Performance Comparison of Priority Rule Scheduling Algorithms Using Different Inter Arrival Time Jobs in Grid Environment

Zafril Rizal M Azmi[1], Kamalrulnizam Abu Bakar[2], Abdul Hanan Abdullah[3], Mohd Shahir Shamsir[4], Wan Nurulsafawati Wan Manan[5],

[1,5]*Faculty of Comp. Systems and Software Engineering, Universiti Malaysia Pahang*
[2,3]*Faculty of Comp. Science and Information System, Universiti Teknologi Malaysia*
[4]*Faculty of Biosciences and Bioengineering, Universiti Teknologi Malaysia*

zafril@ump.edu.my, kamarul@fsksm.utm.my, hanan@fsksm.utm.my, shahir@fbb.utm.my, safawati@ump.edu.my

Abstract

Recent advancement in meta-heuristics grid scheduling studies have applied various techniques such as Particle Swarm Optimization (PSO), Genetic Algorithm (GA) and Ant Colony Optimization (ACO) to solve the grid scheduling problem. All of these technique requires an initial scheduler in order to initiate the scheduling process and the priority rule algorithms will typically be used. However, from the literature, none of these studies elaborate and justify their selection of a particular priority rule algorithms over another. Since the initial scheduler can significantly affect the entire scheduling process, it is important that the correct initial scheduler be selected. In this paper we quantitatively compared six initial scheduler algorithms to determine the best algorithm performance. We believe the performance comparison would enable users to utilize the best initial scheduler to fit their meta-heuristics grid scheduling studies.

Keywords: *Grid scheduling, priority rule algorithms, performance*

1. Introduction

The use of meta-heuristics techniques such as Particle Swarm Optimization (PSO), Genetic Algorithm (GA) and Ant Colony Optimization (ACO) to solve the grid scheduling problem requires an initial scheduler. These schedulers are usually the priority rule algorithms, which have been combined with meta-heuristics technique to schedule the jobs and resources effectively. Priority rules such as First Come First Serve (FCFS) have been previously applied to solve the queuing problems. Reported combinations includes the use of Longest Job First (LJF) and FCFS as an initial schedule for the Fuzzy Particle Swarm based scheduling [1], Shortest Job First (SJF) as initial scheduler for the Swift Scheduler algorithm [9] and the SJF and LJF for initial schedule for the Genetic Algorithms based scheduler [3]. However, there is no comparison study that investigates and ranks the performance of these priority rule algorithms. We feel that a comparison study would enable users to choose the priority rule algorithm best suited for their meta-heuristic grid scheduling techniques. Using performance metrics, we compared different performance parameters that would assist users to decide which priority rule algorithms are suitable for their grid system.

2. Priority Rule Algorithms

Priority rules also referred as Queue-based [7]. Instead of guaranteeing optimal solution, these techniques aim to find reasonable solutions in a relatively short time. Although it is a suboptimal algorithms, it is yet the most frequently used for solving scheduling problem in real world because of the easiness to implement and their low time complexity. In this study, we used six priority rules algorithms

2.1 First Come First Serve (FCFS)

First Come First Serve (FCFS) or also known as First In First Out (FIFO) is the simplest and the most fundamental of grid scheduling that involves client-server interaction. In grid scheduling, FCFS policy manages the jobs based on their arrival time, which means that the first job will be processed first without other biases or preferences. This concept has been used by several well known enterprise scheduler such as MAUI [6] and PBS [5].

2.2 Earliest Deadline First (EDF)

Earliest Deadline First (EDF) is a policy that schedule all the incoming jobs according to the specified due date or deadline. Incoming jobs will be processed or put in the queue based on the chronology indicate by the deadline. The job with the earliest deadline will be placed first in the processing queue.

2.3 Shortest Job First (SJF)

Shortest job First (SJF) also known as Shortest Job Next (SJN) or Shortest Process Next (SPN) is a scheduling technique that selects the job with the smallest execution time. The jobs are queued with the smallest execution time placed first and the job with the longest execution time placed last and given the lowest priority. In theory, the best strategy to minimize the response time is to schedule the shortest job on the fastest resource [1]. Since this policy gives preference to some groups of jobs over other group of jobs, this policy is unfair when compared to FCFS policy. In extreme cases, when jobs with shorter execution time continue to arrive, jobs with longer execution period may never get a chance to execute and would have to wait indefinitely. This is known as ‘starvation’ and would pose a serious problem and reflect the low degree of fairness for this policy [4]. In addition, SJF is believed to have the maximum makespan time compared to other algorithms because of this characteristic.

