RUNTIME ESTIMATION OF PARALLEL APPLICATIONS IN COMPUTATIONAL GRIDS

P. Hu     Z. Qiao     L.F. Sun     E.C. Ifeachor
University of Plymouth, the United Kingdom

ABSTRACT

The purposes of runtime prediction in grid computing are to provide quality information in order to deliver user-required quality of service and to offer an efficient resource sharing environment. The aim of this paper is to investigate efficient methods for runtime estimation of parallel applications in computational grids from the view of both computation and communication costs. The main contributions of the paper are two-fold. First, we present a learning-based method to predict computation time cost. Second, we propose a new mathematical model to predict overall runtime for executing parallel applications in grids. The model can provide a practical and efficient way to quantify overall runtime of a grid application. Experimental results on a grid test-bed are presented to illustrate the efficiency of the two proposed prediction methods. These models are generic and can be used for grid scheduling and resource management.

Keywords: runtime estimation, computational grid, communication cost, computation cost, overlap.

INTRODUCTION

Grid computing offers large-scale distributed systems by uniting geographically located computation and data storage resources via existing and future global networks. It is expected to solve a number of computing problems in new and emerging scientific applications, such as computational and data intensive areas of biomedical informatics.

Grid resources are shared. The amount of resources available to a given grid application and associated performance levels may fluctuate dynamically from time to time. The purposes of runtime prediction in grids are to provide quality information in order to enable users to estimate service levels they can apply, to allow grid administrators to manage the assignment of application loads to resources in order to guarantee quality of service, and to offer an efficient resource sharing environment.

In virtual environments, runtime of a grid application consists of two main aspects, computation and communication. For the prediction of computation costs, previous studies addressed certain issues, such as code analysis, Hou et al (9), analytic benchmarking/code profiling, Yang et al (10) and Yang and Gerasoulis (11) and statistical prediction, Iverson et al (8), Iverson et al (12) and Xu et al (13). However, their predictions are restricted within certain conditions. For example, they are either targeted at specific applications, Ripeanu et al (7), or do not take into account the impact of changing and/or multiple application features, (8) and (12). As a result, they may not be appropriate for different grid applications.

For overall runtime prediction in grids, most existing methods, Wolski (1), Liang et al (2) and He et al (3), are purely based on the consideration of computational power. In reality, grid performance is not just determined by computational capacity. Considering the geographical characteristics of grids, communication time cost also contributes considerably to overall runtime, especially in the transmission of large amount of data among distributed grid resources.

The motivation of this research is to find efficient methods for predicting runtime from the view of both computation and communication costs. The main contributions of the paper are two-fold. First, we present a learning-based method for prediction of computation time cost. Second, we propose a mathematical model for prediction of overall runtime for executing parallel applications in grids. The model can provide a practical and efficient way to quantify the overall runtime of a grid application. Experimental results on a grid test-bed are presented to illustrate the efficiency of the two proposed prediction methods. These models are generic and can be used for grid scheduling and resource management.

The remainder of this paper is organised as follows. First, we review previous work on runtime prediction. Then we describe our new methods of prediction of computation time cost, communication time cost and total (overall) time cost in grid computing. Further, we present our experimental work and analyse the test results of the proposed methods. Finally, we conclude the paper.

BACKGROUND AND RELATED WORK

In this section, we review previous work on prediction of computation, communication and total time cost in grid computing. There are two types of communications in grids, intra-processor and inter-processor communications. In this paper, we mainly focus on inter-processor communication time cost and consider intra-processor communication as a part of computation process.

Computation Time Cost (CPC)

Computation time cost has been widely studied in previous...
work, such as (8), (9), (10), (11), (12) and (13). From the literature, it is mainly related to certain aspects, such as processor power, program codes and application characteristics.

