Execution Time Prediction of Imperative Paradigm Tasks for Grid Scheduling Optimization

Maleeha Kiran [1,2], Aisha-Hassan A. Hashim[1], Lim Mei Kuan[2], Yap Yee Jiun[2]

[1]*Department of Electrical & Computer Engineering, Faculty of Engineering, International Islamic University Malaysia, 53100 Gombak, Selangor, Malaysia*

[2] *Centre for Multimodal Signal Processing, MIMOS BERHAD, Technology Park Malaysia, 57000 Kuala Lumpur, Malaysia*

Summary

An efficient functioning of a complicated and dynamic grid environment requires a resource manager to monitor and identify the idling resources and to schedule users' submitted jobs (or programs) accordingly. A common problem arising in grid computing is to select the most efficient resource to run a particular program. At present the execution time of any program submission depends mostly on guesswork by the user. The inaccuracy of guesswork leads to inefficient resource usage, incurring extra operational costs such as idling queues or machines. Thus, in this paper we propose a job execution time prediction module to aid the user. The proposed system will function as a standalone unit where its services can be offered to users as part of a grid portal. This system focuses on imperative paradigm tasks as they are commonly used in a grid environment. We propose a novel methodology and architecture to predict the execution time of jobs using aspects of static analysis, analytical benchmarking and compiler based approach. Essentially a program is analyzed in segments for execution time and these times are combined together to give the total execution time of the program. The experimental results show that the technique is successful in achieving a prediction accuracy of greater than 80%. Future work may involve handling other paradigms such as object-oriented programming and investigating the possibility of integrating the prediction module into a real grid environment.

Key words:
Prediction module, Grid scheduling, Job execution time

1. Introduction

Grid computing technology coordinates physically distributed resources that cross organizational boundaries to allow aggregation and sharing of heterogeneous resources. It enables access to tremendous computing power that can be harnessed for performing computationally intensive problems in the area of science, technology, commerce and engineering [1]. The basic goal of a grid computing environment is to allow users to access computational resources by just "plugging-in" to the grid, similar to the way electrical energy is supplied when one plugs into the electrical power grid. Grid services are treated like a utility such as electricity, where once the user is connected to the grid it appears as essentially one large computer system [2]. Users do not have to know which resources they are using or where the resources are located, they just "plug in" to the grid to access the computational power and data storage.

As the grid is a heterogeneous environment, it is partitioned into basic units known as "virtual organization" or VO. A VO comprises of a set of grid entities such as applications, services or resources that are related to each other according to some level of trust. This level of trust is defined by sharing rules which determine how the resources are shared by individuals and institutions participating in a VO [3]. A grid could potentially consist of many VO and a grid entity (applications, services or resources) can be a member of more than one VO [4]. In addition, a VO can span across several "physical" institutions and entities can join or leave the VO based on their current needs. In such a complex, dynamic and distributed environment, resource management and task scheduling are the key challenges to improving the resource usage efficiency on the grid. At present, there are many middleware technologies that schedule and distribute all types of application runs (serial, parallel, distributed memory) on all types of hardware (desktops, clusters and supercomputers and even cross

Manuscript received February 5, 2009
Manuscript revised February 20, 2009

sites) with varying levels of security. The most common example is given by the job scheduler that can be any of a complex set of products like Condor, LSF and PBS [5].

As shown in Figure 1, basically, the user interacts with a resource broker (in middleware) that hides the complexities of scheduling and distribution of resources in grid computing. The broker finds resources that the user can access through grid information catalogue, negotiates with grid-enabled resources, schedule tasks to specified resources, deploy the application and finally gather the results [6]. In order to do this, users are required to provide the specifications of requirements for the computational resources needed including the wall time (real running time) of programs upon submission. Providing the run time or execution time is not a minor task in a heterogeneous grid environment. Currently, the run time provided by users is based on guesswork, in which the user estimates a rough run time based on their theoretical knowledge or past experiences. Such estimates of prediction time provided by the users can be of any accuracy and it is impractical to rely solely on the users to provide such an estimate as they lack the knowledge of where the job will be run in a dynamic and heterogeneous grid environment [7]. Therefore, a prediction module which gives an estimated execution time of programs is both useful and relevant.

