Evaluation of execution time of mathematical library functions based on historical performance information

Maciej Brzezniak, Norbert Meyer

Poznan Supercomputing and Networking Center
61-704 Poznan, Noskowskiego 12/14, Poland
{maciekb,meyer}@man.poznan.pl

Abstract. The paper presents the concept of a mechanism for predicting the execution time of mathematical library functions. The technique bases on the analysis of historical information concerning the performance of previous executions. The main application of the proposed mechanism is the prediction of the execution time of mathematical library functions in the Grid RPC systems. It can also be used in other heterogeneous and distributed environments where some parts of code are often executed and the prediction of execution time is crucial for high performance of computations and/or load balancing of the resources.

1 Introduction

The performance of computations led in cluster and distributed environments strongly depends on the accuracy of estimation of task execution time on particular nodes. While such evaluation is relatively simple in the dedicated laboratory installations, it is very difficult in real production systems. The difficulty results from the distributed, heterogeneous, shared and dynamic nature of these environments.

It is impossible to design and implement a universal task execution time prediction technique that would be efficient and accurate for various architectures of computing systems. It would require *a priori* knowledge of performance models of all kinds of computing systems, characteristics of particular subsystems and patterns of other (coexisting in the environment) applications' behaviour. Moreover, the dynamic character of production environments (changes of configuration, structure, and management policy) as well as continuous development of computing systems' architectures would require periodic updates of this knowledge. It is also difficult to design the execution time prediction technique for general type of tasks. It would demand knowledge of the task's algorithm, its resource requirements, communication patterns etc. Moreover, the course of a given execution of the algorithm may depend on the task's input parameters and many unpredictable conditions. In general, such information is not available *a priori*.

However, in several applications and environments some parts of code are often repeated. Moreover, several parameters of the algorithm realised by these parts of code are known (e.g. computational complexity). Our idea is to use the information concerning the performance of previous runs of these parts of code to predict the time that is needed for their future executions.

The main application of the mechanism is the prediction of the execution time of mathematical library functions in the Grid RPC [1][2] systems. Grid RPC allows the end-users to launch functions of mathematical libraries installed on remote machines from inside of their applications. The processing model is similar to the traditional RPC. The programmer calls the function by its name followed by a list of input and output arguments. The resources for computations (i.e. the computational node and the mathematical library) are assigned by the central module of the Grid RPC system.

The important feature of processing in the Grid RPC environments is that the particular library functions are repeatedly invoked. Although, the execution time prediction techniques currently used in the Grid RPC systems do not take advantage of that fact [3][4][5].

In our opinion, this feautre of the Grid RPC systems can be exploited. The paper presents the concept of the mechanism that automatically recognises the relation between the mathematical problem parameters, the computing system's state, architecture of the system and the task execution time basing on analysis of the historical performance information.

2 Definition of the problem

Let us define *the mathematical function execution task* T_n. It is an instance of *mathematical problem* P. n is the *size* of this instance. Instance T_n of problem P can be solved using one of the mathematical libraries that are installed on the computing nodes. Each of these libraries implements *the algorithm* A that solves problem P. Evaluation of the execution time on particular machines is necessary, in order to realise a given scheduling policy in the environment.

The problem is how to evaluate the value of *execution time* ($time_{CA}(T_n)$) that is needed to solve instance T_n of mathematical problem P on the given machine C using algorithm A.

3 The existing solutions

The computational complexity theory defines *the computational complexity function* as the relation between size n of instance T_n of problem P and *the number of elementary operations* (N_{eo}) that are needed to solve this instance using algorithm A. Let us call this relation in short *the computational complexity function* f_{ccA}.

$$N_{eoCA}(T_n) = f_{ccA}(n) \tag{1}$$

The execution time prediction mechanisms used in Grid RPC systems exploit the $f_{ccA}(n)$ function. Suppliers of many mathematical libraries specify the f_{ccA} function of the algorithm implementing particular functions. That makes evaluating the number of elementary operations $N_{eoA}(T_n)$ that are needed to solve the instance T_n of problem P using a given version of the mathematical library (implementing the algotihm A) possible.

