ISSN No. 0976-5697

Volume 2, No. 5, Sept-Oct 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# Scheduling Real-Time Multiprocessor Systems with Fault Tolerance Mechanism: Approaches Survey

Nima Jafari Navimipour*
Department of Computer, Tabriz Branch,
Islamic Azad University, Tabriz, Iran
n.jafari@srbiau.ac.ir

Seyed Hasan Es-hagi
Department of Computer, Tabriz Branch,
Islamic Azad University, Tabriz, Iran
Es-hagi@iaut.ac.ir

***Abstract:*** Real-Time system is a system that time of response to jobs play a very important role. Due to time limitations, fault tolerance mechanism in real-time multiprocessor systems is extremely important. In this kind of systems the fault should be discovered and be repaired as soon as possible to let the jobs are completed in determined deadline.

The specified system needs redundancy for be a fault tolerant system. One of the redundancy types is software redundancy. The special kind of software redundancy is "Primary Backup" that is used for scheduling real time multiprocessor systems. In this paper three new approaches for scheduling jobs in real time, fault tolerance systems is reviewed. Also we explain benefits and disadvantages of each approach are explained. Finally these approaches will be compared.

***Keywords:*** Fault Tolerance, Real Time Systems, Primary Backup, Multiprocessing

## I. INTRODUCTION

Real-time systems perform an important role in societal infrastructure, with application domains ranging from safety-critical systems (e.g., cars, aircraft, and robots) to user-interactive systems (e.g., consumer electronics, multimedia devices, streaming servers) [1]. Although, the real time system is a system in which all the jobs must completed in a specific time. In other word in these systems the accuracy of system depends on not only to the results of logical computations, but also to the time that these results have been produced in system. In hard real time systems, job should be completed exactly on time and no delay is acceptable, otherwise causes system to become incapable. In soft real time systems, response time is important but it is not as vital as the real time systems. Also jobs in real time systems are Periodic and Aperiodic. In periodic system jobs repeats periodically for example measuring temperature in special times. In aperiodic system jobs take place accidentally and this time is not determined.

In fault-tolerant real time systems, detection of fault and its recovery should be executed in timely manner so that in spite of fault occurrences the intended output of real-time computations always take place on time [2]. For fault tolerant technique detection, latency and recovery time are important performance metrics because they contribute to node downtime [2]. In real time systems, jobs should be finished in their deadline despite any fault, therefore the algorithms that can response in determined time with considering faults is needed. Fault tolerance is achieved by redundancy. The types of redundancy are hardware redundancy, software redundancy, information and time redundancy. Fault tolerant techniques implemented by means of scheduling are discussed in [3-5].

One of the most important challenges in real time systems is job scheduling. Job scheduling in real time systems is that, recognize when and on which processor a specific job should be executed. Job scheduling techniques can be used to achieve effective fault tolerance in real time systems [6, 7].

Real time scheduling methods are divided into Dynamic and Static methods. In static method, scheduling decisions are taken before the system start to execute but in dynamic methods, scheduling decisions are taken during the execution time. Static scheduling algorithms are executed on periodic jobs and do not has ability to execute on aperiodic jobs that their entrance time and deadlines are not determined. To scheduling these jobs dynamic methods are used. In dynamic scheduling when a new set of jobs enter a system, the scheduler will decide according to their specifications and flexibility.

Types of faults that may happen in system are permanent fault, transient fault and Intermittent faults. Permanent faults are caused by the total failure of a computing unit and are typically tolerated by using hardware redundancy, such as spare processors [1]. Transient faults can be caused by limitations in the accuracy of electromechanical devices, electromagnetic radiation received by interconnections [8]. Intermittent faults are repeated occurrences of transient faults [9]. Fault tolerance in multiprocessor systems is much more important because several parallel processors are working together and if any of them fails it can influence execution of other processors [8]. Although one of the advantages of distributed system is tolerating the fault in crashed component without disturbing other components [9].

Fault tolerant scheduling algorithms use Forward Recovery and Backward Recovery. One of the forward approaches that are used in fault tolerance scheduling is Primary backup (PB) method. In this method two copies of jobs will be scheduled in two different processors and a test is used to estimate the accuracy of results. The backup method only execute if main copy will damage.