Hence, the aim of the research work presented here is to develop a prediction module that estimates the execution time of programs by using aspects of static analysis, analytical benchmarking and compiler based approach. For this phase, we propose a standalone prediction module whose focus is to predict the execution time of programs written using R!, software which belongs to the imperative programming paradigm. The remainder of this paper is organized as follows: Section 2.0 presents background and motivation for this research; Section 3.0 discusses some related work; Section 4.0 describes the proposed architecture, including the detailed description of each module, the information flow amongst the modules and their implementation; Section 5.0 describes the testing and evaluation phase, including preliminary results and finally, Section 6.0 presents our conclusions and future work.

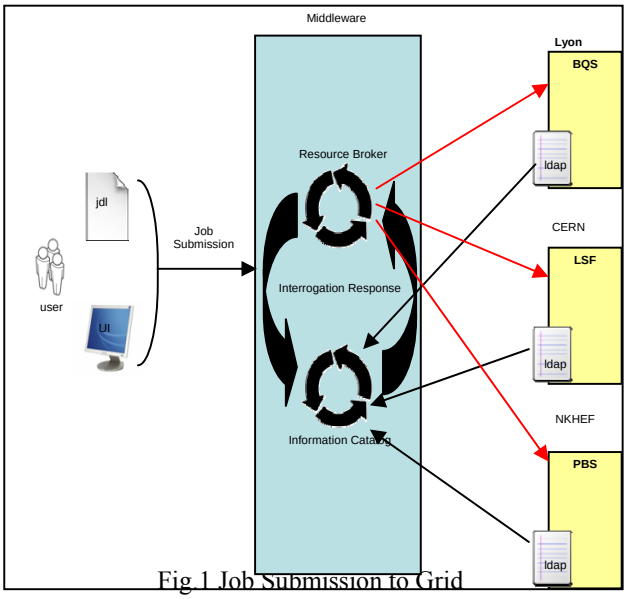


Fig.1 Job Submission to Grid

2.0 BACKGROUND AND MOTIVATION

2.1 The National Grid

The Malaysian Research and Education Network (MYREN) was launched in March 2005. MYREN provides high-capacity broadband to universities, colleges, research organizations and scientific laboratories [8]. As a government-funded program, MYREN acted as a networking super highway which enables researchers to run data-intensive applications, share computing elements and run advance applications within Malaysia as well as overseas.

MYREN consists of two networks: a production network and an experimental network. The production network is for exchange of high performance computing data in collaborative research and is based on the multi-protocol label switching (MPLS) with a capacity of up to 8Mbps. The experimental network employs point-to-point connectivity with 2Mbps. Its primary usage is for network research and testing of new network technologies such as internet applications and grid computing techniques.

MYREN is connected to several international research communities in Asia Pacific, Europe and North America,

via the Trans-Eurasia Information Network 2 (TEIN2), pan-European GEANT2 and Internet2. TEIN2 is partly funded by the European Union's and receives additional support from Maffin, NICT, NII and Juniper Networks. TEIN2 links the national networks in the Asia-Pacific region which comprise of China, Indonesia, Japan, Korea, Malaysia, the Philippines, Singapore, Thailand, Vietnam and Australia at speeds of up to 1Gbps [9]. GEANT2 is co-funded by the European Commission and connects 34 countries through 30 National Research and Education Networks (NRENs), using multiple 10Gbps wavelengths [10].

The high-capacity network aims to bridge the digital divide between different countries across the region. Malaysian researchers benefit from these projects as the broadband connectivity enables greater levels of research collaboration, access to international scientific resources such as biotech databases, scientific equipment and online libraries and information repositories. Furthermore, researchers can have the opportunity to work more efficiently on joint research projects with advance nations in the European Unions as well as in Asia. This partnership enables researchers to bring back best practices and research methodologies. Potential applications include natural disaster warning systems, e-learning and e-health initiatives, linking radio astronomy telescopes and other projects where faster transfer of massive amounts of data is vital.

To further extend the capability of MYREN, it is connected to high performance clusters in other private and public universities as well as government and private research institutions to form the National Grid. At present the whole structure is combined under the trade name of KnowledgeGrid Malaysia. It provides a high-level abstraction which covers both the National Grid as well as MYREN. KnowledgeGrid Malaysia is an initiative of the Ministry of Science, Technology and Innovation (MOSTI) and is being spearheaded by MIMOS which is responsible for its implementation and daily maintenance [11]. It is meant to provide the necessary computing power and resource required by individuals and industries alike.