There are three major classes of solutions to the prediction of CPCs: code analysis, analytic benchmarking/code profiling and statistical prediction. Code analysis (9) uses a given task’s source code to estimate CPCs. It is typically limited to a specified code type or a limited class of architectures. While employing a heterogeneous computing architecture, it seems not applicable (8). Benchmarking/code profiling models, (10) and (11), estimate CPCs by defining a number of primitive code types and determining the composition of a task. It can determine the relative performance difference between machines. However the major disadvantage of this method is that it cannot easily compensate for variations in the input data set (8).

Statistical prediction, (12) and (13), complements analytical benchmarking. It predicts CPCs by modelling the time duration as a random variable and calculates its expectation based on a set of past observations. Statistical methods have certain advantages, such as ability to compensate for parameters of the input data and the freedom from any direct knowledge of internal designs of algorithms and computation capacities of processors. However, statistical prediction is limited to rectangular regions of the solution space and difficult to adapt to changing and/or multiple application features. This motivated our work in learning-based method to predict computation time cost, which will be presented later.

Communication Time Cost (CMC) in Reliable Data Transmission

CMC is another major aspect which will affect the grid performance. The level of CMC mainly depends on network conditions (e.g. bandwidth size, packet loss rate and delay) and the protocols used to exchange data among grid resources (e.g. transport layer Transfer Control Protocol (TCP), and network layer Internet Protocol (IP) and Resource ReserVation Protocol (RSVP)).

CMC in reliable data transmission can be predicted by methods, such as statistical estimation and analytic model. Statistical estimation, Sodhi et al (14) uses information of historical communication behaviours to predict CMCs. It can perform predictions with reasonable accuracy in stable network traffics. Freedom from direct knowledge of communication parameters is its major advantage. However, while data is transmitted over unstable networks, the characteristic of retransferring lost packets in reliable data transmission causes variations of CMCs. The use of statistical estimation can thus result in inaccurate predictions.

Analytic models, Padhye et al (4), Cardwell et al (5) and Sikdar et al (6), predict CMCs by using traffic process analysis. Communication impairments (e.g. packet loss rates, delay levels and bandwidth sizes) are the variables which are utilised for predictions. By giving specific transmission services (e.g. data transmitted via TCP-based traffic over IP networks), those methods can estimate CMCs under any network conditions. Due to the advantage of using analytic model, we use it in our study for prediction of communication time cost.

Total Time Cost (TTC) in Parallel Applications

TTC is a combination of CPC and CMC. In general, three aspects have influences on TTC. They are resource distribution styles (i.e. the quantity and quality of grid resources arranged in each stage of executing a distributable task), levels of computation power and network conditions. In this paper, we consider mainly prediction of TTC in parallel applications.

In a computational grid application, a job can be divided into a number of tasks. Each task contains a specified amount of application data. The data is packetised as messages and transferred from databases or users to computational resources for execution, via local and/or global networks. Note that a message may contain several tasks and a task may also be constituted by several messages, here we only consider the former scenario.

If we ignore outgoing CMCs in a computational cluster/node, TTC on a given cluster/processor for executing an application can be classified into four modes, no overlap, complete overlap, incomplete overlap and fluctuating overlap, as shown in Figure 1. The white cubes in Figure 1 are the incoming CMC for each message, resulting from receiving data from data resources. The gray cubes are the time required to execute each message (the time is abbreviated as ET in this paper). ET includes both CPCs of appointed processors and CMCs between those processors and local cluster administrator if communications between them are required.

Figure 1(a) shows a diagram of no overlap of communication and computation. Before running a message, computational resources must wait for the completeness of the transmission of the data it contains. This results in inefficient use of computational resources and the longest execution time required for a grid job. Figure 1(b) descripts complete overlap mode in which the ET is longer than the CMC for each message. Except the CMC for the first message, all CMCs are thus hidden in ETs. Figure 1(c) presents the incomplete overlap mode in which CMC is higher than ET for each message. It can also cause the inefficient use of computational resources. Figure 1(d) illustrates the scenario of fluctuating overlap.
This occurs when there are changes in levels of computation power, and/or contexts of messages (e.g. increase or decrease of the amount of processors) and/or network conditions (e.g. variations of bandwidth sizes).