The remainder of this article is organized as follows: Section II introduces the real time job scheduling problem, Section III motivates three different approaches are studied for job scheduling by PB method; Section IV compares these approaches; Section V presents conclusion and future works.

## II. REAL TIME SCHEDULING PROBLEM

Real-time scheduling provides a way of predicting the timing behavior of complex multi-tasking computer software [10]. It provides a number of ‘schedulability tests’ for proving whether a set of concurrent jobs will always meet their deadlines or not. Numerous improvements have been made to real time scheduling in recent years. Ways of transferring scheduling theory from academia to industrial practice have also been investigated in [11]. In [12] the model of real time scheduling is assumed that consists of the following components:

a. A set of computational jobs to be performed. Typically these are software ‘processes’.
b. A run-time scheduler that controls which job is executing at any given moment.
c. A set of shared resources used by the jobs. These may include shared software variables, both with and without mutual exclusion control, and shared hardware devices such as data buses.

In real time scheduling is supposed that jobs are three types, characterized by the arrival pattern of their individual invocations.

d. Periodic jobs consist of an infinite sequence of identical invocations which arrive at fixed intervals [13]. Thus their arrival pattern is time driven.
e. Aperiodic jobs consist of a sequence of invocations which arrive randomly, usually in response to some external triggering event [14]. Thus their arrival pattern is event driven.
f. Sporadic jobs are a special case of aperiodic ones in which there is a known worst-case arrival rate for the job, i.e., they have a fixed minimum interarrival time [14].

Many algorithms for scheduling real-time jobs exist in the literature. These algorithms in [15] are categorized as follows:

Static scheduling algorithms require the programmer to define the entire schedule prior to execution. At run time this pre-determined schedule is then used to guide a simple job dispatcher. Cyclic executives are one way to program static job scheduling [16].

Dynamic scheduling algorithms make decisions about which job to execute at run time, based on the priorities of the job invocations in the ready queue. They require a more complex run-time dispatcher or scheduler. Such algorithms can be further categorized into those based on fixed and changeable priorities [15].

Fixed-priority scheduling algorithms statically associate a priority with each job in advance. This can be done arbitrarily by the programmer, or according to some consistent policy.

Two well-known policies for fixed priority assignment are Deadline Monotonic Scheduling, in which jobs with shorter deadlines are allocated higher priorities [17], and Rate Monotonic Scheduling, in which jobs with shorter periods are allocated higher priorities [18].

Dynamic-priority scheduling algorithms determine the priorities of each job invocation at run time. Typically this requires a more complex runtime scheduler than fixed-priority scheduling. Two methods of dynamic priority assignment are Earliest Deadline First, in which the ready job invocation with the earliest upcoming deadline is given highest priority [19], and Least Laxity, in which the ready job invocation with the smallest difference between its upcoming deadline and (estimated) remaining computation time is given highest priority [20].

In the next section the common methods for scheduling the jobs in real time system to tolerate the fault are presented.

## III. FAULT TOLERANCE SCHEDULING METHODS

In this section three new approaches for real time system are reviewed. Each of these methods solves the scheduling problem of multiprocessor real time systems in specific method. All of these methods use PB, this means backups are taken from the main jobs and the jobs scheduled in a way that main job and its backup are not in the same processor also there is a mutual exclusion in the run time of job and its backup. Backup copy will execute just when the main copy can not execute properly and faces a problem. Also the EDF (Earliest Deadline First) algorithm is used in them. In this algorithm the process that has less time will execute first, means it has nearest deadline. This deadline for alternating events is equal to time of next event.

### *A. Backup Overloading Method (B/O):*

This method is proposed by Bindu Mirle and Albert M.K. cheng [21]. This method is based on PB (Primary/Backup). The jobs are assumed to be periodic and two instances of each job (a primary and a backup) are scheduled on a uni-processor system [21]. One of the restrictions of this approach is that the period of any job should be a multiple of the period of its preceding jobs. It also assumes that the execution time of the backup is shorter than that of the primary [21].

Also, in this method Primary Backup Fault Tolerance is used. This is the traditional fault-tolerant approach wherein both time as well as space exclusions are used. The main idea behind this algorithm is that (a) the backup of a job need not execute if its primary executes successfully, (b) the time exclusion in this algorithm ensures that no resource conflicts occur between the two versions of any job, which might improve the schedulability. Disadvantages in this system are that (a) there is no de-allocation of the backup copy, (b) the algorithm assumes that the jobs are periodic (the times of the jobs are predetermined), (c) compatible (the period of one process is an integral multiple of the period of the other process) and execution time of the backup is shorter than that of the primary process [21].