2.2 Motivation for research

One of the factors that can enable efficient usage of resources in a grid environment is having an estimate of job execution time prior to running the job. This can aid the scheduling policy in reducing the queue wait time as well as allow planning of resource allocation in advance. However providing prediction of job execution time is a non-trivial task in a grid environment. Estimates of prediction time provided by the users can be of any accuracy and it is impractical to rely solely on the users to provide such an estimate as they lack the knowledge of where the job will be run in a dynamic and heterogeneous grid environment [12]. Thus developing an accurate model for predicting execution time of jobs on the user's behalf is necessary and before facing it in a grid environment, the problem must be studied and solved for local systems.

3.0 RELATED WORK AND DISCUSSION

Performance prediction of software is not relatively straight forward in the Grid environment due to its dynamic and heterogeneous nature. Generally when a user submits a job to the Grid, they are requested to provide an estimate of the execution time of their jobs. This is usually needed to assist scheduling policies or where such information is not mandatory, it can still be used to fine tune scheduling decisions. Furthermore estimates of job completion time are vital when conducting advance reservation for jobs where such estimates are used for future planning of resource allocation. Thus it is necessary to develop a model for predicting execution time of jobs to assist the user.

Some of the early work in this area focused on using a simulator such as MicroGrid, SimGrid and GridSim to obtain a better understanding of the Grid environment and to simulate the process of running different application on the Grid. A simulation is the process of executing applications on an emulated platform rather than the real platform [13]. Only a model of the application is run rather than the application itself. For example SimGrid was used for the simulation of "C" language application scheduling. But these simulators have their drawbacks as they work off-line, are relatively slow and do not simulate the online, dynamic environment of the grid well [12]. Also their use is impractical because of the performance overhead incurred when applied to making predictions of execution time involving large applications. Hence they are usually used as a tool for verifying results or are combined with other techniques to offset their disadvantages. A broad overview of the techniques used for predicting execution time of programs is shown in figure 2.

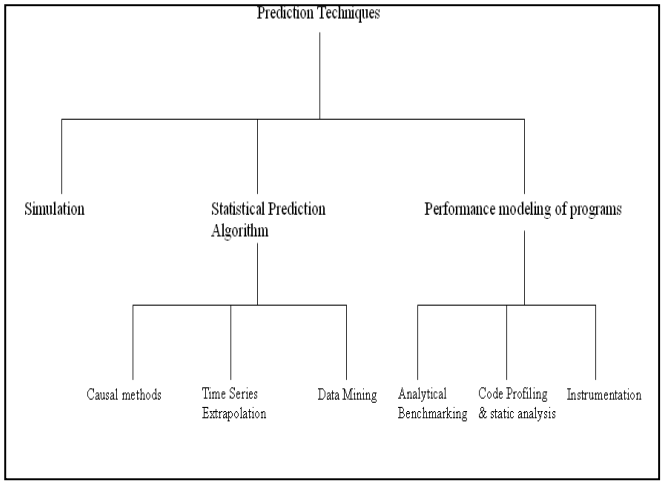


Fig. 2: Prediction Techniques

One of the most important factors affecting the choice of technique when making predictions of execution time depends on the availability of historical data. If a sufficient amount of historical data, demonstrating some degree of regularity is made available then it can be used to predict the execution time of jobs [14]. Statistical prediction algorithms predict execution time using historical data without requiring detailed knowledge of the underlying hardware and the application. A set of past observations are kept for each machine and these are used to make predictions of new incoming jobs. The prediction made is used to assist the scheduler when allocating resources to the job. Statistical algorithms are able to make better predictions as the number of past observations increases [15]. The analysis of the historical data can be used to produce estimates of best and worst case execution time and this information is used to identify critical grid components. The statistical method employed by these prediction algorithms can be grouped into three main categories: causal methods, time series extrapolation and data mining. It must be noted that the distinctions between the groupings are not absolute and many methods employ a combination of these techniques. The choice of which category of statistical method to use depends mainly on the kind of historical data available and type of parameter to be predicted.

Causal methods assume that future demand depends on past or current value of some variables and can anticipate variations in demands. These include regression, econometrics models, input-output models and even neural networks. Their main drawback is that it is difficult to find a variable that leads the forecasted variable in time. On the other hand time series extrapolation is based on the premise that some features of past demand patterns will remain constant and these can be projected to predict future demands. Some of these methods include moving averages, exponential smoothing and decomposition method [14].

