International Journal of Computer Applications (0975 – 8887)
Volume 51– No.3, August 2012

A Genetic Algorithm based Approach for Change Management in Enterprise IT Systems: Optimal Change Scheduling

Habib-ur Rehman
National University of Computer & Emerging Sciences (NUCES)
Islamabad
Pakistan

Irshad Khan
National University of Computer & Emerging Sciences (NUCES)
Islamabad
Pakistan

ABSTRACT
In this paper, we provide a Genetic Algorithm (GA) based approach for change management in Enterprise IT systems which minimizes change delay and improves change capacity. Change management evolves the implementation of change queues on the set of applications which are running on one or more servers. To implement these changes there are some constraints involved: atomic nature of changes; when a change is implemented then it cannot be interrupted; an application has a specific downtime, which causes timing constraint; applications that share same resources such as servers have overlapping downtime which causes conflicts among changes. In such complex system, scheduling of changes becomes a difficult optimization task. GA has the ability to optimize such constraint scheduling problems. Our GA base scheduling improves throughput and minimizes change delay of existing Capacity optimal Fluid Scheduling algorithm.

General Terms
Genetic Algorithms, Optimal Change Scheduling.

Keywords
Genetic Algorithms, Optimal Change Scheduling, Enterprise IT Systems, Fluid Scheduling.

1. INTRODUCTION
Scheduling is an optimization problem. In scheduling we want to schedule and produce an effective queue to schedule. For example, in operating systems we are interested in improving throughput of the operating system. From a simple environment to a complex environment scheduling is a challenging task for scheduler. A complex system involves constraints. e.g. Time table scheduling, change scheduling, and process scheduling.

In the context of enterprise IT System, the scheduling task involves the scheduling of changes to the software. Software change means software updates that must be implemented to those servers which are running applications to which the changes are associated with. Conflicts may exist among changes e.g. two changes of an application cannot be implemented at the same time. There are some properties of changes: a) A change is atomic in nature, therefore it must be implemented on server without interrupting it, until it is not completed, b) It has an estimated time for completion, therefore it must be ensured that it should be completed before the expiration of application permissible time. Those servers and applications will be unavailable for clients which are implementing changes.

The two basic constraints are scheduling constraint and timing constraint. If the two changes associated with some applications cannot be scheduled at the same time, then it is called scheduling conflict. This is due to the overlapping of resources such as servers on which these changes are going to be implemented. Timing constraint involves the application downtime window; downtime means that the application cannot be available to client during that time. As we know that in enterprise IT systems the applications are required to be available to client on a 24/7 basis. It is important to note that a change must be implemented at that time when permissible downtime of that application is available [1].

Every change has an executor which executes changes on specific application. An executor is a service specialist with appropriate skill sets. The entire problem which we are going to deal with in this paper consists of applications, servers, change queues, and executors.

GA is an optimization technique based on the principal of Darwin’s theorem, “survival of the fittest” [3], [7], [8]. GA has the ability to optimally solve NP-Hard Problems. GA is very good to solve such NP-Hard scheduling problems [4], [7], [8]. It is an evolutionary algorithm that use the principal of natural selection to evolve a set of solutions called chromosomes towards an optimum solution.

The model used in our work is based on Genetic Algorithm to solve the problem of change management. The problem is also modeled and solved in [1]. The novelty of our work is to design a GA to find out the optimal scheduler chromosome to minimize the change delay and increase the change capacity of the Enterprise IT system. Change capacity in general is the ability of a system to achieve the maximal set of change requests that can be scheduled [2].

The change scheduling policy is somehow analogous to time table scheduling of a college. Time table scheduling is a complex NP-Hard problem. Time table problem is very efficiently solved using GA [9], [10] and it is basically the prime motivation for our work.

2. PROPOSED MODEL
Our problem scenario has servers Si and applications Ai. Figure 1 shows servers, s1, s2, s3, s4, where each server is

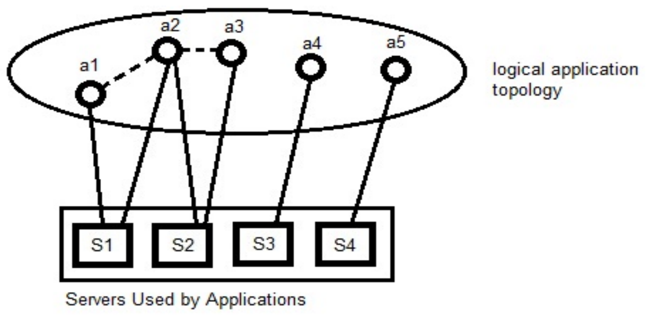


Figure 1: Set of servers {s1}, {s1, s2}, {s2}, {s3}, {s4} are used by applications a1, a2, a3, a4 and a5 respectively; application conflicts are represented by dotted line.

running one or more applications from a1, a2, a3, a4, a5. The conflict constraint is given by dotted line, which means that a1 and a2 are said to be in conflict with each other because it shares the same server S1, similarly a2 and a3 are said to be in conflict with each other because it shares server S2.

