Estimating execution time of distributed applications

M.Drozdowski[1]

Abstract

In this work we consider the problem of estimating execution time of distributed applications. The main difficulty stems from the communication delays and shared nature of computing and communication media. A simple method taking into account communications and concurrency of computations is proposed to estimate the execution time of a distributed application. The proposed technique is evaluated in a sequence of experiments. It is demonstrated that this method is feasible. Other interesting consequences are also outlined.

Keywords: Parallel and distributed processing, scheduling, performance evaluation, communication delays, clusters of workstations.

1 Introduction

Measurement of the execution time of a computer application is fundamental to assess its efficiency. This observation applies also to parallel programs. Contemporary parallel applications are executed in shared and spatially distributed environments. Consequently, parallel programs are exposed to communication delays and coexistence of many applications competing for the resources of the computer system. Hence, a method of measuring execution time of a distributed application in a shared environment would be very desired.

The need for efficient allocation of the activities (programs) to the resources (processors, communication channels) gave rise to many scheduling models in parallel processing (cf. monographs: [1, 3, 5, 6]). These deterministic scheduling models assume initial knowledge of the parallel application parameters such as processing times or communication delays. In this work we consider a method of obtaining these parameters, the precision and soundness of the final model. The considerations of this work may lead to some general conclusions on usefulness of the well-known scheduling models.

[1]Institute of Computing Science, Poznań University of Technology, ul.Piotrowo 3A, 60-965 Poznań, Poland. This research was partially supported by KBN grant.

The study presented here has also practical origin and motivation. It has been observed that while reporting the efficiency of parallel applications run in shared environments, the influence of the competing programs is difficult to be dealt with. The indications of the computation efficiency tend to be unreliable. Even artificial superlinear speedup may be reported. Thus, a practical method of eliminating such influence is needed.

The above considerations boil down to a question: How to measure distributed application execution time? In the case of sequential applications executed on a single computer a natural answer, even in time-shared environment, is processor time consumed by the application. However, in parallel applications executed in a shared and distributed system, communication delays include influence of other communications. Likewise, computation time may include influence of other user applications.

Let us analyze several ad hoc solutions to this problem. Firstly, we can use *astronomical time* (wall time). This seems to be an ideal solution. Yet, only in a completely dedicated system. This means that not only other users are not allowed to use our computers, our network segment must be completely isolated from the outside world, but also the software suite running on our computers must be fully controllable. These requirements are realistic in small dedicated systems. However, it is far more difficult to fulfil them in geographically scattered and multiplatform systems. As a second solution, it is possible to measure processor times and system times consumed on all computers by our program. Still, this method does not take into account communication delays and precedence constraints between the operations executed on various processors. We conclude that a "good" method of measuring real execution time of parallel applications should

1. account for the order the program operations,
2. respect communication delays,
3. eliminate influence of other users and applications sharing the computing environment,
4. a method simple conceptually and in implementation is preferred.

An ideal satisfying the above requirements may be hard to achieve due to distributed nature of the application. We discuss it in the last section. However, we believe that a satisfactory estimation can be obtained by the method presented in this paper.

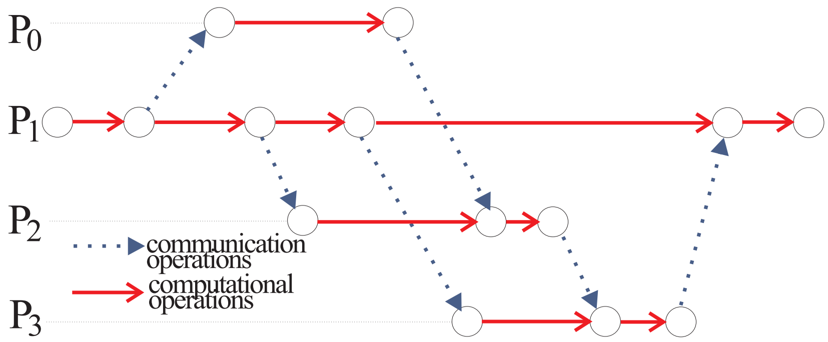

Figure 1: Control flow graph and estimation of the execution time

2 Proposition of a solution

Without loss of generality let us assume that communications involve two different computers connected by some kind of network. Thus, we exclude for the time being broadcast and internal communications. We will use term *processor time* to denote both the processor time consumed by the program executed in the user space and by the system on behalf of our application.