In recent years, the application of data mining techniques in predicting job completion time has gained prominence. These techniques can be applied to very different kind of data, regardless of the nature of the data and they also include an automatic learning mechanism that allows them to discover or derive new knowledge without the necessary interaction of a third party [12]. Some of the data mining techniques which have been applied in this area include classification trees, clustering and statistical tests [16]. Much of the newer research is focused on using data mining techniques for performance prediction. The choice of which category of statistical method to use depends mainly on the kind of historical data available and type of parameter to be predicted. But it must be remembered that no prediction method can be considered superior to the other in every aspect.

Statistical predictions have their drawbacks as the accuracy of their prediction depends on how well the past observations are reflective of future incoming jobs. In addition they require that a separate set of historical data be maintained for different machines. Also it would be improper to employ statistical prediction algorithms in situations where no historical data is available [13]. This is the main reason for not exploring the usage of statistical prediction algorithms for the prediction module as there is no historical data available for programs written using R software. Thus the prediction module must rely on a method which aims to develop an understanding of incoming jobs.

Techniques which make prediction of execution time of programs without relying on historical data, have been borrowed from performance modeling of programs in traditional computing where exact qualification of software program and resources is carried out to predict their execution time. These range from analytical benchmarking, code profiling and static analysis to instrumentation. The main purpose of these techniques is to understand the different aspects of the behaviour of a program and they are usually used in conjunction with other approaches.

Analytical benchmarking involves specifying a number of primitive code types and obtaining benchmark data which determines the performance of each machine for each code type [15]. The output produced using this technique is specific to the architecture of each machine and needs to be repeated for each type of machine if cross-platform

predictions of execution time are to be performed. The proposed prediction module uses this approach when building the statistical database which contains benchmarked execution time of commonly used segments of R code. However if prediction of execution time needs to be made on a machine with a different architecture, then benchmarking needs to be carried out on this machine and the appropriate data added to the database. This will mean that the portability of the prediction module is limited to the machine whose benchmarked data is available in the statistical database.

Code profiling is not used by the prediction module as it involves recording the run-time behaviour of a program using a selected set of input data [15]. This method does not compensate for variation in input data set and it requires running the program at least once which is not a feasible option for programs meant for the grid environment as they run for much longer duration, such as days or even weeks. Some aspect of static analysis is adapted for use in the prediction module but this method alone is not sufficient as it does not account for program input and usually makes assumptions about the program properties that are not available [13]. For example the outcome of conditional statements, loop iteration counts, and recursion depths are rarely predictable using static analysis alone. This method also does not distinguish between frequently and infrequently executed program paths [17].

Instrumentation of code is done by inserting counters in each basic block to produce the dynamic statistics at run time. This method is used to profile basic block execution frequency and for analyzing memory hierarchy performance. Instrumentation of code is not used in this phase of the prediction module as using it would mean including extra code into the R script and running this code will add to the execution time of the script [17].

As there was no historical data available for the prediction module to rely upon when making its predictions, the approach meant for developing it had to rely upon analyzing the behaviour of incoming jobs (i.e. R scripts). In order to do this some aspects of static analysis were used and the data required for the database was obtained through analytical benchmarking. However these techniques alone are not sufficient to acquire a complete understanding of the behaviour of R scripts. Thus the prediction module also adopts a compiler-based approach to extract the additional information from the R scripts. The uniqueness of the prediction module lies in combining these three approaches and then performing the necessary computation to predict the execution time of R scripts.