2.4 Longest Job First (LJF)

Longest Job First (LJF) have the contradiction behavior of SJF. While shortest job is believe will reduce the response time, processing longest job first on the fastest resource according to Abraham in [1] will minimize the makespan time. However, LJF will be suffering due to slightly increase in the response time.

2.5 Earliest Release Date (ERD)

Earliest Release Date (ERD) put the highest priority to the job that has the earliest release date in the queue. Release date is the start time of each and every job and it can be different or same. If there are two or more jobs that have the same release date, FCFS rule will be applied. Studies have also shown that if there is only a few numbers of jobs in the queue, the ERD

performance will be similar to FCFS but when the number of jobs increases, the results will defer. [8]

2.6 Minimum Time to Due Date (MTTD)

Minimum Time to Due Date (MTTD) is a scheduling algorithm which put the priority on the jobs according to the time that can be considered for the job to be executed with minimum tardiness [8]. To achieve this objective, MTTD define the time as follow:

(Deadline-Release Date) (1)

3. Experimentation On Grid Simulator

Based on the characteristics of priority rule algorithms explained earlier, a set of algorithms have been designed to implement the priority rule algorithms into grid environment. General architecture of the Priority-rule algorithms in grid environment is shown in Figure 1. Algorithm 1-6 representing Priority rule algorithms while Algorithm 7 determines resource selection and job allocation. This work uses Alea [7] the extended version of GridSim to simulate the scheduling process in a grid computing environment. The experiment were conducted by using simulation of 150 machines with different CPU number and speed and total number of 3000 jobs. The jobs datasets used consist of five folders. Each folder contains 20 different jobs file and each file has 3000 different jobs with specific inter-arrival time ranging 1-5 generated from negative exponential distribution. Folder DataSets-1 contained jobs with inter-arrival 1, DataSets-2 contained jobs with inter-arrival 2 and so on. There are total 150 heterogeneous machines were used in this experiment with a total numbers of experiments are 20 x 5 = 100 for each algorithm and the results obtained are the average performance of these data sets. For inter arrival time, the lesser job inter arrival time, the higher the contention in the system will be. In this experimentation, group of jobs with inter arrival time representing the heaviest load of jobs the grid have to schedule. To evaluate the performance of priority rule algorithms, we have used five different performance metrics: total number of delayed jobs, makespan time, flowtime, percentage of machine usage and total tardiness.

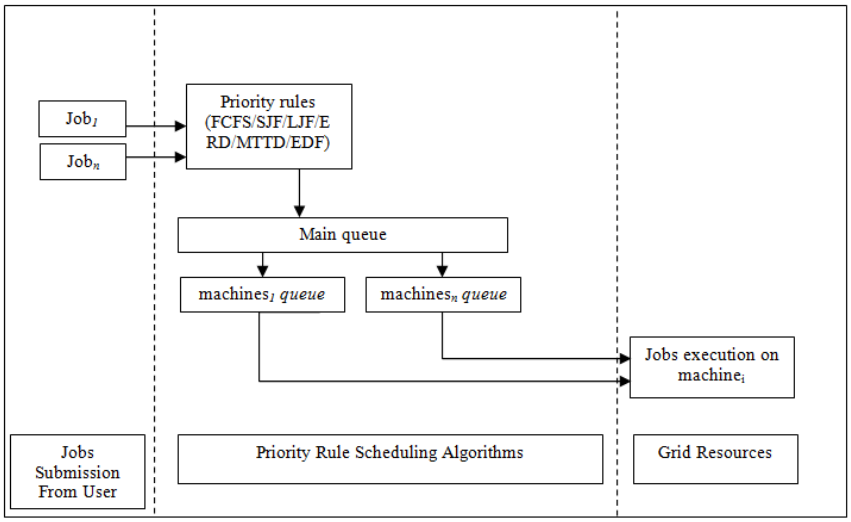

Figure 1. General Architecture of Priority Rules Scheduling Algorithms

Algorithm 1. FCFS

0 Add job_i last in the main queue

Algorithm 2. SJF

0 **for** i=0 to i<main queue.size
1 **if** job_{i+1} length < job_i length **then**
2 add job_{i+1} in front of job_i in the queue
3 **end if**
4 **if** main queue.size = 0 **then**
5 add job_i last in the main queue
6 **end if**