In the scheduling theory it is common to represent activities, programs or projects by a precedence constraints graph. Let $G(V, A)$ denote a directed acyclic graph with set V of events representing fork or join of the program, and set A of arcs representing communication or computation operations (cf. Fig.1). This method of representing precedences between the operations is called *activity on arc* in the deterministic scheduling theory [1]. Graph $G(V, A)$ represents control flow in the distributed application. Let every arc has a weight equal to the duration of the operation it represents. Hence, arc j representing computations has weight a_j equal to the processing (i.e. computation) time, while arc k corresponding to the communication has weight c_k equal to the communication time. The length of the longest path in graph G represents the execution time of the whole program. Example graph $G(V, A)$ is presented in Fig.1. Activities of processors $P_0, \ldots, P_m$ are shown along dotted horizontal lines. Solid lines represent computations, dashed lines represent communications between the processors. Determining the longest path in G is a well known and easily solvable problem [4]. Observe that G takes into account both precedence constraints and communication delays. Hence, calculating the running time of a distributed application as the longest path in G fulfills the first and the second requirement of the "good" method. However, a problem of satisfying the third requirement and collecting data to apply this method in practice still remains open.

In many cases the structure of G can be obtained fairly easily. In some applications, e.g. in linear algebra and numerical analysis, G is known a priori and fixed. In other applications it is possible to reconstruct G using logs of communications between the computers. The duration of a computational arc between two events (or nodes in graph G) can be measured as processor time used by our application. This value can be obtained, for example, by use of function `times` (UNIX) or `GetProcessTimes` (Windows) in C language. The values returned by the above functions are related to our own process, and eliminate influence of other users sharing the computing environment. Determining duration of the communication arcs is a harder problem. Communication time can be measured, for instance, as the difference between the communication initiation time on the sender, and the communication completion on the receiver. However, this method requires that distributed processors have perfectly synchronized clocks. Furthermore, influence of the external communications is not eliminated. We propose a different technique. Observe that there is a dependence of the communication delay t_{com} on the amount of the transferred data v: $t_{com} = f(v)$. As the data structures communicated over the network are explicitly declared by the programmer in the message passing environments it is not difficult to determine the amount v (e.g. in bytes) of the transferred information. $f(v)$ can be measured off-line as astronomical time. If $f(v)$ is measured in an unloaded or dedicated network, then the communication time estimated from $f(v)$ for known v, will retain character of a dedicated network even if the real communication took place in shared and loaded environment. In this way influence of the external communications can be eliminated. Thus, labeling communication arcs in graph G with the amount of data transferred is equivalent to labeling it with the communication time. The procedure is applied off-line after the application termination on the basis of the recorded logs.

3 Experimental evaluation

The method presented in the previous section has been tested in a series of experiments conducted on various applications and environments. Unfortunately, the experiments not always were performed in the same way.

Therefore, the results are preliminary. Due to space limitations we are not able to present all the results we obtained. We believe, however, that the main conclusions are valid despite that.

The goals of the evaluation were to verify stability of the reported running time in changing environment conditions, and to investigate coincidence of the results with the running time in dedicated environment. Satisfying the former condition is necessary for drawing conclusions on the behavior and the performance of distributed applications in shared and changing environment. The latter condition is imposed by the need for eliminating influence of the competing applications.

The first experiments (experiments 1) were performed in a cluster of four heterogeneous Sun workstations. The communication medium was a single segment 10Mb Ethernet. The second group of experiments (experiments 2) was performed in a cluster of five homogeneous PCs: Pentium 166MMX, 32 MB of RAM, Windows NT 4.0, connected by a single segment 10Mb Ethernet. The third series experiments (experiments 3) were done on six PCs: Pentium 200MMX, 64MB of RAM memory, with Linux operating system (Red Hat 6.0, kernel 2.2.5), interconnected by 100Mb single segment Ethernet. PVM message passing library was used in all experiments. The applications were: distributed search for a pattern in a text file (experiments 1), distributed multiplication of two matrices (experiments 2), distributed computation of Julia set image (experiments 3).

In the search for a pattern application [8] a big data file was divided into chunks. The chunks of text were sent to the processors to seek for a text pattern. A ready processor applied for data to the master processor. The order of activating the processors and the assignment of the data pieces to the processors was not known a priori. Therefore, the structure of the control flow graph G was known only after the execution of the program.

Matrix multiplication application [7] consisted in computing a product of two square matrices. The logical processor interconnection was a mesh topology. The multiplied matrices were divided into stripes and sent along the columns of the mesh. The results were sent along the rows of the mesh. In this application the control flow graph is known a priori. Nondeterminism was possible only in the order of final receiving the results by the master processor.