The authors found some weak point for PB and tried to solve them. The weak points are: 1- After allocating backup copy, deallocation does not happen. 2- It assumes that jobs are periodic; in other word their execution time is specific. 3- It considers that the compatibility time and execution time of backup copy is shorter than main copy. To solve these weak points they tried to develop PB to correct these problems.

In this method, the job is in one processor and its backup is in the other processor. The backup copy is allocated in a way that can overlap with other backup copies of jobs and this cause optimized use of existing processors. The purpose of overloading is to utilize the available time slot better because the chances that the backup copies have to run simultaneously, is remote [21]. If P1, P2 and P3 are existing processors, PRI1 and PRI3 are copy of jobs and Bk1, Bk3 are backup copies then the method of overlapping is shown in fig.1. The violet region in fig.1 is the overloaded region.

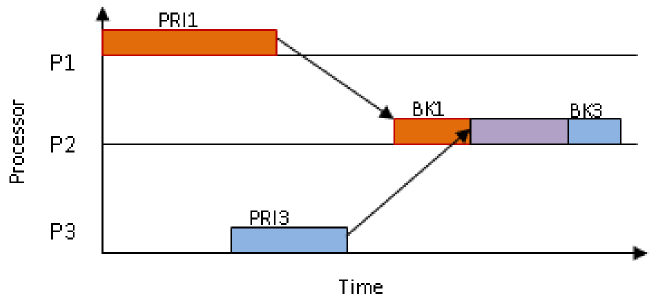

Figure.1. overlapping example

Two advantages of this method are optimum use of processors due to overlapping of backup execution time and deallocation of backup copies. It means when a job executed successfully, its backup will be deleted; so this cause to free the allocated space that can be used for scheduling other jobs. This method can tolerate only a single fault. The method cannot afford the failure of more than one processor at the same time. If more than one processor crashes at the same time, the method will fail to execute the jobs on time [21]. There are basically 2 kinds of faults considered here:

a. Transient Fault: This fault is only temporary and the system will start running correctly after some time. So, only that backup during which the processor had failed needs to be executed [21].
b. Permanent Fault: Here the processor undergoes a permanent flaw and all the processes scheduled on it crash. So, all backups and primaries on that processor have to be re-executed [21].

It has been deduced that the PB overloading algorithm has been quite successful in overcoming 86% of the transient faults but not the permanent faults.

## B. *Distributed Recovery Block Method (DRB):*

This method present a fault tolerance approach in heterogeneous distributed systems by software techniques based on distributed recovery block (DRB)[22]. Recovery blocks consist of several routines (their name is try block) that they are used to compute results like the results of main job and also there is acceptance test in these blocks that estimates the results from correctness and time limitations points of view. To simplify this method we use two blocks: Main block and Backup block. The fault process technique is the same acceptance test that is parallel among nodes but it functions serially inside each node. DRB is an approach for realizing both hardware fault tolerance and software fault tolerance in real-time distributed and/or parallel computer systems [22].

In each system it has considered two queues, one for main jobs and other one for backup jobs and also there is a central queue that at first all the jobs enter into it then one of the processors will be chosen to execute that job. In all queues use EDF technique in a way that, the job that has nearest deadline will execute sooner than the others.

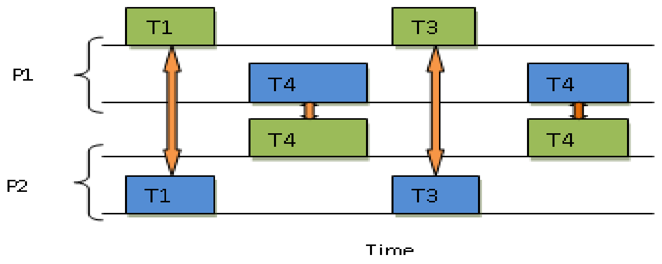

Figure.2: a possible initial schedule according to the period of the jobs

The idea of the DRB has been adapted from [23][24]. In this method each job is divided to sub-jobs. Each sub-job is executed in main block; second copy of its sub-job is updated in backup block. If a job fails, last updated sub-job in backup block can continue the job. Fig.2 shows scheduling jobs in this method.