Our model consists of a set of applications Ai and set of servers Vi. The set of servers used by an application Ai is represented by Vi where $V_i \subseteq V$. We have set of changes C. Each change is associated with a unique application, which is represented by Ci. Two applications are said to be in conflict if they use the same servers. i.e. $Vi \cap Vj \neq \varphi$. Therefore their corresponding changes Ci and Cj will also be in conflict. An application is also in conflict with itself if it has two changes to be implemented. Therefore, its changes cannot be scheduled at the same time. To keep in mind these notion and timing constraints of applications, we use GA to solve this problem. Note that change is atomic in nature; therefore, a change should only be selected if it can be completed in the permissible time of that application.

Approximate Capacity-optimal fluid scheduling algorithm is used to solve the problem of change management [1]. The authors used his model and algorithm in his earlier work to consider the scheduling question for multichannel wireless networks [5]. In the case of change scheduling, they consider both fluid and approximate fluid regime. In the fluid regime the change is considered as non atomic in nature. Once a change is scheduled it can be preempted and scheduled again. But it is not a valid scenario, because our change is atomic in nature.

Our model is based on GA. GA performs sophisticated scheduling in such a constrained environment like time table scheduling solved by [6]. Authors model the time table problem in the domain of GA. Literature survey shows that the time table problem also consists of timing and conflict constraints. This motivates our idea to use GA for change scheduling optimization.

The GA can be performed as a local strategy or as a global strategy. In local strategy the changes of a single application represents a chromosome. While in global strategy the changes of all applications that can complete within the permissible time of application represents a chromosome.

3. PROPOSED TECHNIQUE

The proposed system has set of V servers which running the set of applications A and have X executors. Each application has downtime H hour in a day. When a change is arrived, it is stored in the corresponding queue of that application to which change is associated with, as shown in Table 1.

Table 1: Application Change Queues

Change	c1	c2	c3	c4	d1	d2	d3
Application	a1	a1	a1	a1	a2	a2	a2
Executor	x1	x1	x2	x2	x1	x2	x1

Here c1, c2, c3 and c4 represent the number of available changes for application a1, and d1, d2 and d3 represent the number of available changes for application a2 and so on. Each change is associated with an application and executor. The above Queue represents an individual because we have used global GA strategy. The drawback of local strategy is that it only considers current application; hence we have overlapping time of conflicting applications due to which fragmentation loss can occur. Fragmentation loss means that if the time window is 10 and there are four changes having execution time 3, 4, 4 and 1 then only (4+4+1=9) hours can be utilized optimally and 1 hour will be wasted. Changes are queued in the queues U. It will ensure that the changes in the queue U' ϵ U are not in conflict with each other and each change should be completed within the application permissible time. Our GA based approach works as following:

The chromosome representation for GA is the collection of all those changes which are currently available to schedule such as c1, c2, c3, c6, c9, d1, d2, d3, e1, e2. Here c1, c2, c4, c6, c9 represent available changes of application a1; d1, d2, d3 represent available changes of application a2; e1, e2 represent available changes of application a3, and so on.

The fitness of the chromosome is calculated to find a chromosome which schedules the changes under the constraints and increases the throughput of the systems:

$$fit(U') = \sum_{k=1}^{c} U'(C_k) \qquad (1)$$

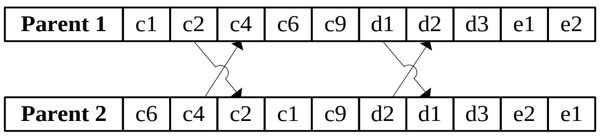
where c is the number of changes in the queue U'.

To create new population selection strategy is performed, which selects two parents to generate a new offspring to explore solution space. Selection is performed at random.

Crossover: to generate new offspring midpoint crossover operator is performed to parents as shown below in Figure 2.