Evaluating the time that is needed to solve instance T_n of problem P on machine C (using algorithm A) requires calculating the time that is needed to perform a given number of elementary operations using the library installed on machine C:

$$time_{CA}(T_n) = time_C(N_{eoCA}(T_n)). \tag{2}$$

In the existing solutions the time that is needed to perform a given number (N_{eo}) of elementary operations ($time_C$ (N_{eo})) on the computing machine C is computed as:

$$time_C(N_{eo}) = \frac{N_{eo}}{V_{Creal}}, \tag{3}$$

where V_{Creal} is the real processing speed of machine C, i.e. the number of elementary operation that machine C is able to perform in a time unit. Determining V_{Creal} in shared environments is not trivial. It depends on the state (S_C) of computing system C. In the existing solutions, the relation between the state (S_C) of computing system C and its real processing speed V_{Creal} is reflected by *the performance model*. Let the f_{pmC} function represent this model. Then:

$$V_{Creal} = f_{pmC}(S_C). \tag{4}$$

The state (S_C) of computing machine C is a set of parameters of the computing system: the number of processors N, load L, the amount of available memory M and other performance-related information.

$$S_C = \{P, L, M, ...\}$$

The time that is needed to perform a given number of elementary operations (N_{eo}) that are needed to solve a given instance T_n of problem P on machine C using algorithm A can be expressed as (from (2) and (3)):

$$time_{CA}(T_n) = time_{CA}(N_{eoCA}(T_n)) = \frac{N_{eoCA}(T_n)}{V_{Creal}}. \tag{5}$$

Substituting in (5) $time_C(T_n)$ for $time_C$ ($N_{eoC}(T_n)$) (see (2)), the right side of (1) for $N_{eoC}(T_n)$ and the right side of (4) for V_{Creal} we receive:

$$time_C(T_n) = \frac{f_{ccA}(n)}{f_{pmC}(S_C)}. \tag{6}$$

For a given instance of problem T_n value of n is known. Values of elements of the S_C set are predicted (e.g. using the Network Weather Service [8][9]).

3.1 Limitations of the existing solutions

The presented method of evaluating the execution time of remotely invoked functions is used in the Grid RPC systems and other distributed environments, e.g. RCS [6][7]. However, it has the following limitations.

Firstly, the computational complexity functions of some algorithms used in mathematical libraries may be unknown *a priori*. The form of the function or its coefficients may be unspecified by the library supplier. Therefore, it might be impossible to use this function to evaluate the number of elementary operations that are needed to solve instance T_n of problem P using some of the mathematical libraries.

Secondly, the number of elementary operations that is determined using the computational complexity function is a very high-level measure. It allows to evaluate the number of the algorithm steps, but it does not determine the number of machine instructions that implements (on the low level) the high-level algorithm steps. The execution of the same algorithm may require different number processor time, depending on processor architecture, level of binary code optimisation and other unpredictable factors.

Thirdly, the performance models of some computing systems are unknown *a priori*, i.e. at the moment of designing the prediction techniques. Therefore, the usage of these models is limited to the architectures that are known at the moment of designing the prediction mechanisms.

Fourthly, the performance models of computing systems that are used in the Grid RPC systems are relatively simple. They cannot reflect the features of complicated computing systems such as parallel machines, cluster systems etc.

Finally, the method of evaluating the execution time used so far in the Grid RPC systems does not take advantage of the fact that several mathematical library functions may be often invoked.

4 Proposed solution

We propose another approach to the problem of predicting task T_n execution time. Seeing the limitations of the existing solutions, we desist from the usage of the computational complexity function (f_{ccA}) and the performance models of computing systems (f_{pmC}). Instead, we propose another model.

4.1 Definitions

Let us define *instance* $T_{n,OP}$ *of problem* P as the mathematical function execution task. n_T is the *size* of this instance. OP_T includes *other run-time parameters of the remote function call* that can be specified by the programmer (e.g. algorithm version).

Let us also define *work* W_{CT} that has to be done to solve this instance of the problem on *computing machine* C. This work is defined as the amount of time that is spent by the process solving the problem in the user and system mode on machine C. It is not the real time. It can be measured e.g. using standard *Unix* system mechanisms.

Let us also define *function of work* f_{CT} that reflects the relation between size n_T and other parameters OP_T of instance $T_{n,OP}$ of problem P and work W_{CT} that must be done on computing system C to solve this instance of the problem.

$$W_{CT} = f_{CT}(n_T, OP_T) \tag{7}$$

We decided to use the function of work instead of the computational complexity function as the measure of the work that must be done to solve a given mathematical problem on a given computing system. The main motivation is the fact that the time spent by the process in the user and system mode can be measured. On the contrary, it is not possible to measure the number of elementary operations that have been done by the process when solving a given problem on the computing system. We also observed that the system and user time that is needed to execute several parts of code (e.g. mathematical library function) depends not only on the number of elementary operations of the algorithm. It also depends on the other factors, e.g. the degree of binary code optimisation and the quality of the compiler used to prepare the binary code.