The third application [2] was a distributed computation of Julia set image. The image of the Julia set (a.k.a. Mandelbrot fractal) is showing the convergence of a sequence $z_0 = 0, z_{n+1} = z_n^2 + c$ of complex numbers. The convergence of z_n was mapped from a square [-2,2]$\times$[-2,2] of parameter c values to a 1000$\times$1000 bitmap. Slave processors analyzed convergence of z_n in a submesh determined by the coordinates of the opposite corners (lower-left and upper-right). Work was distributed to the processors in equal chunks representing some number of full horizontal lines in the bitmap. After initial distribution of the work the first processor requesting work also received new load first from the work distributor (master). Hence, the structure of the control flow graph was not known a priori as the order of sending work and receiving results from the slaves was not fixed.

To restore the control flow graph for all operations the consumed processor time, receiver/sender identifiers, amount of transferred data were logged. In experiments 1 communication time for the transferred amount of data was calculated using Langrangean interpolation. In the remaining experiments communication time was approximated by a linear function.

The first test of the method consisted in verifying the coincidence of the astronomical execution time in a dedicated system, with the execution time calculated by our procedure in the system under varying load. These times should be equal, or at least, the difference should be constant. Example results of this test in experiments 1 are shown in Fig.2. Horizontal axis in Fig.2 represents system load reported by the operating system, along the vertical axis relative difference between the astronomical and the estimated times is shown. As it can be seen the difference is less than 18% and it is distributed in 6% range. Thus, the difference is small and stable.

In the second test stability of the method was examined. The changes of the execution time of the same application estimated by our method were recorded under various computer loads. The reported value should be constant under all loads. In experiments 3 the changes of the load were caused artificially by running a simple program with one infinite loop containing one integer add operation. The results of this test are presented in Fig.2 for experiments 3. On the vertical axis time is reported, on the horizontal axis the load is shown in the units of the number of running loading programs. The upper line is the astronomical execution time. Obviously, this time increases with the load. The lower line is the execution time estimated by our method. It is constant as expected.

Interesting results were obtained in experiments 2 (cf. Fig.4). The difference between the estimated execution time and the astronomical time was stable in the central range of the multiplied matrices sizes. However, the difference was non-negligible. It was observed, that the astronomical time

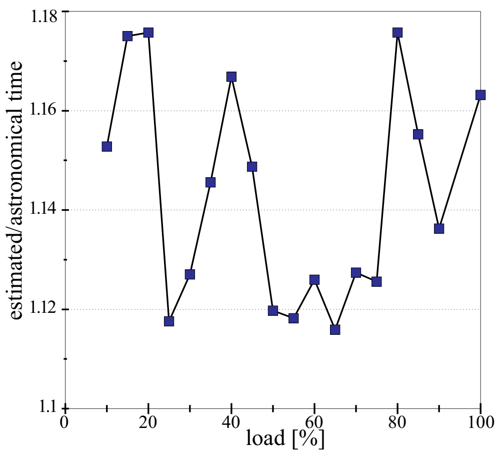

Figure 2: Differences in the estimated execution time under varying load. Astronomical time measured in unloaded system. Experiments I.

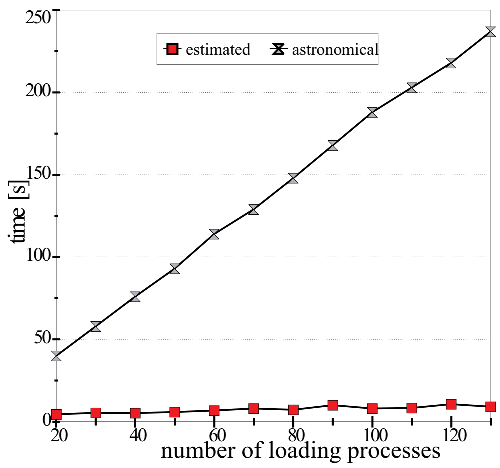

Figure 3: Stability of the estimated execution time under varying load. Astronomical time changes with increasing load. Experiments III.

comprises execution time of the operating system services and runtime environment (e.g. `pvmd`). These overheads are not included in the consumed processor time of our application. We concluded that astronomical time was longer than it should be. During the execution of our application, the operating system and other services consumed, e.g., 20% of the processor time. Thus, the astronomical time inevitably includes contribution of the operating system overheads. In order to have a fair comparison the astronomical time should be decreased by approx. 20%. Alternatively, the computational arcs in control flow graph G should be lengthened approx. $1/(1-0.2)$ times. The results of such a correction in the estimated time are shown in Fig.4.