After crossover two off springs are created offspring1 and offspring2. Offspring1 contains c1, c2, c4 from parent1, then c1, c9, d2, d1 from parent2 and then the remaining d3, e1, e2 again from parent1. The same phenomenon is followed for offspring2 as well.

Mutation: to ensure the diversity mutation operator is used which randomly selects two genes (changes) of the created offspring and swap it, as presented in Figure 3. After mutation



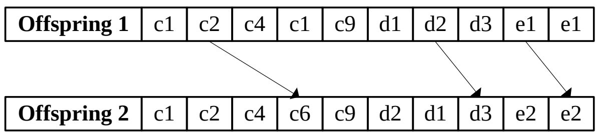
Parent 1	c1	c2	c4	c6	c9	d1	d2	d3	e1	e2
Parent 2	c6	c4	c2	c1	c9	d2	d1	d3	e2	e1

a) Crossover between parent1 and parent2

Offspring 1	c1	c2	c4	c1	c9	d2	d1	d3	e1	e2
Offspring 2	c6	c4	c2	c6	c9	d1	d2	d3	e2	e1

b) Two offspring after crossover

Figure 2: Crossover operator



Offspring 1	c1	c2	c4	c1	c9	d1	d2	d3	e1	e1
Offspring 2	c1	c2	c4	c6	c9	d2	d1	d3	e2	e2

a) Before Mutation

Offspring 1	c1	c6	c4	c1	c9	d1	d3	d3	e1	e2
Offspring 2	c1	c2	c4	c2	c9	d2	d1	d2	e2	e1

b) After Mutation

Figure 3: Mutation Operator

the new mutated chromosomes are developed as in Fig. 3 (b).

For the construction of a complete chromosome we need a repair method to ensure that all the available changes present in the new offspring and there is no repeated gene in the chromosome. After genetic operators chromosomes are given to scheduler that will schedule it. The chromosomes may have conflicting changes. Scheduler schedules changes and gives a queue of those changes that are scheduled. The number of changes scheduled by the scheduler for that chromosome will be the fitness of that chromosome.

The objective function contains the maximization of total changes scheduled by scheduler and minimizing the average change delay.

$$\text{Max U} = \sum_{i}^{p} \sum_{k=1}^{i_k} U_k \qquad (2)$$

$$\text{Min W} = \sum_{i}^{p} \frac{1}{K} \sum_{k=1}^{i_k} W_k \qquad (3)$$

where U is the queue scheduled by scheduler for chromosome i of population p having changes k. And W is the average waiting time of chromosome i of population p.

It is not necessary that a chromosome will have all the changes that are available at that time. A chromosome will have those changes which meet the permissible time of the application, and it optimizes the capacity of the system.

The algorithm runs capacity optimal fluid scheduling CFS at the side not considering the fluid regime. In CFS the system consists of change queuing policy and change scheduling policy. When a new change arrives, then change queuing policy queues the change in the appropriate queue. While change scheduling policy activates and reschedules when permissible time of an application is activated or deactivated. In change scheduling policy, all those queues do not consider for scheduling whose completion time do not meet permissible time of that application. The CFS faces the problem of fragmentation loss.

Our proposed system involves change queuing policy and change scheduling policy, the change scheduling policy is based on GA. GA gives the optimal combination of changes which increases the throughput and minimizes the average change delay of the system. The pseudo code of our proposed algorithm GA-CFS is given in Figure 4:

4. EXPERIMENT AND RESULTS

The experiments are carried out on the system having 2.27 GHz core 3 processor, 2GB Memory 32 bit window 7 home premium. Java 2 SE, JDK 1.6 is used to program the algorithm for simulating results.

```
Algorithm: GA-CFS
Step 1: Change Queuing Policy
Queue changes in the appropriate queue of application
Step 2: Run CFS parallel
Step 3: Change Scheduling Policy Using GA
    Step a: Initialize the population randomly
    Step b: repeat until the desire results achieved or
    maximum iteration
    Step c: calculate the fitness of individuals
    Step d: selection; select the two chromosomes to
    create new offspring
    Step e: Crossover, Mutation
    Step e: Repair
    Step f: Finish
```

Figure 4: Algorithm: GA-CFS

Our simulation model consist 5 applications 4 servers as shown in Figure 1. The changes arrive in continuously having random execution time between 1 and 4. The changes association to application is random. For example when 4 changes arrives it can be associated with application a1, a2 or a2, a5 etc. The arriving time of changes is between 1 and 24. The time window for application a1, a2, a3, a4, a5, are {1-8}, {6-13}, {12-19}, {11-20}, and {17-24} respectively. All the results are obtained by averaging over 24000 hours.

