Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 12 Issue 15 Version 1.0 Year 2012
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Simulator for Resource Optimization of Job Scheduling in a Grid Framework

By P.K. Suri & Sumit Mittal
M.M. University, Mullana, Ambala, Haryana, India

Abstract - Traditionally, computer software's has been written for serial computation. This software is to be run on a single computer with a single Central Processing Unit (CPU). A problem is broken into a discrete serial of instructions that executed in the exact order, one after another. Only one instruction can be executed at any moment of time on a single CPU. Parallel computing, on the other hand, is the simultaneous use of multiple computer resources to solve a computational problem. The program is to be run using multiple CPU's. A problem is broken into discrete parts that can be solved concurrently and executed simultaneously on different CPU's. The purpose of this proposed work is to develop a simulator using Java for the implementation of Job scheduling and shows that Parallel Execution is efficient with respect to serial execution in terms of time, speed and resources.

Keywords : grid computing, grid framework, job scheduling, parallel computing, resource optimization.

GJCST-C Classification : D.4.1

SIMULATOR FOR RESOURCE OPTIMIZATION OF JOB SCHEDULING IN A GRID FRAMEWORK

Simulator for Resource Optimization of Job Scheduling in a Grid Framework

P.K. Suri [α] & Sumit Mittal [σ]

Abstract - Traditionally, computer software's has been written for serial computation. This software is to be run on a single computer with a single Central Processing Unit (CPU). A problem is broken into a discrete serial of instructions that executed in the exact order, one after another. Only one instruction can be executed at any moment of time on a single CPU. Parallel computing, on the other hand, is the simultaneous use of multiple computer resources to solve a computational problem. The program is to be run using multiple CPU's. A problem is broken into discrete parts that can be solved concurrently and executed simultaneously on different CPU's. The purpose of this proposed work is to develop a simulator using Java for the implementation of Job scheduling and shows that Parallel Execution is efficient with respect to serial execution in terms of time, speed and resources.

Keywords : grid computing, grid framework, job scheduling, parallel computing, resource optimization.

I. Introduction

Among the many disciplines of computer science, parallel processing is a discipline that deals with system structure and software processes related to the contingency performance of computer programs. It has been an area of active research interest and application for many fields, mainly the focus on powerful processing, but is now growing as the frequent processing model due to the semiconductor industry's move to multi-core processor chips.

Typically, software has been programmed for sequential computation, i.e., to be run on just one computer having just one main processing unit; where Instructions are implemented one after another, a problem is divided into distinct sequence of guidelines and only one instruction is executed at any instant. Although simple and economical serial computing is far much slower as compared to parallel computing. In essence, parallel computing is the simultaneous use of more than one processor or computer to solve a problem. Problems are run on multiple processors where each problem is broken into discrete parts which are further broken down into series of instructions to be executed simultaneously on different processors.

In the recent days, parallel computing has become popular based on multi-core processor chips. Most desktop computer and laptop computer systems are now delivered with dual-core micro-processors with quad-core processor chips which becomes easily available. Processor producers have started to increase overall computing performance by including additional CPU cores to provide the maximum parallelism in a program. The reason is that improving performance through parallel computing can be far more energy-efficient than improving micro-processor time wavelengths. In a world which is progressively mobile and energy aware, this has become essential.

II. Parallel Computing

Parallel computing is an outline of computation in which many data operations functions are carried out simultaneously [1].

Parallel computing takes four different forms: The first one is data parallelism / loop-level parallelism. In this computational form, a single thread controls all the data operations and in other situations, multiple threads control the execution, but they execute the same code. Second is Instruction level parallelism where instructions can be re-ordered and then, they are combined into groups and executed in parallel without affecting the result of the program [2]. The third form is task parallelism which is in contrast with data parallelism where the processor executes different threads in same or different sets of data. It focuses on allocating the processes on different parallel computing nodes. Task parallelism does not generally changes with the size of a problem [2]. The last form of parallel computation is bit level parallelism in which processors have to execute an operation on variables whose sizes are larger than the size of word.