Total time cost is important because it gives an overall view of the performance for executing a grid application. However, there is no existing method/model to predict overall runtime in the case of overlap of computation and communication in the literature. This motivated our work to develop models for prediction of overall runtime, which will be described in the following section.

**RUNTIME ESTIMATION AND MODELLING**

In this section, we present the proposed learning-based model and mathematical model to predict computation and total time cost, respectively.

**Learning-based Method for Prediction of Computation Time Cost**

The general concept of the learning-based method for prediction of CPCs is presented in Figure 2. As can be seen, a specified number of history CPCs for executing a group of application data are captured from the monitor points located in the related computational resources. The relationship weights between each application feature and measured CPCs are then trained in order to generate trained models to predict CPCs. After a trained model has been generated, when the similarity between a group of predicted CPCs and corresponding measured CPCs do not reach a specified target error range, the weights between application features and CPCs will be readjusted based on measured CPCs as retraining. A new trained CPC prediction model are then generated and gives predicted CPCs.

![Figure 1. Total time cost](image1)

![Figure 2. Learning-based method for prediction of computation time cost](image2)

A more detailed learning process is illustrated in Figure 3. In the figure, appropriated features are extracted from grid applications and those serve as inputs for training (e.g. message size and application data characteristics). Based on measured CPCs, the relationship between each feature and measured CPCs is then analysed by learning systems (e.g. neural network or neural-fuzzy combined system). If the similarity between predicted CPCs and the target CPCs reach a specified error range, a trained CPC prediction model is then generated.

![Figure 3. Concept of learning process](image3)

The learning-based method has the ability to find how each application feature affects CPC, and can also self-adjust when application feature changes. The number of application features in the method is almost unrestricted, so it can be used to predict CPC in most grid applications.
Modelling Total Time Cost in Parallel Applications

Modelling TTC requires two variables, CMC and ET. For CMC, In general, for transmitting a given size of application data, CMC can be briefly described as

\[ CMC = \frac{D_{size}}{S_{ave}} + T_{other} \]  

(1)

where \( D_{size} \) is the total size of data, \( S_{ave} \) is the average transmission rate and \( T_{other} \) is the delay including the time for initialising the communication between two end-points etc. Ignoring \( T_{other} \), the time required to transmit a given size of data, CMC, may then be simplified as

\[ CMC = \frac{D_{size}}{S_{ave}} \]  

(2)

Since \( D_{size} \) is given, we thus only need to consider the variable \( S_{ave} \).

The value of \( S_{ave} \) is related to two major categories of parameters, communication protocols and network impairments. To predict \( S_{ave} \) in reliable data transmission, we would rather use analytic model than statistical estimation, as discussed in the previous section.

If analytic model is employed, the average transmission rate \( S_{ave} \) can first be modelled by relevant analytic models while communication protocols are given. Then based on captured values of network impairments, \( S_{ave} \) can be calculated. For instance, if given communications are TCP-Reno based, we may first use the algorithm proposed by Padhye (4) to express \( S_{ave} \), as

\[ S_{ave} = \begin{cases} 
\frac{1 - \frac{p}{p} + E[W]}{RTT} & \frac{1}{\frac{1}{p} + E[Z]} + \frac{1}{\frac{1}{p} + E[Z]} \\
\frac{1 - \frac{p}{p} + W_{max} + \frac{1}{\frac{1}{p} + E[Z]} + \frac{1}{\frac{1}{p} + E[Z]}}{RTT} & \frac{1}{p} + \frac{1}{\frac{1}{p} + E[Z]}
\end{cases} \text{ if } E[W] < W_{max} \]

(3)

where \( RTT \) is the round trip time, \( p \) is the loss rate, \( W_{max} \) is the maximum window size, \( b \) is the number of successive data segments to be received before sending an acknowledgement to the send, and other expected values. The predicted value of \( S_{ave} \) can then be calculated using the values of those variables obtained from protocol specifications and captured by network monitor tools.