4.0 PROPOSED SYSTEM ARCHITECTURE

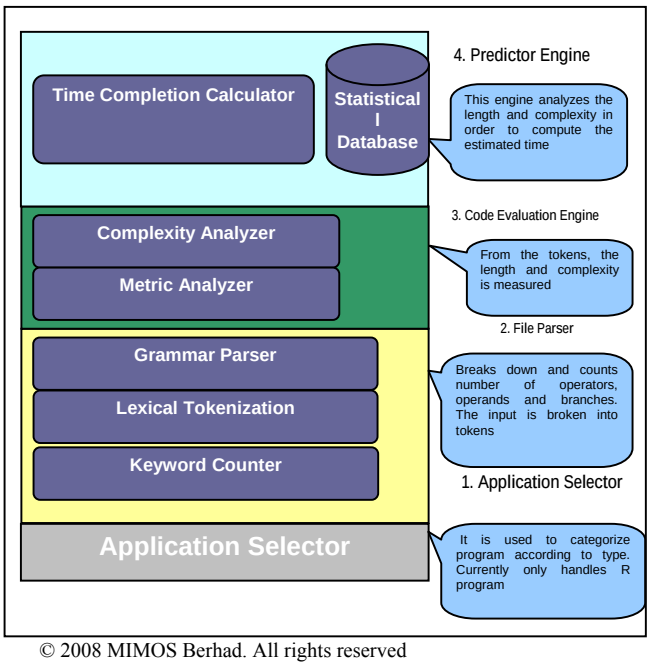
A common type of job on the grid may involve software programs written in high-level languages, e.g. C++, C, Pascal, etc. In the present phase, our work focuses on programs written using R! software. Therefore, the approach used to develop the prediction module is very much dependent on the characteristics of R! software. Program written using R! software belong to the imperative paradigm, which means it maintains a modifiable memory and computation are performed through a sequence of steps specified by a list of commands [1][18]. This allows the user greater flexibility in developing their programs according to individual coding style. Unfortunately it also means the program developed does not have an identifiable pattern which can be studied to make predictions regarding future incoming job. Thus the R programs are based on the imperative paradigm but work exactly like shell script where they are compiled and then executed. R! belongs to the open source community, allowing its code to be freely available. Thus the source code of the programs developed by the users is accessible and not bounded by confidentiality agreement.

4.1 System Architecture

The architecture of the prediction module is derived from a compiler-based approach to help it develop an understanding of the R jobs submitted to it. This means that the source code of an R job (or script) will be parsed and tokenized similar to the way a compiler does when it checks for errors in a program. Then the execution time of the tokens found in the R program are obtained from the database and combined using mathematical calculation to predict the execution time of the entire R program.
The proposed system architecture consists of four interdependent layers: Application Selector, File Parser, Code Evaluation Engine and Predictor Engine as shown in figure 3. The architecture follows a bottom-up approach, whereby it starts from the bottom layer and moves layer by layer until it results in an estimated execution time of the input job. Each of the layers plays a significant role in giving an accurate prediction of the execution time.

a) **Application Selector** – The first layer consists of the application selector. When a user submits a job, the program will first go through this layer to identify the application used to write the job. A job submitted by user can be classified depending on the software (such as Matlab, Maple or Microsoft Visual) used to write the job. Currently the prediction module only has the capability to parse and tokenize programs written using R software. In the scenario where the prediction module will be extended to predict completion time of jobs written using different

software, then the parser module loader will load the appropriate file parser depending on the type of incoming job.



© 2008 MIMOS Berhad. All rights reserved

Fig 3. Overall System Architecture

b) **File Parser** – The second layer consists of 3 components; lexical tokenization, keyword counter and grammar parser. The approach used to develop this layer was guided and adapted from compiler design and constructions tools such as Lex / Yacc. These tools help one to write programs that transform structured input which means dividing the input into smaller units and discovering the relationship amongst these units [19]. In the context of R programs, these units can be names, constants, strings, operators, punctuation and so forth. The process of dividing the program into units (usually called *tokens*) is known as lexical analysis. The keyword counters in this layer keeps track of the number of basic operators (such as addition, subtraction, multiplication and exponents) found and also creates a list of the functions and iterations found in the script. The grammar parser on the other hand contains a set of rules that enables the prediction module to understand the structure of the conditional and iteration statements.

c) **Code Evaluation Engine** – This layer of the prediction module comprises of the metric and complexity analyzer module. Software metrics are proposed to measure the complexity of software artifacts such as functions, classes and whole program. Complexity in software can arise from a program's structure as well as application domain in which the software is used [20]. In the case of the prediction module the metric analysis of an incoming R! program is carried out during the parsing phase where the number of operators and built-in functions found in the script are counted. Also, this layer measures the complexity of nested for loops as well as nested conditional statements by identifying the parent and child statements.

d) **Predictor Engine** – The final layer in the proposed architecture has two components, one is a time completion calculator which is basically a program for measuring the time taken to execute the entire program. It makes use of the tokenized version of the R script and computes the overall predicted execution time for a particular job based on the benchmarked data (of operators, built-in functions and overheads) stored in the statistical database. The predictor engine analyzes the tokenized script file line by line, identify the benchmarked time for each token and finally compute the accumulated time of all the identified tokens in the file. The statistical database contains the execution time of characterized functions and operators which are commonly used in R script files. The statistical database contains the execution time of characterizing the functions and operators commonly used in R script files. This procedure was done by creating script files for each of the commonly used operators and built-in functions. These files mimicked how the operators and functions were typically used in R scripts along with timer function to obtain the execution time of these operators and functions. Once an R script has gone through all the layers of the prediction module the system will display an estimated execution time of the script.