Algorithm 3. LJF

0 **for** i=0 to i<main queue.size
1 **if** job_{i+1} length > job_i length **then**
2 add job_{i+1} in front of job_i in the queue
3 **end if**
4 **if** main queue.size = 0 **then**
5 add job_i last in the main queue
6 **end if**

Algorithm 4. ERD

0 **for** i=0 to i<main queue.size
1 **if** job_{i+1} release date < job_i release date **then**
2 add job_{i+1} in front of job_i in the queue
3 **end if**
4 **if** main queue.size = 0 **then**
5 add job_i last in the main queue
6 **end if**

Algorithm 5. MTTD

0 **for** i=0 to i<main queue.size
1 **if** job_{i+1} due date - job_{i+1} release date < job_i due date - job_i release date **then**
2 add job_{i+1} in front of job_i in the queue
3 **end if**
4 **if** main queue.size = 0 **then**
5 add job_i last in the main queue
6 **end if**

Algorithm 6. EDF

0 **for** i=0 to i<main queue.size
1 **if** job_{i+1} due date < job_i due date **then**
2 add job_{i+1} in front of job_i in the queue
3 **end if**
4 **if** main queue.size = 0 **then**
5 add job_i last in the main queue
6 **end if**

Algorithm 7. Resource Selection

```
0  Perform Algorithm 1/2/3/4/5/6
0  if main queue.size > 0 then
1    get first job in the queue (job_i)
2    for j=0 to j<resource.size
3      get information of resource_j
4       if resource_j is free and the number of PE equal or more than job_i requested then
4          if resource_j speed > resource_j-1 speed
5            Accept resource_j
6          end if
7        end if
8      end for
9      if resource_j is accepted then
10       remove job_i from the main queue
11       send job_i to resource_j
12     end if
```

4. Results and Discussion

Previous work has demonstrated that when the job inter-arrival time increases, the number of late jobs decreases [2][7]. As expected, the results show that total number of delayed jobs decreases when the average inter arrival time increases (Figure 2). This is because, under low system contention, the job competition is also low and chances for machines to process more jobs will be high. Furthermore, from Table 1, we can clearly see that SJF have the least number of delayed jobs for DataSets-1. This shows that the SJF is the best algorithm to handle a great amount of jobs that came to the system at one time (in this case averages inter arrival time =1 and the objective is to reduce delayed jobs). However when the system become lighter (inter arrival time = 2-5), the performance of MTTD is the best.

Table 1. Total Number of Delayed Jobs by Priority Rule Algorithms

	FCFS	SJF	LJF	ERD	MTTD	EDF
DataSets-1	1967	1697	1953	1967	1698	1951
DataSets-2	1954	1366	1874	1955	1341	1839
DataSets-3	1937	1122	1764	1936	1061	1714
DataSets-4	1886	951	1595	1886	841	1227
DataSets-5	1802	850	1377	1803	716	720

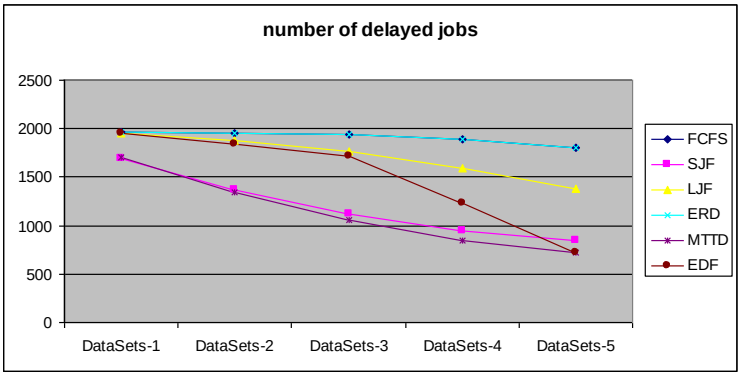

Figure 2. Graph Comparing the Number of Delayed Jobs by Priority Rule Algorithms

SJF also proven to be the best scheduler that can speed up the entire process. Figure 3 and Table 2 show that SJF have the least flowtime and at the same time outperform other schedulers for all inter arrival time.