In DRB two conditions should be considered. First, the execution of backup copy should not prevent main copy from execution. The second is if a main job fails and not finishes in determined time, the backup copy should executes but backup copy does not starts the job from beginning and continue it from the point that the sub-job created by main job. This method is used to tolerate timing faults and permanent faults.

The results show that this algorithm outperforms the traditional EDF uniprocessor scheduler, which has missed deadline in presence of timing and crash fault, and a randomized assignment of jobs [22]. Experimental results show that DRB method that is based on random EDF, can tolerate 10% to 20 % of permanent faults and several determined time faults [22].

## C. *Genetic Based Method:*

This model presents the fault tolerant scheduler based on genetic algorithm and backup copy of jobs. This algorithm designed for a soft real time multiprocessor systems that jobs are non exclusive and alternating, and also there is no relation between jobs. In genetic algorithm model, some heuristics used instead of classic methods that it finds optimized results mostly. This algorithm is designed for non-preemptive periodic jobs without any precedence relationship between jobs on a soft real-time multiprocessor system. Each job is assumed to have a primary and a backup copy that are allocated to different processors and the backup copy would be executed only if the primary copy fails due to the fault. The objective of our algorithm is to be fault tolerant and the jobs use the processors equally [8].

Genetic Algorithm is an efficient searching tool that was invented by John Holland [25].The genetic algorithm has great application for optimization of complicated problems mostly in where is not enough information about search space [27]. For solving any problem by genetic algorithm, seven components must be defined [26].

Representation (definition of individual): represents each chromosome in the real world. A chromosome is a set of parameters which define a proposed solution to the problem that the genetic algorithm is trying to solve.

Fitness function: These function shows the fitness of each chromosome. It is used to evaluate the chromosome and also controls the genetic operators [27].

Population: The population consists of chromosomes that each chromosome represents one result for the problem [27].

Parent selection mechanism: The role of parent selection is to distinguish among individuals based on their quality, in particular, to allow the better individuals to become parents of the next generation [27].

Reproduction: The reproduction operator is based on the Darwinian notion of "survival of the fittest"[27]. Individuals taking part in successive generations are obtained through a reproduction process or evolution operation. Individual strings are copied into a mating pool according to their respective fitness values. The higher the fitness values of the strings, the higher the probability of contributing one or more offspring in the next generation [27].

Crossover operators: Recombination operator selects two or more chromosomes and then produces two new children from them. It aims at mixing up genetic information coming from different chromosomes to make a new individual [27].

Mutation operators: Mutation operator selects one chromosome and then produces one new child from it by a slight change over the parent [27].

Survivor selection mechanism: The role of survivor selection is to distinguish among individuals based on their quality. This mechanism survives the individual among the passing from one generation to the next generation [27].

Termination Condition: The condition to ending the running of genetic algorithm [27].

The implemented genetic algorithm for this problem is: If n jobs exist, 2n jobs have to be scheduled in m processors. Each processor has a list that keeps the jobs execute in that processor and also have R value that shows remaining capacity. Firstly R is 1 for all the processors and their lists are empty. When a job, Ti, adapted to a processor, this job is added to its list and R value changes according to R - (Ci is maximum needed time for computations and Di is existing deadline for the job number i) [8]. Until $\sum \frac{Ci}{Di} <$, Jobs can be added to a processor. Fig.3 illustrates a sample chromosome [8].\

| 1 | 2 | 3 | … | n-1 | n | fitness |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |

Figure. 3: a sample chromosome

The first row in fig.3 indicates processor number, the second row shows the list of jobs that are scheduled and the last row indicates R (remaining time of processor).

One of the other purposes of this method is that trying to distribute load balances among CPUs. Therefore the fitness value for each chromosome is a variance of R. The fitness value is $\sqrt{\sum_{j=1}^{m}(\frac{Rj-R}{m-1}}$ that m is number of processors, Rj is remained efficiency of j processor, and R' is average R [8].

At last the summery of all these steps are as follows:

a. Initial and chromosome encoding
b. Generating initial population
c. Applying genetic operators: crossover and mutation
d. Adopting a set of operators to balance the load on processors.
e. Repeating steps 3 and 4 as much as needed duo to termination condition.