4.2 Handling Complexity in Programs

There are several general characteristics which contribute to the complexity of programs. Our approach to handling complexity in programs is adapted from three different aspects of measuring software complexity: the lines of codes (LOC), identification and enumeration of distinct operators and operands as discussed by Halstead and the nesting depth as mentioned by McCabe.

Halstead states that the amount of computation carried out by a program is based on the number operators and operands found in the program. Operators can be "+" and "*" whereas operands consists of numbers of literal expression, constants and variables. The McCabe metric, on the other hand, assumes that program complexity is related to the number of control paths generated by the codes [20]. The complexity of the codes is related to the number of decision and control statements found. As most

non-trivial programs predominantly consist of nested conditional statements or iterations, we focus on handling the complexity of nested loops to improve the prediction accuracy of the module. Predicting the execution time of an entire program with these nested loops and conditional statements involves complex computations. As the level of nesting of the loop grow, the computational steps increases as well. Our approach to handling the complexity of nested loops involves breaking down the nested loops into separate blocks identified by the start line and end line.

The calculation of the execution time starts with the innermost loop, followed by its parent loop and this step is repeated until the outermost loop is reached. One iteration of the parent loop includes the complete number of iterations of the child loop. Depending on the level of nesting found in the loops, the calculation of execution time increases multiplicatively. The recurring calculations lead to the following simplified equation:

$$N = \sum_{m=1}^{(l-1)} \left[\prod_{i=1}^{m} n_i \right] \Delta_m + \prod_{j=1}^{l} n_j \left(\Delta_l + P(t_1) \right) \qquad (1)$$

Where, N is the total execution time for the entire job, Δ_1 is the time taken between the starting of one loop and the starting of the successive loop. $P(t_1)$ is the amount of time needed to execute the innermost loop. The calculation of the execution time of the outermost loop is equivalent to the time taken to execute the entire nested loop contained within it. This issue of nested loops is handled in the complexity analyzer and in the time completion calculator components of the proposed prediction module.

4.3 Implementation

The execution time of a program varies significantly when it is run across different platform. This is due to the different specifications of each machine that leads to wide variation in execution time. Therefore in our work, all tasks starting with benchmarking of operators, operands and functions to getting the actual executing time are all done using the same machine. This is to minimize the variation and deviation in predicting and then obtaining the actual execution time, thus leading to better accuracy of prediction. Even though the prediction module is machine dependent, the proposed architecture allows cross-platform prediction to be made with relative ease. Thus if prediction of execution time needs to be made on a machine with a different architecture, only the benchmarked scripts need to be run on this machine to obtain the execution time of identified tokens. The benchmarked data are then stored in the statistical database for future prediction.

The type of CPU of the machine used throughout this project cycle was AMD Athlon 848 (2X2 Core) which was running on CentOS 4.4 operating system. The CPU speed was 3200 GHz with 2048 MB of RAM. As for the development of the prototype, Java was used to write the functional codes using Eclipse version 3.2.2 as the IDE. MySQL server version 5.0 was used to store the database involved in this work. This prediction module runs on a unix-based environment and is wrapped around an API (Application Programming Interface) implementation. To interact with this module, the API has to be utilized. The API wrapper was developed using Java Server Page which was later used during the testing phase of the prediction module.