For ET, if a message is only computed by a single process, ET can be equal to CPC. If a message is executed on a cluster, it will relate to local data CMCs, scheduling skill, CPCs on each processor etc. Since the intention in this paper is not to investigate ET, we assume that ET is known or predictable.

If running \( m \) messages on a cluster/processor, the total CMC and ET can be expressed as \( \sum_{i=1}^{m} CMC_i \) and \( \sum_{i=1}^{m} ET_i \), respectively. In the case that communication and computation is not overlapped, the TTC can be easily expressed as the sum of the total CMC and the total ET, as

\[ TTC = \sum_{i=1}^{m} CMC_i + \sum_{i=1}^{m} ET_i \]  

(4)

In the case that communication and computation are completely overlapped, TTC can be expressed as the first message CMC plus the total ETs for \( m \) messages, as

\[ TTC = CMC_1 + \sum_{i=2}^{m} ET_i \]  

(5)

In the situation when communication and computation are incompletely overlapped, TTC can be expressed as the total CMCs plus the final message ET for \( m \) messages, as

\[ TTC = \sum_{i=1}^{m} CMC_i + ET_m \]  

(6)

While communication and computation are fluctuating overlapped, the key of prediction of TTC is to find the message while it is the final time computation of a message must wait for the end of transmission of it. If we refer the message as \( n \)th message, TTC can be expressed as the total CMC of former \( n \) messages plus the total ET from \( n \)th to \( m \)th messages, as

\[ TTC = \sum_{i=1}^{n} CMC_i + \sum_{i=n+1}^{m} ET_i \]  

(7)

To find the message \( n \), it takes two stages, the first time incomplete overlap and the subsequent time incomplete overlaps. The first time incomplete overlap happens at \( x \)th message can be defined as the total CMC of the former \( x \) messages is larger than the sum of CMC of the first message and the total ET of the former \( x \)-1th messages. The subsequent time incomplete overlaps happens at \( y \)th message can be detected when the total CMC of the former \( y \) messages is larger than the sum of the total CMC of the former \( x \) messages and the total ET from \( x \)th message to \( y \)th message, where \( x \) is the message when last time incomplete overlap happens. The procedure of finding the message \( n \) is presented in Figure 4.

EXPERIMENTS

This section describes our experimental work on implementation and validation of the two proposed runtime prediction methods.

Test-bed Setup

The structure of our grid test-bed is presented in Figure 5.
It consists of a database node with one 3GHz Pentium 4 Linux machine and a compute cluster, which comprises one 450MHz Pentium 2, one 899MHz Pentium 3, one 2.4GHz Pentium 4 and one 3GHz Pentium 4 Linux machines. The Globus version 3.2 is used to implement the grid environment.

The database node and the compute cluster are connected by a network emulator, Shunra Storm (19), in order to emulate world wide communications. Shunra Storm emulates the network, including multiple endpoints, gateways, WAN clouds, and connections, and sets their network parameters. It can also capture traffic information between two end-points.

TCP-Reno over IP is used in the data traffic. Maximum Segment Size (MSS) is set to 1448 Bytes. Timeout (TO) for a loss indication is set to 1 second.

CMCs between the database node and the compute cluster are obtained from Shunra Storm traffic records. CPCs are captured from monitor points built in the application programs. From measurement, the use of those monitor points may cause further delay of up to an average of 0.1ms per loop of running a message on 450MHz Pentium 2, and less than 0.05ms on other more advanced machines in the test-bed.

### Protein Sequence Alignment in Grid

In order to present our experimental work and validate the concepts of the two proposed methods, we use protein sequence alignment as an example of grid applications for our experiments. Protein sequence alignment is a common and often repeated task in biology. The analysis process consists of finding similarities between a particular query sequence and all the sequences of a bank.

The Smith-Waterman algorithm, Smith and Watermen (15), is the protein sequence alignment algorithm we used in our experiments. We modified the corresponding program code in JAiligner (16) and implemented the software in the grid test-bed using java.