This method is designed for scheduling of non-preemptive independent jobs on a soft real-time multiprocessor system. Each job is periodic and is assumed to have a primary copy and backup copy that are assigned to different processors since the backup copy is executed only if the primary copy fails due to the fault [8].

## IV. METHODS COMPARISON

In this section the comparing of these models are presented. In table 1 these models are compared by eight aspects.

Table1: Comparing the B/O, DRB and GA methods with eight aspects

| **Method Aspects** | **B/O Method** | **DRB Method** | **GA Method** |
|---|---|---|---|
| Type of Fault Toleration | 86% of transient faults | 10% to 20% permanent faults and some transient faults | Permanent faults and transient faults |
| Number of Processor Failed Tolerated | 1 processor | About 10% of processors | Not mentioned |
| Types of Scheduling | Dynamic | Dynamic | Static |
| Types of Jobs | Not periodic | Periodic | Periodic |
| Job Dependency | Not exist | Not exist | Not exist |
| Types of Processor | Homogeneous | Heterogonous | Homogeneous |
| Central Scheduler | Yes | Yes | Yes |
| Other Advantages | Deallocation of backup copies. | Dividing a job to sub-jobs that prevent executed sub-jobs from re-executing. | The discovering method cause that find best or near the best optimized answer. |

As shown in Table1, B/O method can tolerate 86% of transient faults only one processor. The B/O schedules the job dynamically and the jobs are non periodic. In this method job dependency is not exist. The types of processor in the B/O method are homogeneous and same. In DRB method 10% to 20% permanent faults and some transient faults and about 10% of processors can tolerated. The DRB schedules the job dynamically and the jobs are periodic. The job dependency not exists in this method. The types of processor in the DRB method are heterogeneous. This method dividing a job to sub-jobs that prevent executed sub-jobs from re-executing. In GA method Permanent faults and transient faults can be tolerated. The GA schedules the job statically and the jobs are periodic. Same as the previous methods the job dependency not exists in this method. The types of processor in this method are homogeneous and have equal features. The discovering method cause that find best or near the best optimized answer in GA method.

## V. CONCLUSION AND FUTURE WORKS

In this paper three methods for job scheduling in real time fault tolerant systems are discussed. Also these methods are compared with eight most important aspects exactly. The B/O and DRB use PB for scheduling the jobs. Jobs are randomly adapted to processors then are scheduled and executed according to EDF. In third method genetic algorithm is used for scheduling the jobs. Third methods produces more optimized results because of its discovering procedure; but it has assumptions that restrict it, and If these assumption can be removed then more desirable answers will come out.

Although genetic algorithm finds optimized result but it has basic restricting assumptions. One of these assumptions is not considering job dependencies. In fact most of the jobs that are executing have depending to each other; so if job dependencies not considered, the executable jobs is limited and just involves some special types of jobs. If the job dependencies considered, then the number of jobs will increase and the scheduler system improved. Also if the system has intelligent that can delete backup copy of completed jobs then high efficiency obtained. Finally, it is clear that such a system that can resolve deficiencies of available system will be very useful.

## VI. REFERENCES

[1] Shinpei Kato, Yutaka Ishikawa, Ragunathan (Raj) Rajkumar, CPU scheduling and memory management for interactive real-time applications, Springer Science+Business Media, LLC 2011

[2] B Sahoo, A Ekka,2007. “A Novel Fault Tolerant Scheduling Technique In Real-TimeHeterogeneous Distributed Systems Using Distributed Recovery Block”, Proceedings of National Conference “VISION’07” on “High Performance Computing” 2nd April’07

[3] A. Girault, C. Lavarenne, M. Sighireanu, and Y. Sorel, “Generation of fault-tolerant static scheduling for real-time distributed embedded systems with multi-point links,” Proceedings 15th International, pp. 1265 – 1272, April 2001.

[4] G. Manimaran and C. Murthy, “A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis,” IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 11, pp. 1137 – 1152, November 1998.

[5] S. Ghosh, R. Melhem, and D. Mosse, “Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems,” IEEE Transactions on Parallel and Distributed Systems, pp. 272–284.

[6] R. Mall, Real-Time Systems, 1st ed. Pearson Education, 2007.

[7] K.H. Kim, “Slow advances in fault-tolerant real-time distributed computing," in Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, October 2004, pp. 106 - 108.

