuration parameters and the CPU utilization of the application in most cases. In addition to the benefit of cloud provider to use such prediction for using better scheduling policies, cloud customers also can get benefit. Generally, a cluster in a middle size company is used for 2-3 years which means all applications are run on the same platform. If some of the applications repeated frequently, our proposed technique can be utilized for profiling and modelling of each application separately and therefore calculate the cluster performance for these applications. Moreover, by finding a mathematical relation between these parameters and the CPU utilization (or maybe other resources), it is easily possible to twist the problem of finding the best values of configuration parameters into an optimization problem for an input file with size $I_{in}$ as following:

$$\begin{align*}
\text{Input: } & I_{in} \\
\text{minimise: } & CPU_{Utilisation} = f(M, R, F, I_{in}) \\
\text{such that:} & \begin{align*}
1. & 1 \leq M \leq 40 \\
2. & 1 \leq R \leq 40 \\
3. & 10M \leq F \leq 500M
\end{align*}
\end{align*}$$

(5)

where $M$, $R$ and $F$ are the number of Mappers, the number of Reducers and the size of MapReduce File System for this experiment, respectively.

**Future work:**

The plan for future work is to model the other resources such as network, memory and disk utilities. As in pseudo-distributed mode, network is not under stress we have to move to the real-distributed mode but we expect the results of modelling for CPU be almost the same for both pseudo- and real-distributed modes. Moreover, we are also interested in utilizing complex modelling techniques to reach accurate models. Our final next step is to investigate the possibility of generalizing CPU utilization model for all applications in Mapreduce. In this paper, we model each application individually which, as we mentioned before, the CPU utilization model of a Mapreduce application for a specific platform may not be useful for predicting the CPU utilization of other applications on the same platform. Moreover, modelling of an application highly depends on the platform. This means the obtained model of an application must be repeated when it moves to another platform.

**6. Conclusion**

The motivation behind this work is the need to predict CPU requirement of a Mapreduce application before running on real cluster/cloud. Therefore, we proposed an approach to extract/profile CPU usage information of a Mapreduce application from native system and then apply linear regression model to make a relation between the Mapreduce configuration parameters and the CPU utilization of the
application. This model can be utilized by developers to estimate how much CPU resources their application need and by cloud providers to estimate the CPU need of an incoming application in order to make appropriate scheduling and resource allocation strategy. Our evaluation shows that our modelling can effectively predict the computation cost of two realistic applications with the median prediction error of less that 4%. In conclusion, although the modelling was done for a few applications, we expect the modelling procedure be valid for other applications. It means CPU utilization of running an application on MapReduce can be modelled by linear regression regards to four Mapreduce major configuration parameters. It is obvious that the coefficients of the modelling changes from one application to another and from one platform to another platform.

7. Acknowledgment

The work reported in this paper is in part supported by National ICT Australia (NICTA). Professor A.Y. Zomaya's work is supported by an Australian Research Council Grant LP0884070.

References