Parallel computing has some advantages that make it attractive for certain types of problems that are suitable for use of multiprocessors, especially given limited computer memory, provides concurrency, saves money - Parallel computing resources can be built from cheap commodity components as shown in the figure 1, uses resources from a wide area network and saving time - Allocating more resources for a task shortens, it's time for completion with potential cost savings.

Conversely, parallel programming has also some disadvantages. By increasing the processors, memory in parallel computers, hence, produces a lot of data (I/O) and require parallel file system, need more space and more power, which leads to load imbalance.

Author α : HCTM Technical Campus Kaithal, Haryana, 136 027, India. E-mail : pksuritf25@yahoo.com
Author σ : M.M. Institute of Computer Technology & Business Management, M.M. University, Mullana, Ambala, Haryana, 133 207 India. E-mail : sumit_amb@yahoo.com

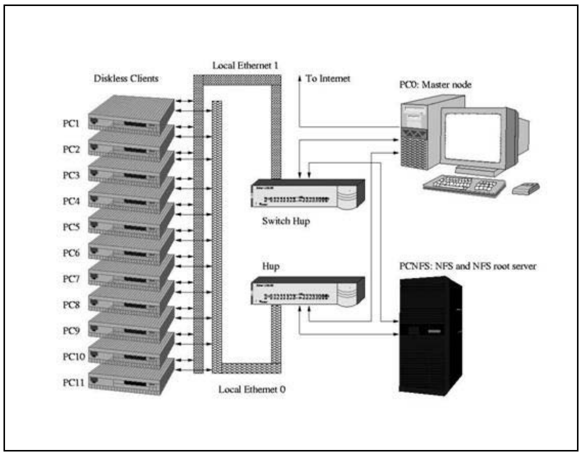


Figure 1 : Parallel Computing Cluster [3]

Parallel computer programs are more difficult to write than sequential ones, [7] because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically some of the greatest obstacles to getting good parallel program performance.

III. Association With Grid Framework

Grid computing combines computers from multiple administrative domains to solve a single task [4]. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. Grid computing tends to be more loosely coupled, heterogeneous and geographically dispersed [4]. On a single-processing machine, testing one model can take as long as five days. Using grid framework, the model can be distributed into different number of processing segments, each of which goes to its own processor; a task that normally takes five days can be completed in several hours.

IV. Comparison of Grids and Conventional Supercomputers

Distributed or grid computing in general is a special type of parallel computing that relies on complete computers resources (with onboard CPU's, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus [6]. The size of a grid may vary from small network of workstations within a corporation to large public collaborations across many companies and networks. The notion of a confined grid may also be known as intra-nodes cooperation whilst the notion of a larger, wider grid may thus, refer to inter-nodes cooperation [7].

The primary advantage of distributed computing is that each node can be purchased as commodity hardware, which, when combined, can produce a similar computing resource as multiprocessor supercomputer, but at a lower cost. The primary performance disadvantage is that the various processors and local storage areas do not have high-speed connections. This arrangement is thus well-suited to applications in which multiple parallel computations can take place independently, without the need to communicate intermediate results between processors [8].

V. Simulation of Resource Optimization

The purpose of this research work is to develop an algorithm for the resource optimization of job scheduling in a grid framework and shows that parallel execution is efficient in terms of time, speed and throughput. In addition to total time taken to execute all jobs, we will also calculate the CPU's usage efficiency using the following equation:

$$Efficiency = \frac{\sum_{i=1}^{number\ \ of\ CPUs} Worktime\ of\ CPU_i}{Tolal\ number\ of\ Jobs\ \times\ Time\ for\ 1\ Job} \times 100\%$$

The proposed application consists of 3 different classes:

1. Main Class: It contains a method that is being executed at program startup. The main flow occurs in this method.
2. Processor Class: This class contains methods to assign a job, check whether processor is free and ready for the next job and to calculate the amount of time it was busy.
3. Scheduler Class: It assigns jobs to the processors using the Round Robin scheduling. When a job needs to be assigned to the CPU, scheduler searches for the free processor and assigns the job to it. If at some moment, all CPUs are busy, it waits until one of the processors becomes free.