Table 2. Flowtime by Priority Rule Algorithms

	FCFS	SJF	LJF	ERD	MTTD	EDF
DataSets-1	9024.624	6818.591	11456.98	9019.339	8109.23	8358.44
DataSets-2	7605.925	5570.798	10028.41	7624.006	6865.687	7247.473
DataSets-3	6265.873	4454.252	8615.786	6257.477	5680.557	5974.412
DataSets-4	4881.744	3456.51	6998.278	4890.977	4600.077	4697.654
DataSets-5	3567.71	2669.424	5254.541	3562.958	3516.616	3505.087

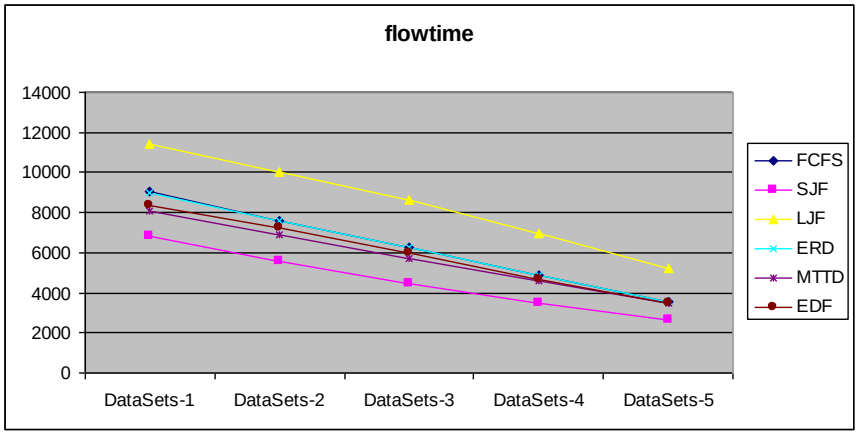

Figure 3. Graph Comparing the Flowtime by Priority Rule Algorithms

Figure 5 once again proven that SJF shows the best performance in terms of percentage of machine usage. It also means that SJF algorithm is able to better exploit the system computational resources compared to the other algorithms.

However, despite the good performances shows by the SJF algorithms for the delayed jobs, flowtime and machine usage, SJF suffer from the worst performance of the makespan. This can be seen from Table 3 and graph in Figure 4. On the other hand, although LJF shows bad performance for the three criteria mentioned before, it has the best makespan compared to others with 25% lesser then SJF.

Table 3. Makespan by Priority Rule Algorithms

	FCFS	SJF	LJF	ERD	MTTD	EDF
DataSets-1	24486	25106	19942	24262	24303	24316
DataSets-2	24620	25012	20132	24515	24705	24564
DataSets-3	24462	25455	20328	24879	24442	24663
DataSets-4	24981	25777	20424	24805	24470	24480
DataSets-5	24980	25583	21294	24787	24994	24957

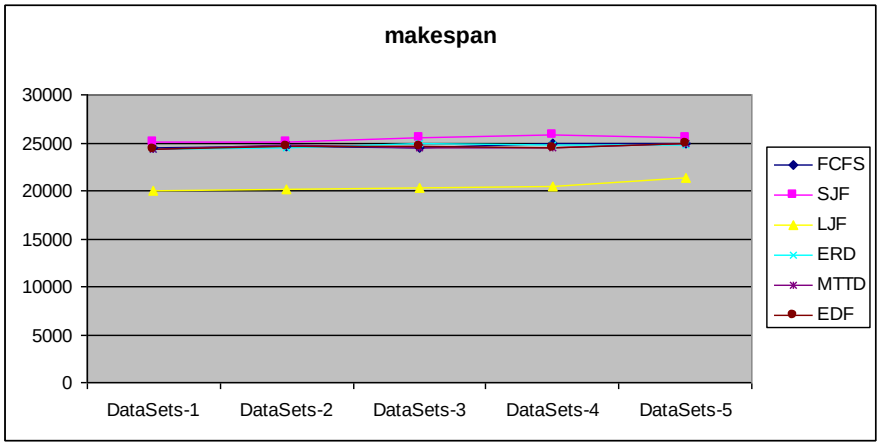

Figure 4. Graph Comparing the Makespan by Priority Rule Algorithms

Table 4. Machine Usage by Priority Rule Algorithms

	FCFS	SJF	LJF	ERD	MTTD	EDF
DataSets-1	88.896	90.0245	86.373	88.8405	88.8515	89.0425
DataSets-2	89.2195	90.183	86.502	89.142	89.1565	89.2485
DataSets-3	89.3615	90.039	86.634	89.2985	89.5345	89.501
DataSets-4	89.473	90.605	86.875	89.7135	89.7765	89.656
DataSets-5	89.773	90.7005	87.71	89.763	89.94	89.718