Let us define *machine performance model function* g_C. It reflects the relation between the state S_C of the machine C, the amount of work W to be done on the machine and the amount of real time ($timeW$) that is needed to do this work on C.

$$time_W = g_C(W, S_C) \tag{8}$$

The g_C function reflects the scheduling and resource assignment policies in a given computing system.

Let us also extend (in comparison to the existing solutions) the set of parameters considered as the elements of the state (S_C) of computing system C. Let it include: number of processors N, load L, amount of available memory M, number of processes in the run queue R, intensity of I/O operations I and other performance-related information. All this data can be acquired in *Unix* operating systems e.g. using simple user-level commands as *ps*, *uptime*, *sar*.

$$S_C = N, L, M, R, I, ...$$

The extension aim in providing the prediction mechanisms the ability to consider the widest possible range of system state' parameters.

Substituting in (8) W_{CT} for W we receive the formula for computing the real time that is needed to solve the instance $T_{n,OP}$ of problem P:

$$time_{W_{CT}} = g_C(W_{CT}, S_C). \tag{9}$$

For a given instance ($T_{n,OP}$) of the problem, values of n_T and OP_T are known. Values of elements of the S_C set can be predicted using the external prediction systems, e.g. Network Weather Service.

The evaluation of the time that is needed to solve a given instance ($T_{n,OP}$) of the problem P requires knowledge of the form and coefficients of f_{CT} and g_C functions.

4.2 Statements

The form and coefficients of the function of work f_{CT} may be reconstructed by using the experimental data. It requires collecting measurements of works W_{CT} done to solve particular instances $T_{n,OP}$ of problem P with the usage of a mathematical library installed on machine C as well as measurements of the parameters' values (n_T and OP_T) that are describing these task instances. Having collected enough measurements of values of the corresponding n_T, OP_T and W_{CT} quantities makes the reconstruction of the f_{CT} function possible.

The form and coefficients of the machine performance model function g_C can be reconstructed from the experimental data. That requires collecting measurements of the values of the work W for particular tasks as well as the values of real times ($time_W$) that are needed to do these works on system C. The parameters' values that are describing state S_C of system C while doing these portions of work should also be gathered. Having gathered enough measurements of values of the corresponding $time_W$, W, and S_C parameters, makes the reconstruction of the g_C function possible.

4.3 Detailed considerations

The reconstruction of the f_{CT} and g_C functions requires running a large number of test computations. In practice, the cost of the experimental reconstruction of the functions of work (f_{CT}) for all the implementations of a given mathematical functionality that exist in the environment may be unacceptable. Similarly, the cost of reconstructing the performance model functions (g_C) of all computing systems in the environment may be unacceptable.

The designed mechanism is intended to be used for evaluating future execution times of these parts of code that are often used. Therefore, the functions of work f_{CT} of the implementations of mathematical functions that are often invoked must be analysed. Similarly, performance models g_C of the computing systems that are used intensively have to be reconstructed. Therefore, we assume that the reconstruction is done continuously during the Grid RPC system work.

The mechanism has two phases of work. In the initial state – *gathering phase* – prediction mechanism uses Grid RPC-native methods for evaluating the execution time. During these executions, for each implementation of mathematical function that has been used, values of the problem's parameters (n_T, OP_T) and values of the corresponding works (W_{CT}) are collected. Similarly, for each computing system exploited by the computations, values of works W, values of state parameters (S_C) and the quantities of real times that have been needed to do the relevant works are gathered. When the sufficient number of "probes" is collected, the f_{CT} and g_C functions are reconstructed. Then they can be used to predict the future execution times of mathematical library functions, that is the *exploitation phase* begins.

In the *exploitation phase* the execution time prediction is performed with use of reconstructed f_{CT} and g_C functions. In order to discover possible changes of the features of the mathematical libraries and computing systems, the analysis of performance data and reconstruction of the f_{CT} and g_C functions are continuously performed. In addition, the accuracy of the predictions done by using the reconstructed f_{CT} and g_C functions is periodically verified. If it degrades, the mechanism returns to the *gathering phase*.