VI. Algorithm to Compute the Efficiency of Job Scheduling in Grid Framework

Step 1. Read one line of data from the input file:

i. Test serial number
ii. Number of jobs
iii. Number of processors.

Step 2. Create an array of instances of the Processor class. Create Job scheduler instance.

Step 3. Run Job Scheduler.

Step 4. When work is finished (all jobs have been executed); and calculate the results (time taken).

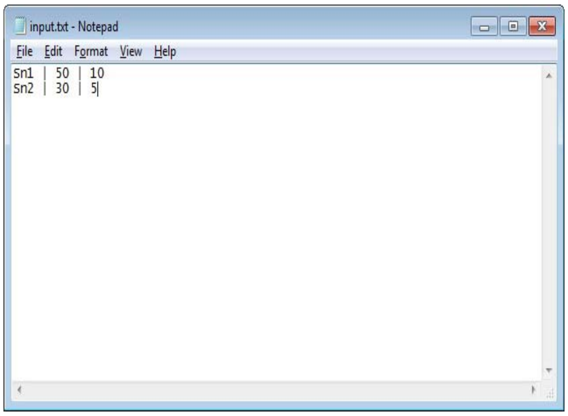
Step 5. Write results to the output file.

Step 6. Go to step no. 1) unless input file is fully processed.

VII. Simulation Results and Discussions

In order to compare serial and parallel implementations, a few series of test has been performed. For a fixed number of CPU's (2, 5, 10, 20, 30, 40, 50 in different series of tests), we ran 200 tests with different number of Jobs (5, 10, 15, etc up to 1000).

The input file is a simple text file that can be created in any text editor as shown in figure 1. Each line of this file should consist of 3 values separated by pipes ("|"). First value is a string that is a serial number of the test, second value is an integer that is number of jobs and third value is an integer that is number of processors.

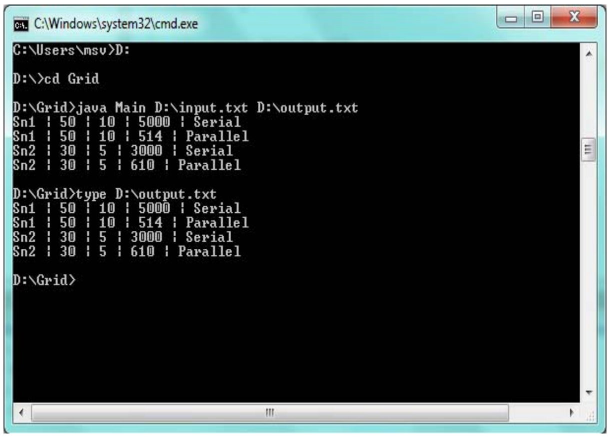


```
Sn1 | 50 | 10
Sn2 | 30 | 5
```

Figure 1 : Input File

Figure 2 shows the output file of the developed simulator, in this, each row consists of 5 columns separated by pipe char ("|"):

Sr. N. | Number of Input Jobs | Number of Processors | Time Taken | Execution Type (Serial / Parallel)



```
C:\Users\msv>D:

D:\>cd Grid

D:\Grid>java Main D:\input.txt D:\output.txt
Sn1 | 50 | 10 | 5000 | Serial
Sn1 | 50 | 10 | 514 | Parallel
Sn2 | 30 | 5 | 3000 | Serial
Sn2 | 30 | 5 | 610 | Parallel

D:\Grid>type D:\output.txt
Sn1 | 50 | 10 | 5000 | Serial
Sn1 | 50 | 10 | 514 | Parallel
Sn2 | 30 | 5 | 3000 | Serial
Sn2 | 30 | 5 | 610 | Parallel

D:\Grid>
```

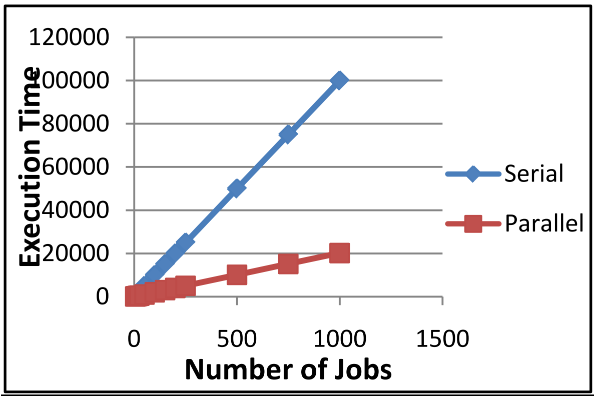
Figure 2 : Output File

Test Case 1: Table 1 shows the execution time (in microseconds) taken by the processors in serial and parallel computation and efficiency of the system for the 5 CPU's.