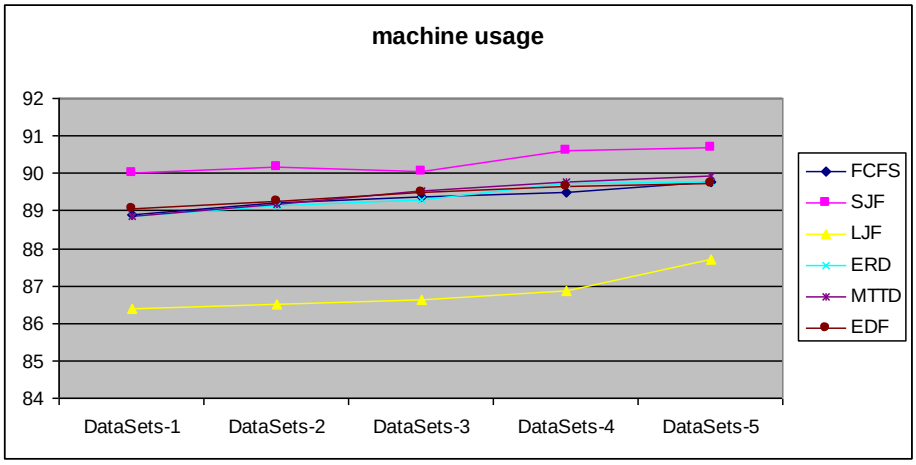

Figure 5. Graph Comparing the Machine Usage by Priority Rule Algorithms

In Table 5 and Figure 6, the result showed the total tardiness that is represented by the duration of jobs between its completion time and the corresponding deadline. MTTD scheduler outperforms other algorithm in all cases. This is because, by ordering a jobs based on equation 1, total lateness of jobs can be reduced

Table 5. Tardiness by Priority Rule Algorithms

	FCFS	SJF	LJF	ERD	MTTD	EDF
DataSets-1	15172016	10607690	20324174	15164512	7523838	8203722
DataSets-2	12173537	8224574	17281259	12213547	5168139	6117202
DataSets-3	9359418	5977296	14448580	9339971	3095640	3627323
DataSets-4	6488452	3987283	11095144	6513019	1325633	1282996
DataSets-5	3758901	2421436	7607480	3756994	541185	541446

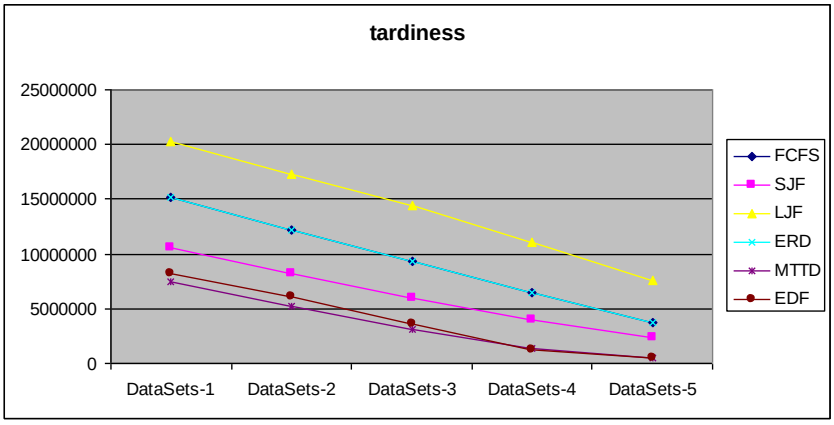

Figure 6. Graph Comparing the Tardiness by Priority Rule Algorithms

To summarize these results, we have used FCFS as a performance indicator and compared the percentage of other algorithms to show which algorithms is the best for the case of heavy load grid system (inter arrival time = 1). In Table 6, the results shows that SJF is the most promising algorithms to handle system with heavy contention. However it is depend on what objective the scheduler wants to achieve. For example, if the objective is to achieve best tardiness, MTTD is the best algorithms to be applied.