4 Discussion and conclusions

In this section we discuss the obtained results, potential extensions, and limitations of the method presented above.

The results obtained in the experiments show that the method is rational. The calculated estimates are stable and close to the execution time in dedicated environment. This technique can be applied in performance profilers.

There are, however, potential limitations in drawing conclusions on the basis of the calculated estimates. In many cases the structure of control flow

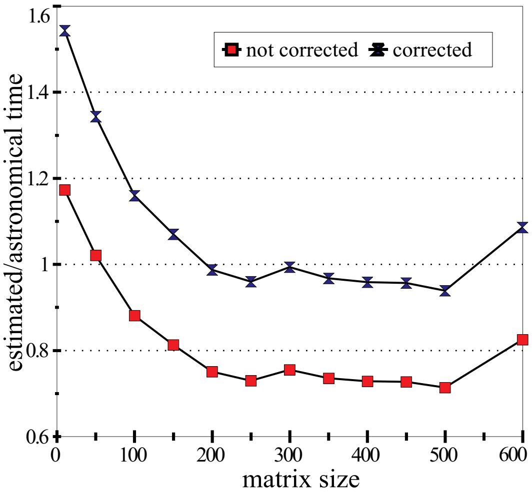

Figure 4: Influence of the standard system load

graph is nondeterministic, and can change, e.g., in different load conditions. Therefore, it may be unjustified to extend the results from the previous runs to the future ones.

There are potential sources of error in our method. The logging procedure may change the load of the computers. Consequently, the durations of the program operations and the structure of the control flow graph may be different than if the execution logs were not done. The results of our experiments show, however, that on average this influence is not big. Our method uses procedures of the operating system to measure consumed processor time. It is an open question how reliable these services are. Errors can be introduced also in the estimation of the communication delay. We observed that with the increasing speed of the communication networks and the processors the contribution of the linear dependence of communication time on the volume of transferred data diminishes. The nonlinear phenomena arising from communication initiation, buffering and reception become dominating. Note, that in all our tests these operations were implemented in software.

The results of experiments 2 (cf. Fig.4) lead us to yet another confusing observation that our method is able to eliminate influence of other application completely. This means that also operating system and runtime environment can be excluded from the estimation. Still, it would be difficult to use a computer system without such a software layer.

The technique we presented can be extended in various directions. It would not be difficult to include interprocess communications done on the same processor. Furthermore, broadcast messages can be logged and included in the control flow graph. In our considerations we assumed full interconnection (clique) between the processors, which is hardly ever the case. As a result the messages of our own application may overlap in time and compete with each other for the communication medium. This leads to the extension of the communication operations. Yet, such an extension can be calculated if the communication network topology and the medium access protocol are known. For example in Ethernet, when two (or more) messages share the communication channel, the bandwidth decreases proportionately to the number of competing messages. Practical verification of these extensions can be a subject of future research.

We finish presentation of a method estimating execution time of a distributed application in a shared environment with the conclusion that the results of the experiments demonstrate method usability. Still, many questions remain open.

References

[1] J.Błażewicz, K.Ecker, E.Pesch, G.Schmidt, J.Węglarz, *Scheduling Computer and Manufacturing Processes*, Springer, Berlin, 1996.

[2] B.Burba, B.Figas, Ł.Wasyłyk, Pomiar czasu wykonania aplikacji w środowisku rozproszonym, B.Sc. thesis, Institute of Computing Science, Poznań University of Technology, 1999.

[3] P. Chretienne and C. Picoleau. Scheduling with communication delays: A survey. In P. Chretienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, editors, *Scheduling theory and its applications*. J. Wiley, 1995.

[4] T.H.Cormen, C.E.Leiserson, R.L.Rivest, *Introduction to Algorithms*, Massachusetts Institute of Technology, 1990.

[5] E.G. Coffman Jr. (editor). *Computer and job-shop scheduling theory.* Wiley & Sons, New York, 1976.

[6] M.Drozdowski, *Selected problems of scheduling tasks in multiprocessor computer systems*, Series: Monographs, No.321, Poz-

nań University of Technology Press, Poznań, (1997), (see also http://www.cs.put.poznan.pl/~maciejd/h.ps).

[7] R.Janasiak, Pomiar rzeczywistego czasu wykonania aplikacji w środowisku rozproszonym, M.Sc. thesis, Institute of Computing Science, Poznań University of Technology, 1998.

[8] R.Maciejewski, Szacowanie czasu wykonania aplikacji w środowisku rozproszonym, M.Sc. thesis, Institute of Computing Science, Poznań University of Technology, 1997.