	Time (In Microseconds)		
Jobs	Serial	Parallel	Efficiency
5	500	105	95.24
10	1000	206	97.09
15	1500	307	97.72
20	2000	408	98.04
25	2500	509	98.23
50	5000	1014	98.62
100	10000	2024	98.81
150	15000	3034	98.88
200	20000	4044	98.91
250	25000	5054	98.93
500	50000	10104	98.97
750	75000	15154	98.98
1000	100000	20204	98.99

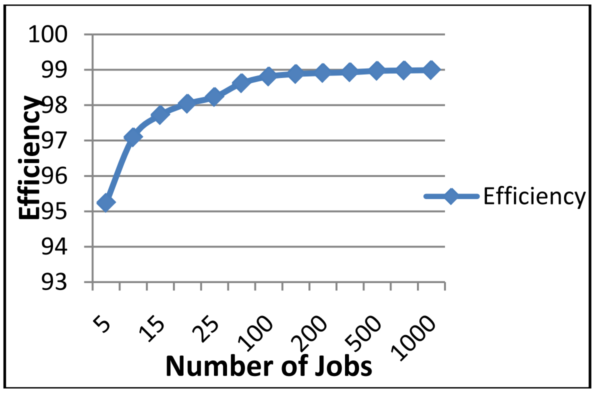
Table 1 : Execution Time and Efficiency (Number of CPU = 5)

The graph 1 depicts the relationship between the no. of jobs & the execution time estimation for the test case 1 (number of CPU's = 5) and graph 2 shows the efficiency vs. no. of jobs in terms of the time & resources.



Graph 1

Global Journal of Computer Science and Technology (C) Volume XII Issue XV Version I Year 2012



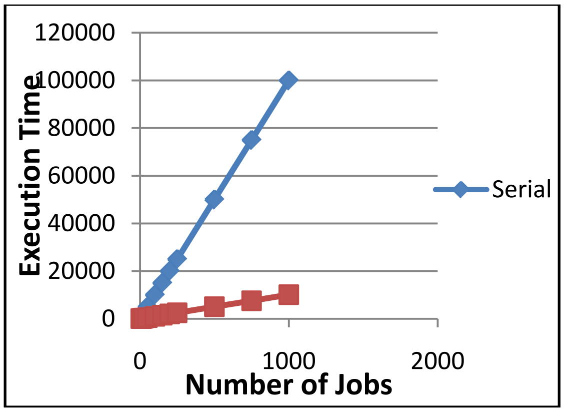
Graph No. 2

Test Case 2: Table 2 shows the execution time taken by the processors in serial and parallel computation and efficiency of the system for 10 CPU's.

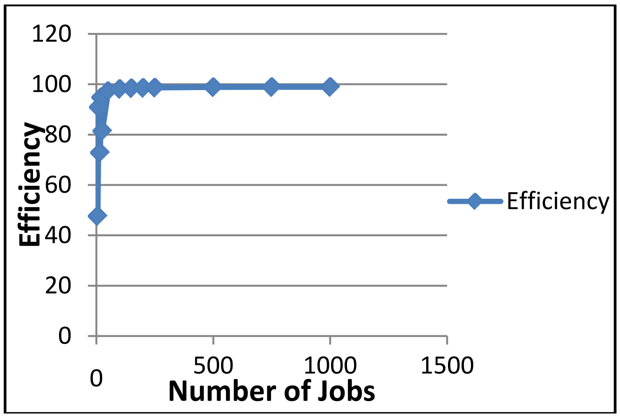
Jobs	Time (In Microseconds)		Efficiency
	Serial	Parallel	
5	500	105	47.62
10	1000	110	90.91
15	1500	206	72.82
20	2000	211	94.79
25	2500	307	81.43
50	5000	514	97.28
100	10000	1019	98.14
150	15000	1524	98.43
200	20000	2029	98.57
250	25000	2534	98.66
500	50000	5059	98.83
750	75000	7584	98.89
1000	100000	10109	98.92