5.0 TESTING AND EVALUATION

Two main criteria were used when testing the prediction module: firstly, checking whether each component functioned correctly and secondly, checking the prediction accuracy of the execution time of the R scripts. Beginning with the parsing phase until the predicted execution time is output to the console, the prediction module was inserted with output statement to see how the R script was processed. Thus one could see the tokenized version of the script, followed by an output which identifies the complexity in the nested loops and conditional statement. Then gradually the time to execute each block or token of the script is added up. Finally the complete execution time of the script is displayed. All these output statements were later suppressed once the testing phase of the prediction module was completed. As for the second criterion, the module is expected to provide a prediction accuracy of 80%, under a Normal Distribution. In the Normal Distribution, 68% of the sampling lies within the first standard deviation. A sampling test is shown in the following example:

$T_{estimated}$: Time that was estimated by the prediction module
T_{actual}: Time taken for actual estimation

$$\textbf{Accuracy Error} = \left| \frac{T_{estimated} - T_{actual}}{T_{actual}} \right| \times 100\%$$

Accuracy = 100% - Accuracy Error

Example:

$T_{estimated}$: 100 days T_{actual} : 110 days

Accuracy error = (110 – 100) / 110 * 100 = 9.09 %

Accuracy = 100 – 9.09 = 90.91%

Hence, we can say that this is a successful estimation as the accuracy of prediction is 90.91%, which is more than the 80% of the desired goal. Under Normal Distribution with one sigma or one standard deviation of error, for a total sample of 100 test cases, our prediction module should be able to predict execution time for more than 68 jobs with more than 80% accuracy.

5.1 Evaluation of the System

For testing purpose, a total of 60 R! scripts were taken at random as the test cases. The estimated time to run each of these 60 test cases was predicted using the wrapper developed. The predicted time was then compared with the actual execution time for each job. Only predicted time that falls within the range of 80% -120% of the actual execution time is considered successful. Otherwise, the prediction is considered to have failed. Table 1.0 shows the summarized result of the testing phase.

Table 1: Summarized results of the testing phase

Accuracy	Count	Total	Percentage of jobs
90-100 %	24	Total > 90% Accuracy	40
80-89 %	20	Total > 80% Accuracy	33.33333333
70-79%	2	% > 80% accuracy	73.33333333
60-69%	2	Total > 60% Accuracy	48
50-59%	0	% > 60% accurcay	80
40-49%	3	Total > 40% Accuracy	51
30-39%	4	% > 40% Accuracy	85
20-29%	0	Total > 20% Accuracy	55
10-19%	3	% > 20% Accuracy	91.66666667
0-9%	0	Total > 0% Accuracy	58
less than 0%	2	% > 0% Accuracy	96.66666667

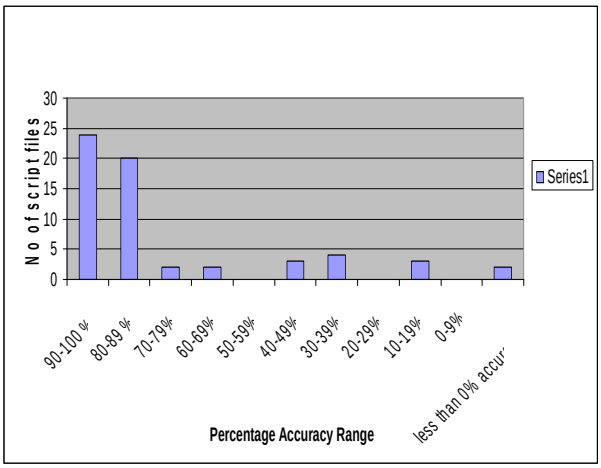


Fig. 4 Accuracy Chart

The result of the testing phase shows that 44 out of 60 R! scripts were predicted with more than 80% accuracy, which means the testing objective was met. The table and figure above shows that a total of 73.3% of the entire test cases was predicted with more than 80% accuracy. Out of the 73.3%, 40% of the files were predicted with more than 90% accuracy. Only about 3.33% of the entire files were predicted with less than 0% accuracy. This is probably due to the limitations of the prediction module where it does not cater for situation such as the occurrence of an indefinite while loop.

6.0 CONCLUSION

The research problem for our project arose from the need to optimize resource usage by assisting scheduling and to enable advance reservation in a grid environment. A solution to this problem will lead to improvement in advance scheduling for resource allocation as well as hasten the transition of the research-driven grids to commercial grids. Our project aimed to develop a prediction module which estimates the execution time of R scripts by using a combination of static analysis, analytical benchmarking and compiler-based approach. Usually this meant that the source code of the script underwent a static and a dynamic analysis, to extract a profile of the script which was then used to predict the completion time of the script. Even though our work focused on predicting execution time of programs written using R!, but the system architecture and techniques are robust enough such that it allows adaptability and flexibility when applied to other program written using imperative paradigm. It is believed that if a solution can be found for predicting the execution time of programs written using specific software as proof of concept than a generic method can be

abstracted from it and applied to a whole category of programs.