The parameters for GA are set as below:

- Population: 30
- Generation: 5
- Crossover Rate: 80%
- Mutation Rate: 20%

The above parameters are set through empirical knowledge. The generation is kept to 5 because the available solutions are not too much. The population and generation parameter can be changed according to solution space.

The results of GA-CFS, CFS, i-CFS, and Rand algorithm are compared in this section. i-CFS (incremental CFS) schedule changes one by one but break the changes when current permissible time window of the application ended [1]. Rand

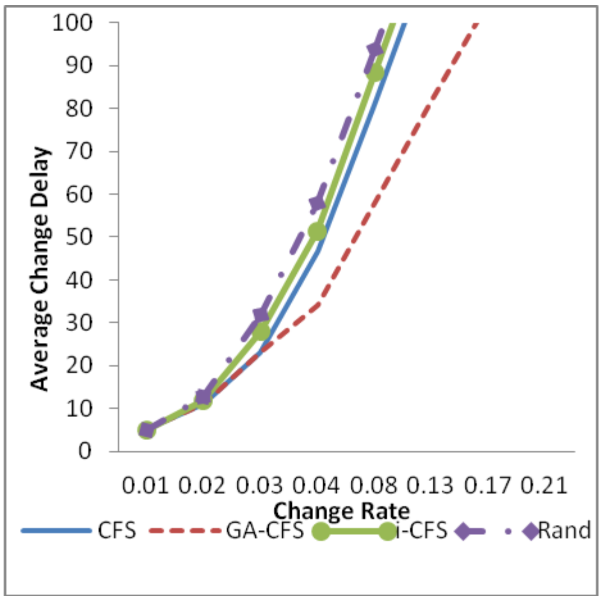


Figure 5: average change delay of CFS, GA-CFS, i-CFS, and Rand

work in the manner when a scheduling opportunity is available the selection of changes is done at random from the set of available changes.

The two major performance metrics observed are Change Delay, Throughput. Average change delay means, the average amount of time a change needs to wait before scheduling, whereas the throughput means the number of changes scheduled by a policy during a particular time window. The two metrics are plotted against change rate, which is the number of changes arrive in a day.

Figure 5 shows the average change delay. Initially both CFS and GA-CFS perform equal but when the change rate increased the result of GA-CFS is better than of CFS. i-CFS and Rand have performed less than both CFS and GA-CFS.

Figure 6 shows the average throughput of CFS and GA-CFS algorithms. Both algorithms performed same but GA-CFS shows some good results when the change arriving rate increased. Initially all the changes available are implemented by both algorithms within time windows of applications, but when the change arrival rate increased all the changes cannot be implemented within the time window. Therefore competition among changes involves, in this scenario GA performs excellent because it checks different combination.

The performance comparison of CFS and GA-CFS shows that GA-CFS performs well in both performance measures i.e.,throughput and average change delay. The system will perform well if the throughput of a system is high and average change delay of a system is low. In order to observe the effect of high change rate on Throughput, we increased the change arrival rate. Figure 7 shows difference of CFS and GA-CFS as the arrival rate increased. Whenever there is a competition among events GA gives better solution, because it gives good solution. GA also overcomes the fragmentation loss to some extent.

In the simulation results as shown in figure 5, 6 and 7, the performance of i-CFS is close but lower than CFS due to fragmentation loss, where as Rand algorithm performs lower than all the other algorithms because of the random nature of selecting changes.

5. CONCLUSION

In our work we have provided a novel approach to optimize change delay and throughput using genetic algorithm. The performance of GA is distinct when the load is high i.e. the arrival rate of change is high. Because GA assists when there are more candidates for a particular time to perform. GA provides an optimal candidate through its natural selection.

This work can be enhanced by tuning the operators for GA. Our model can also provide good solution for other network scheduling problems like in multi channel access point. As a future work, we like to provide other meta–heuristics to solve such problems.