Table 2 : Execution Time and Efficiency (Number of CPU = 10)

The graph 3 depicts the relationship between the number of jobs and the execution time estimation for the test case 2 (number of CPU's = 10) and graph 4 shows the efficiency vs. number of jobs in terms of the time and resources.



Graph No. 3



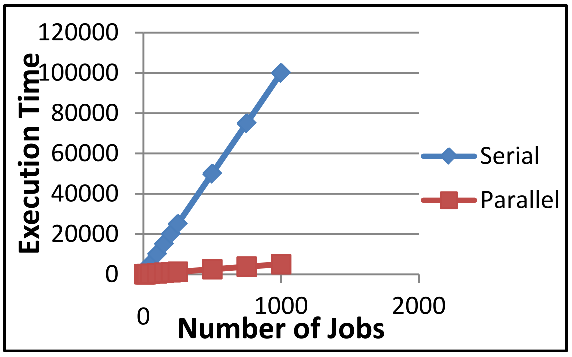
Graph No. 4

Test Case 3: Table 3 shows the execution time taken by the processors in serial and parallel computation and efficiency of the system for 20 CPU's.

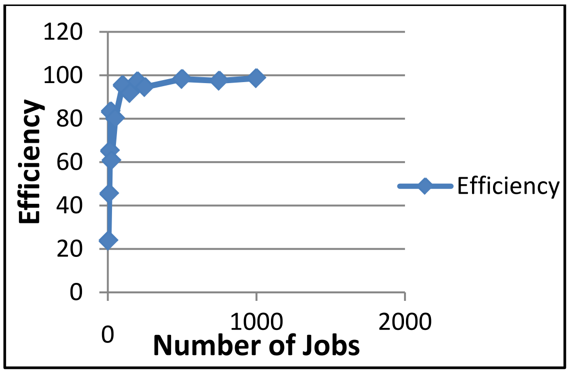
Jobs	Time (In Microseconds)		Efficiency
	Serial	Parallel	
5	500	105	23.81
10	1000	110	45.45
15	1500	115	65.22
20	2000	120	83.33
25	2500	206	60.68
50	5000	312	80.13
100	10000	524	95.42
150	15000	817	91.80
200	20000	1029	97.18
250	25000	1322	94.55
500	50000	2544	98.27
750	75000	3847	97.48
1000	100000	5069	98.64

Table 3 : Execution Time and Efficiency (Number of CPU = 20)

The graph 5 depicts the relationship between the number of jobs and the execution time estimation for the test case 3 (number of CPU's = 20) and graph 6 shows the efficiency vs. number of jobs in terms of the time and resources.



Graph No. 5

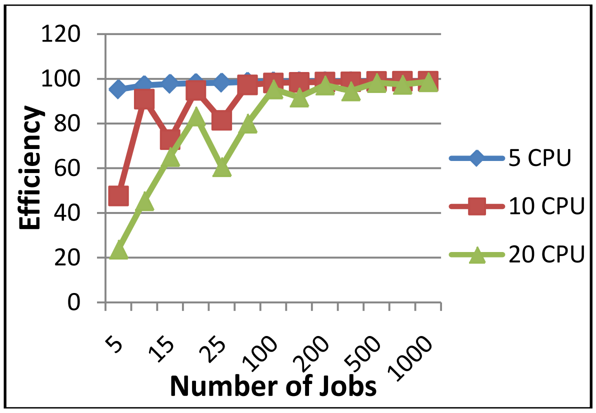


Graph No. 6

Table 4 shows the efficiency for different number of CPU's for a given set of jobs executed in the grid framework.