[8] G Zarinzad, A M Rahmani, N Dayhim, 2008. “A Novel Intelligent Algorithm for Fault-Toleran Task Scheduling in Real-Time Multiprocessor Systems”, Third 2008 International Conference on Convergence and Hybrid Information Technology

[9] Burns, A. and Wellings, A. J. 1995b. Engineering a hard real-time system: From theory to practice. Software–Practice & Experience 25, 7 (July), 705–726.

[10] Buttazzo, G. C. 1997. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Kluwer.

[11] Ghosh, S., Melhem, R., Mosse, D., Sarma, J. S.: Fault- Tolerant Rate Monotonic Scheduling. Journal of Real-Time System 15(2) (1998) 149-181

[12] Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. 1993. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal 8, 5 (Sept.), 284–292.

[13] Buttazzo, G. C. 1997. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Kluwer.

[14] Sprunt, B., Sha, L., and Lehoczky, J. 1989. Aperiodic task scheduling for hard real-time systems. Journal of Real-Time Systems 1, 1 (June), 27–60.

[15] Buttazzo, G. C. 1997. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Kluwer.

[16] Burns, A. and Wellings, A. J. 1990. Real-Time Systems and Their Programming Languages. Addison-Wesley.

[17] Tindell, K. 2000. Deadline monotonic analysis. Embedded Systems Programming 13, 6 (June), 20–38.

[18] Briand, L. P. and Roy, D. M. 1999. Meeting Deadlines in Hard Real-Time Systems: The Rate Monotonic Approach. IEEE Computer Society Press.

[19] Liu, C. L. and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM 20, 1, 46–61.

[20] Jones, M. B., Barrera III, J. S., Forin, A., Leach, P. J., Rosu, D., and Rosu, M.-C. 1996. An overview of the Rialto real-time architecture. In Proc. Seventh ACM SIGOPS European Workshop (Sept. 1996).

[21] B Mirle, AMK Cheng, 2006." Simulation of Fault-tolerant Scheduling on Real-time Multiprocessor Systems Using Primary Backup Overloading", Real-Time Systems Laboratory, Department of Computer Science, University of Houston, 2006

[22] R. Mall, "Real-Time Systems", 1st ed. Pearson Education, 2007.

[23] K. Kim, "Designing fault tolerance capabilities into real-time distributed computer systems," in IEEE Proceedings., Workshop on the Future Trends of Distributed Computing Systems in the 1990s, September 1988, pp. 318 - 328.

[24] Proceedings. 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Feb. 2004,pp. 434 - 439.

[25] Holland, J. H. (1975) "Adaption in Natural and Artificial Systems". The University of Michigan Press, Ann Arbor, MI, 1975

[26] A.E.Eiben, J.E.Smith, "Introduction to Evolutionary Computing", Springer, 2004

[27] N Jafari navimipour, A M Rahmani, The New Genetic Based Method with Optimum Number of Super Node in Heterogeneous Wireless Sensor Network for Fault Tolerant System, International Journal of Computer and Electrical Engineering, Vol. 2, No. 1, February, 2010, 1793-8163

## AUTHOR'S PROFILE

**Nima Jafari Navimipour** received his B.S. in computer engineering, software engineering, from Islamic Azad University, Tabriz Branch, Tabriz, Iran, in 2007, the M.S. in computer engineering, computer architecture, from Islamic Azad University, Tabriz Branch, Tabriz, Iran, in 2009. From 2011, he is a Ph.D student in Science and Research Branch, Islamic Azad University, Tehran, Iran. He has published more than 20 papers in various journals and conference proceedings. His research interests include Traffic Control, Traffic Modeling, Computational Intelligence, Evolutionary Computing, Computational Grid, and Wireless Networks.

**Seyed Hasan Es-hagi** received his B.S. in computer engineering, hardware engineering, from Shomal University, Amol, Iran, in 2007, the M.S. in computer engineering, computer architecture, from Islamic Azad University, Tabriz Branch, Tabriz, Iran, in 2009. From 2008, he worked as a researcher with the Young Researchers Club, Tabriz Branch, Islamic Azad University, Tabriz Branch. He is the author/co-author of more than 12 publications in technical journals and conferences. His area of research is in Digital Communications, Error Correction Codes, and Wireless Networks.