In the experiments, we used three protein sequence banks, “domo.dat”, “Oryzasativa.Protein.fasta” and “Arabiopsisthaliana.Protein.fasta”. The sequence bank “domo.dat” was downloaded from BIOSUPPORT protein genetic database (17) and another two were downloaded from PlantGDB (18). Some information of the three sequence banks is presented in Table 1.

#### Table 1. Description of selected protein sequence banks

<table>
<thead>
<tr>
<th>Protein sequence bank name</th>
<th>Size (Mbytes)</th>
<th>Sequence amount</th>
<th>Sequence length</th>
</tr>
</thead>
<tbody>
<tr>
<td>domo.dat</td>
<td>16.8</td>
<td>71687</td>
<td>10</td>
</tr>
<tr>
<td>Oryzasativa.Protein.fasta</td>
<td>27</td>
<td>60542</td>
<td>2</td>
</tr>
<tr>
<td>Arabiopsisthaliana.Protein.fasta</td>
<td>32.2</td>
<td>70638</td>
<td>1</td>
</tr>
</tbody>
</table>

### Implementation of Performance Prediction Methods

#### Implementation of learning-based method for prediction of communication time cost.

A neural network model was used to implement the proposed learning-based method. The model is constructed as a two-layer feed-forward neural net architecture and the standard back-propagation learning algorithm. From input side to output side, there are one hidden layer and one output layer. The hidden layer is implemented by hyperbolic tangent sigmoid transfer function, and the output layer is implemented by linear transfer function. The set of multiple layers provides ability to predict non-linear relationship between inputs (application features) and corresponding CPCs. Figure 6 presents the architecture of determined artificial neural network model for CPC prediction of grid-enabled protein sequence alignment.

The number of inputs is application-dependent. In our
experiments, we use the size of each message and the corresponding number of protein sequences it contained as the two inputs to the neural network model for simplicity. From previous work on neural network, Peng et al (20) and Masters (21), and with the consideration of real-time prediction issues, the number of hidden neurons is set to two.

![Image](64x468 to 287x638)

**Figure 6. Architecture of determined artificial neural network model**

**Total time cost modelling.** To model TTC, two variables are required, as ET and CMC. We directly use the TCP throughput modelling algorithm (4) (Equation (3)) to predict CMC in TCP-Reno employed traffics. Since data transmissions in our experiment are long-lived, slow-start phase is ignored. The relationships between transmission delay and message size as well as TCP segment size are not considered in this paper.

For ET, it is related to a variety of aspects, such as local data CMCs, scheduling skills and CPCs on each processor. As our intention of this paper is not to analyse ET on a cluster, we simply use computational power of a single processor to imitate the computation capacity of a cluster.

**Experimental Results and Analysis**

To perform CPCs on the processors, we use a protein sequences with length 200 to find the most similar sequences in the three sequence banks, respectively. Messages where each contains random number of protein sequences were generated in order to avoid invaluable predictions which may result from the use of invariables of inputs to the neural network set.

In the learning model we determined, a training sample is referred to two input variables with an output variable. Two input variables are the size of each message and the corresponding number of protein sequences it contained, respectively. The output variable is the related measured CPC. All training samples were generated from the first 10% of protein sequences in each sequence banks.

Normalized percentage error was used to measure the prediction accuracy, which is defined by

$$P_{\text{Error}} = \frac{\sum_{i} \left| M_i - P_i \right|}{m \cdot M_{\text{ave}}}$$

(8)

where $P_i$ is the predicted value and $M_i$ is the measured value of $i$th prediction. $M_{\text{ave}}$ is the average of measured values and $m$ is the total number of predictions.

Figure 7 shows the performance of CPC prediction using determined neural network set. We observed that all prediction errors are within 3% when the three processors and three protein sequence banks were employed. There are no clear relationships between prediction accuracy and different processors, and between prediction accuracy and different protein sequence banks used. This may reflect the effective prediction of CPC by the neural network set, at least under this scenario.