Jobs	Efficiency		
	5 CPU	10 CPU	20 CPU
5	95.24	47.62	23.81
10	97.09	90.91	45.45
15	97.72	72.82	65.22
20	98.04	94.79	83.33
25	98.23	81.43	60.68
50	98.62	97.28	80.13
100	98.81	98.14	95.42
150	98.88	98.43	91.8
200	98.91	98.57	97.18
250	98.93	98.66	94.55
500	98.97	98.83	98.27
750	98.98	98.89	97.48
1000	98.99	98.92	98.64

Table 4 : Comparison in terms of efficiency for different CPU's



Graph 7 : Comparison of efficiency for the different test cases

VIII. Discussion and Conclusion

After analyzing the generated results, it has been concluded that parallel execution is very efficient in terms of execution time and resources. Parallel execution on N processors is almost N times faster than serial execution. It's not exactly N times faster because of the job scheduler that needs some time to delegate tasks to the processors available. For large number of tasks, it approaches to 98% and 2% is that time when CPU's are unused waiting for a task from the job scheduler.

For a fixed number of jobs, when we are increasing the number of processors, the efficiency will be decreasing as depicts by graph no. 7, because when all processors are busy all the time; efficiency will be equal to 100%. When some of the processors were free (without a job assigned) some of the time, efficiency will be less than 100%.

As multi-core processors bring parallel computing to mainstream customers, the key challenge in computing today is to transition the software industry to parallel programming. Future capabilities such as photorealistic graphics, computational perception and machine learning rely heavily on highly parallel algorithms. Enabling these capabilities will advance a new generation of experiences that will expand the scope and efficiency of what users can accomplish in their digital lifestyles and work place. These experiences include more natural, immersive and increasingly multi-sensory interactions that offer multi-dimensional richness and context awareness.

References Références Referencias

1. Gottlieb. A, Almasi. G. S, "Highly parallel computing", Benjamin- Cummings Publishing Co., USA ©1989, ISBN:0-8053-0177-1.
2. David Culler, J.P, "Parallel computer architecture, Morgan Kaufmann Publishing Inc., San Fancisco, p. 15. ISBN 1-55860-343-3, 1997.

3. URL: http://proj1.sinica.edu.tw/∼ stat phys/ computer/buildPara.html.
4. http://www.e-sciencecity.org/EN/ gridcafe/ what -is-the-grid.html
5. Pervasive and Artificial Intelligence Group publications, Diuf.unifr.ch, 2010.
6. Berman, F., Anthony J. G. H. and Geoffrey C. F, “Grid Computing: Making the global infrastructure a reality”, ISBN 0-12-742503-9, 2003.
7. Asanovic, Krste et al., "The Landscape of Parallel Computing Research: A View from Berkeley", University of California, Berkeley, Technical Report No. UCB/EECS-2006-183, 2006.
8. URL: http://www.e-sciencecity.org/ EN/ gridcafe/ computational-problems.html.
9. S.V. Adve et al., "Parallel Computing Research at Illinois: The UPCRC Agenda", Parallel@Illinois, University of Illinois, Urbana, 2008.
10. Barney, Blaise. "Introduction to Parallel Computing". Lawrence Livermore National Laboratory, 2007.
11. Stallings, William (2004). Operating Systems Internals and Design Principles (fifth international edition). Prentice Hall. ISBN 0-13-147954-7.
12. Hennessy, John L.; Patterson, David A., Larus, James R. (1999). Computer organization and design : the hardware/software interface (2. ed., 3rd print. ed.). San Francisco: Kaufmann. ISBN 1-55860-428-6.
13. Hennessy, John L.; Patterson, David A. (2002). Computer architecture / a quantitative approach. (3rd ed.). San Francisco, Calif.: International Thomson. p. 43. ISBN 1-55860-724-2.
14. Bernstein, A. J., "Analysis of Programs for Parallel Processing", IEEE Transactions on Electronic Computers EC-15 (5): 757– 763, DOI: 10.1109/ PGEC. 1966.264565, 1966.
15. Roosta, Seyed H, “Parallel processing and parallel algorithms: theory and computation, New York, Springer, pp no. 114, ISBN 0-387-98716-9, 2000.