The processes of reconstructing the f_{CT} and g_C functions have two stages. In case of the f_{CT} function, in the first stage, the subsets of parameters of the problem OP_T that may be significant for the form of the f_{CT} function are selected. Similarly, in the first stage of reconstruction of the g_C function, subsets of parameters of computing system's state that may be significant for the form of the g_C function are selected. It is known that the size (n) of a given instance $T_{n,OP}$ of problem P has influence on the amount of work that should be done in order to solve this instance. However, it is necessary to evaluate the influence of other elements of the parameter set (OP_T) describing the problem instance $T_{n,OP}$ on the value of the work that should be done. The evaluation requires the analysis of the correlation between the parameters of particular problem's instance and values of W_{CT} for the given (constant) size of the problem (n). It is also necessary to recognise what parameters of state S_C of system C have influence of the value of $time_W$ for the given amount of work (W). The evaluation requires the analysis of the correlation between particular elements of the S_C set and values of $time_W$ for the given (constant) work W. As the result of the evaluation appropriate weights are assigned to elements of the OP_T set and the S_C set. These weights can be useful in reconstructing the form and coefficients of the f_{CT} and g_C functions (the second stage of the reconstruction).

The details of methods for reconstructing the f_{CT} and g_C functions are subject of present work. Example considered method for reconstructing the f_{CT} function tries to fit some templates of the f_{CT} function to the experimental data. Templates can include linear and non-linear functions, polynomial and exponential functions as well as the known form of the f_{ccA} function of the algorithm implementing the given mathematical functionality (if specified by the library supplier). The mechanism tries to fit coefficients of template functions to the experimental data concerning executions of the given mathematical library function. Other methods of reconstruction are also taken into consideration.

5 Summary

The paper provides the concept of the execution time prediction mechanism. The proposed solution copes with the limitations of the existing methods. It takes advantage of the fact that particular mathematical library functions are often executed in the Grid RPC systems.

The mechanism is able to recognise the performance-related features of the mathematical functions' implementations even if computational complexity of the algorithm is unknown *a priori*. The mechanism can also recognise the performance characteristics of computing systems. The recognition is based on analysis of historical information concerning the performance of previous executions of mathematical functions on particular computing systems. Additionally, the mechanism is able to discover possible changes of performance-related features of the computing resources (mathematical library functions and computing machines) dynamically as the prediction module works continuously.

In our opinion, the proposed technique may improve the accuracy of prediction of execution time of mathematical library functions in Grid RPC systems. It may be also exploited in cluster and distributed environments, where several parts of code are often executed and the performance characteristics of code parts and computing systems are *a priori* unknown and may change dynamically. We believe that the technique is cost-efficient, since only the features of often-used computing resources are analysed.

The current focus is to work out the methods for reconstructing f_{CT} and g_C functions. The accuracy, efficiency and costs of the mechanism are going to be evaluated in real production environments.

We plan to exploit the results of this work in order to optimise the usage of mathematical libraries in Grid environment. These efforts are broadly discussed in [10].

References

1. Grid RPC: A Remote Procedure Call API for Grid Computing. K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, H. Casanova, ICL Technical Report, ICL-UT-02-06, June, 2002.
2. Overview of Grid RPC: A Remote Procedure Call API for Grid Computing. K. Seymour, H .Nakada, S. Matsuoka, J. Dongarra, C. Lee and H. Casanova. Grid Computing – Grid 2002, LNCS 2536, pp. 274-278, November, 2002.
3. NetSolve: A Network Server for Solving Computational Science Problems. H. Casanova and J. Dongarra. The International Journal of Supercomputer Applications and High Performance Computing, Vol. 11, Number 3, pp. 212-223, 1997.
4. Utilizing the Metaserver Architecture in the Ninf Global Computing System. H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, S. Sekiguchi. High-Performance Computing and Networking '98, LNCS 1401, pp. 607-616
5. Innovation of the NetSolve Grid Computing System. D. Arnold, H. Casanova, J. Dongarra. To appear in Concurrency and Computation: Practice and Experience, 2002.
6. The Remote Computation System. P. Arbenz, W. Gander, M. Oettli. Parallel Computing (23): 1421-1428, 1997.
7. The Remote Computation System. P. Arbenz, W. Gander, M. Oettli. High Performance Computing and Networking, H. Liddell, A. Colbrook, B. Hertzberger and P. Sloot (eds.). Springer-Verlag, Berlin, 1996, pp. 820-825. (Lecture Notes in Computer Science, 1067).
8. Dynamically Forecasting Network Performance Using the Network Weather Service. R. Wolski, Journal of Cluster Computing, Vol. 1, pp. 119-132, January, 1998.
9. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. R. Wolski, N. Spring, and J. Hayes, Journal of Future Generation Computing Systems, Vol. 15, Numbers 5-6, pp. 757-768, October, 1999.
10. Optimisation of Usage of Mathematical Libraries in the Grid Environment. M. Brzezniak, N. Meyer. Proceedings of Second Cracow Grid Workshop, pp. 74-86, Cracow, Poland, 2003.