![Image](64x468 to 287x638)

**Figure 7. Performance of CPC prediction using determined neural network set**

We further evaluated the performance and characteristics of our TTC prediction method. Figure 8 shows the numerical values of the predicted TTCs and the measured TTCs under different network conditions and different computation power levels. We emulated loss rate from 0 to 4% on bandwidth 512kbps with RTT 180ms and bandwidth 768kbps with RTT 120 ms network traffics, to perform different network conditions, respectively. Computation power levels were simulated by execution of the selected sequence with others in sequence bank “domo.dat” on two machines with CPU 899MHz and 2.4GHz, respectively. The first 10% of protein sequences in the sequence bank were used for training in order to predict ETs.

In Figure 8, the numerical values of the predicted TTCs exactly follow the trends of the measured TTCs. When ET is larger than CMC, TTC is in complete overlap mode, as presented in Figure 8(a) and 8(b). The TTCs are thus equal to the first message CMC plus the ETs for all messages. When ET is smaller than CMC, TTC is in incomplete
overlap mode. The TTCs are thereby equal to the CMCs for all messages plus the ET for the final message.

We further examined the characteristics of proposed TTC modelling method on prediction of TTC in fluctuating overlaps. Table 2 illustrates the prediction errors of TTC with other related parameters which can affect the performance of TTC prediction. In the table, the bandwidth column represents the bandwidth varies between two numerical values every 30 seconds. The left numerical value in the bandwidth column is the bandwidth size at the beginning of execution of the grid job. The L_M column represents the errors of prediction of the message n. A negative value of L_M implies the predicted sequence number of the message n is smaller than its actual sequence number. A positive value implies the predicted sequence number is larger than its actual sequence number.

From experiments, we found that underestimated CMC and ET will result in underestimated TTC whereas overestimated CMC and ET will lead to overestimated TTC. We also found that the accuracy of prediction of the message n is related to the prediction performance of both ET and CMC. If the numerical value of the deviation of predicted ET is smaller than that of predicted CMC, the predicted sequence number of the message n will be smaller than its actual sequence number and vice versa.

CONCLUSION

In this paper, we have investigated computation time cost, communication time cost and total time cost in grid computing. We have proposed two novel methods on prediction of computation time cost and total time cost, respectively. We conducted performance test in a grid testbed. Experimental results show that the proposed methods can efficiently predict computation and overall runtime, respectively.

Future work will focus in two directions. First, we will investigate the relationships between computation time cost and application features for other grid applications. We will compare the proposed method with other existing methods based on different grid applications. Second, we will continue the investigation on the effects of communication on overall performance degradation of running computational jobs in grids and further the interference of communication on computation

ACKNOWLEDGEMENT

The work reported here is supported in part by BIOPATTERN EU Network of Excellence. EU Contract 508803.

REFERENCES


17. BIOSUPPORT. http://bioinfo.hku.hk/
18. PlantGDB. [http://rdp.cme.msu.edu/misc/resources.jsp](http://rdp.cme.msu.edu/misc/resources.jsp)


![Figure 8. Total time cost in complete and incomplete overlap](image)

**Table 2. Prediction of total time cost in fluctuating overlaps**

<table>
<thead>
<tr>
<th>Computation and communication conditions</th>
<th>Prediction error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ET</td>
</tr>
<tr>
<td>CPU 2.4GHz Pentium 4</td>
<td>-2.76%</td>
</tr>
<tr>
<td>256/1024</td>
<td>-2.76%</td>
</tr>
<tr>
<td>1024/256</td>
<td>-2.00%</td>
</tr>
<tr>
<td>CPU 899MHz Pentium 3</td>
<td>-2.00%</td>
</tr>
<tr>
<td>128/512</td>
<td>-2.58%</td>
</tr>
<tr>
<td>512/128</td>
<td>-2.58%</td>
</tr>
</tbody>
</table>