Table 6. Comparison of performance percentage based on DataSet-1

	FCFS	SJF (%)	LJF (%)	ERD (%)	MTTD (%)	EDF (%)
Delayed Jobs	0	13.73	0.71	0	13.68	0.81
Flowtime	0	24.44	-26.95	0.06	10.14	7.38
Makespan	0	-2.53	18.56	0.91	0.75	0.69
Machine Usage	0	1.27	-2.84	-0.06	-0.05	0.16
Tardiness	0	30.08	-33.96	0.05	50.41	45.93

4.5 Conclusions

In this paper, we have presented a performance comparison of priority rule scheduling algorithms that schedules jobs in grid computing system. We have present results based on five different inter arrival time that representing different level of contention in the grid system. This study is very important as a reference for researcher that wish to use one of this algorithms as a seed or initial schedule for their proposed scheduling algorithms especially the one involving heuristics.

References

[1] A. Abraham, R. Buyya, B. and Nath, Nature's heuristics for scheduling jobs on computational Grids, Proceedings of the 8th International Conference on Advanced Computing and Communications, Tata McGraw-Hill, India, 2000, pp. 45–52.

[2] G. Capannini, R. Baraglia, D. Puppin, L. Ricci, and M. Pasquali. A job scheduling framework for large computing farms. In ACM/IEEE International Conference for High Performance Computing, Networking, Storage and analysis (SC'07), 2007.

[3] J. Carretero, and F. Xhafa\, Use of Genetic Algorithms for Scheduling Jobs in Large Scale Grid Applications. Journal of Technological and Economic Development. A Research Journal of Vilnius Gediminas Technical University, ISSN 1392-8619, Vol. 12, 2006, No. 1, p. 11-17.

[4] J. M. Garrido. Performance modeling of operating systems using object-oriented simulation: a practical introduction, Kluwer Academic Publishers, Norwell, MA, 2000.

[5] R. Henderson. Job scheduling under the portable batch system. Job Scheduling Strategies for Parallel Processing, Springer-Verlag, Berlin 1995, pp. 337–360.

[6] D. Jackson. Snell Q. and Clement M. Core Algorithms of the Maui Scheduler, Lecture Notes in Computer Science, Vol. 2221, 2001, pp. 87-102.

[7] D. Klusacek, L. Matyska, and H. Rudova. Alea - Grid Scheduling Simulation Environment. In Parallel Processing and Applied Mathematics. Heidelberg, Germany : Springer-Verlag, Lecture Notes in Computer Science, 2007.

[8] A. Rasooli, M. Mirza-Aghatabar and S. Khorsandi. Introduction of Novel Rule Based Algorithms for Scheduling in Grid Computing Systems; Second Asia International Conference on Modeling & Simulation, 2008.

[9] K. Somasundaram, S. Radhakrishnan, Task Resource Allocation in Grid using Swift Scheduler, Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844, Vol. IV, 2009, No. 2, pp. 158-166

Authors

Zafril Rizal M Azmi obtained his BSc and MSc in Computer Science from Universiti Teknologi Malaysia. Currently he is a PhD student working on Grid Scheduling.

Kamalrulnizam Abu Bakar obtained his Ph.D degree from Aston University (Birmingham, UK) in 2004. Currently, he is an Associate Professor in Computer Science at Universiti Teknologi Malaysia (Malaysia) and member of the "Pervasive Computing" research group. He involves in several research projects and is the referee for many scientific journals and conferences. His specialization includes mobile and wireless computing, information security and grid computing.

Mohd Shahir Shamsir obtained his BSc in Biotechnology and Plant Sciences from University of Sheffield in 1996 and PhD in Biological Sciences (Computational Biology) from University of Exeter in 2005. Currently he is a Senior Lecturer at Faculty of Bioscience & Bioengineering, Universiti Teknologi Malaysia. His research focuses on molecular dynamics and biodiversity databases.

Abdul Hanan Abdullah received his BSc and MSc in Computer Science from the University of San Francisco and his PhD in the same field from the Aston University, UK in 1995. Currently he is a professor at Faculty of Computer Science and Information System, Universiti Teknologi Malaysia. His research interest includes network security, grid computing and active network.

Wan Nurulsafawati Wan Manan received her Bcs in computer science major in networking from University Technical Malaysia Malacca (UTeM) in 2006 and Master in Information Technology (Network) at Queensland University of Technology (QUT) on 2009. Currently she is a lecturer at Faculty of Computer System & Software Engineering, University Malaysia Pahang (UMP). Her research interest focuses on grid and cloud computing, network security and wireless network.