6.1 Limitations

There are several limitations to our current work. As the execution time of a program depends greatly on the machine specification that runs the program, our work is platform specific. Our current work also does not cover program written using the object oriented paradigm such Java and data-feed oriented programs such as Blast. Instead it focuses on jobs written using R!, but the architecture, methods and process applied are flexible enough to be adapted and applied to other jobs written in various imperative paradigm. Another assumption made throughout the project life cycle is that jobs will not be pre-empted or interrupted by other jobs. The dedicated consumption of CPU is assumed so that the effect of processors resource management can be reduced, thus leading to better accuracy in our prediction.

ACKNOWLEDGEMENT

We gratefully acknowledge the support of Dr Lai Weng Kin throughout the project.

References

[1] X. Che, L. Hu, D. Guo, K. Tang, D. Hu, Information Service Prototype System for Run- time Prediction of Grid Applications, 1-6, 2007
[2] M. Irving, G. Taylor, P. Hobson. Plug in to Grid Computing, IEEE Power and Energy Magazine, 2, 2004, 40-44.
[3] J. Nabrzyski, J. Schopf and J. Weglarz, eds. Grid Resource Management – State of the Art and Future Trends, Kluwer Academic Publishers, 2003.
[4] P. Plaszczak, R. Wellener Jr., Grid Computing: The Savvy Manager's Guide (San Francisco, CA: Morgan Kaufmann Publishers, Elsevier Inc, 2006).
[5] Luvisetto, Grid Middleware Technology, http://www.bo.infn.it/alice/introgrd/introgrd/node 7.html, 2006
[6] R. Buyya, S. Chapin, D. DiNucci. Architectural Models for Resource Management in Grid, www.buyya.com/papers/gridmodels.pdf , 2000
[7] T. El-Ghazawi, K. Gaj, N. Alexandridis, F. Vroman, N. Nguyen, J.R. Radzikowski, P. Samipagdi, and S.A. Suboh, A Performance Study of Job Management Systems, Concurrency and Computation: Practice & Experience, Vol 16, Issue 13, John Wiley & Son, 2004
[8] What is MYREN? http://www.myren.net.my/?g=A&c=2006111632 0062658172169882&a, 2005
[9] The TEIN2 Network, http://www.tein2.net/server/show/nav.622 , 2003- 2004
[10] The GEANT2 Network, http://www.geant2.net/server/show/nav.740, 2003-2004
[11] KnowledgeGrid Malaysia, http://knowledgegrid.net.my/index.jsp?p=ab, 2007
[12] The European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies D.RMS.06-Review of Performance Prediction Models and Solutions, CoreGRID - Network of Excellence European Grid Research, 2006, http://www.coregrid.net/mambo/content/view/43 1/295/
[13] R. Zhang, Z. Budimlić, K. Kennedy, Performance Modeling and Prediction for Scientific Java Applications, IEEE, 2006.
[14] A. Attanasio, G. Ghiani, L. Grandinetti, E. Guerriero, F. Guerriero, Operations Research Methods for Resource Management and Scheduling in a Computational Grid: a Survey, GRID COMPUTING: The new frontier of High Performance Computing, Advances in Parallel Computing, 14, 2005, 53-81.
[15] M.A. Iverson. F. Özgüner, L. Potter, Statistical Prediction of Task Execution Times through Analytic Benchmarking for Scheduling in a Heterogeneous Environment, IEEE Transactions on Computers, vol. 48, no. 12, Dec. 1999.
[16] D. Binkley, Source Code Analysis: A Road Map, Future of Software Engineering (FOSE'07), 2007 IEEE.
[17] P.P. Chang, S. A. Malkhe, W. W. Hwu, Using Profile Information to Assist Classic Code Optimizations, Software Practice & Experience, 21(12): 1301 – 1321, 1999, URL: http://www.crhc.uiuc.edu/IMPACT/ftp/journal/spe.profile-classic.91.pdf
[18] W. Li, H. Delugach, Software Metrics and Application Domain Complexity, IEEE Proc. of Asia Pacific Software Engineering Conference & International Computer Science Conference (APSEC '97), Hong Kong,1997,513–514.
[19] J. R. Levine, T. Mason, D. Brown., lex & yacc, O'Reilly, United States of America, 1995
[20] L. Marco, Measuring Software Complexity, Enterprise Systems Journal, April 1997