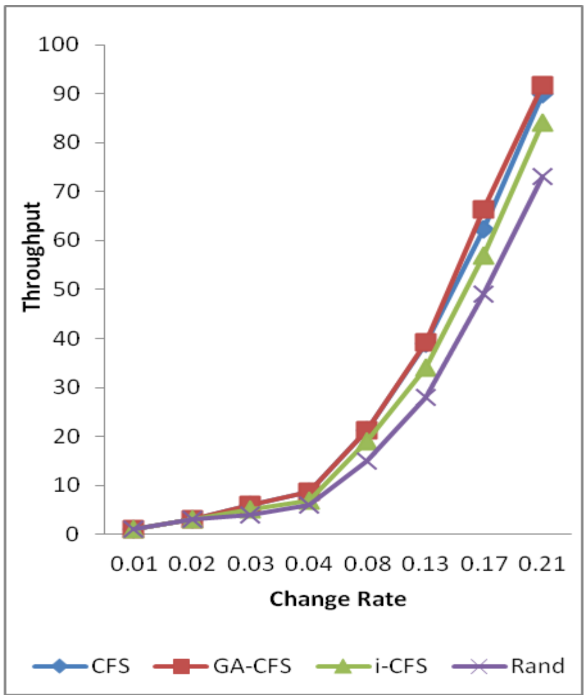


Figure 6: Average throughput of CFS, GA-CFS, i-CFS, Rand

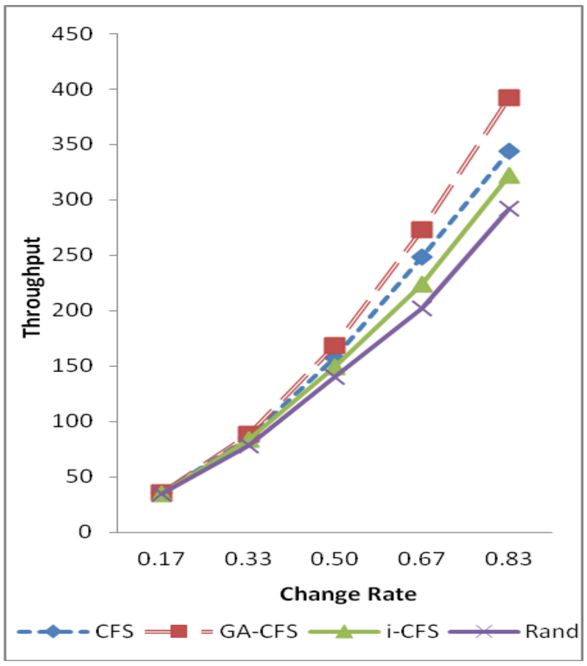


Figure 7: Average throughput of CFS, GA-CFS, i-CFS and Rand in rapid increased change rate

6. REFERENCES

[1] P. Kumar et al., "Change Management in Enterprise IT Systems: Process Modeling and Capacity-optimal Scheduling", Proc. IEEE INFOCOM 2010.

[2] L. Tassiulas and A. Ephremides, "Stability properties of queueing systems and scheduling policies for maximum throughput in multihop radio networks", IEEE Trans on Automatic Control, 37(12), 1992.

[3] Davis L (Ed) (1991): "Handbook of Genetic Algorithms". New York: Van Nostrand Reinhold.

[4] X.Luo, K.Kar et al., "On Improving Change Management Process for Enterprise IT Services", Proc. IEEE International Conference on Service Computing, Washington, DC, USA, 2008.

[5] K. Kar, X. Luo and S. Sarkar, "Throughput-optimal Scheduling in Multichannel Access Point Networks under Infrequent Channel Measurements", Proc. IEEE Infocom 2007, Anchorage, AK, May 2007.

[6] S.S. Rawat, L. Rajamani, “A Timetable Prediction for Technical Education System Using Genetic Algorithm”, Journal of Theoretical and Applied Information Technology, JATIT 2005-2010.

[7] L.M.Schmitt, “Fundamental Study Theory of Genetic Algorithms”, International Journal of Modeling and Simulation Theoretical Computer Science. 2001.

[8] J.H. Holland, Adaptation in Natural and Artificial Systems, 2nd, MIT Press, Cambridge, MA, 1992.

[9] B. Sigl, M. Golub, and V. Mornar, “Solving Timetable Scheduling Problem Using Genetic Algorithms”, 25th International Conference Information Technology Interfaces, Cavtat, Croatia, (2003).

[10] S. Ghaemi and M. T. Vakili, “Using a Genetic Algorithm Optimizer Tool to solve University Timetable Scheduling Problem”, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran, TR-2006-2, (2006).