This paper appeared in the *Proceedings of the 16th IEEE International Conference on Computer Communications and Networks* (ICCCN), Honolulu, Hawaii, Aug. 2007.

Interplay of Security and Reliability using Non-uniform Checkpoints

Kiranmai Bellam[†], Raghava K. Vudata[‡], Xiao Qin[†*], Ziliang Zong[†], Xiaojun Ruan[†], Mais Nijim[§]
Department of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849[†]
{kbellam, xqin, zzong, xruan}@ eng.auburn.edu
Department of Computer Science
New Mexico Institute of Mining and Technology, Socorro, NM 87801[‡]
Department of Computer Science
University of Southern Mississippi, Hattiesburg, MS 39406[§]

Abstract

Real time applications such as military aircraft flight control systems and online banking are critical with respect to security and reliability. In this paper we presented a way to integrate both by considering confidentiality and integrity services for security and non-uniform checkpoint strategy for reliability. The slack exploitation interacts in subtle ways for security in regards to the placement of checkpoint. The checkpoints are placed in to the task at low frequency in the beginning because the slack available can accommodate a large amount of work at risk and the frequency is increased there after considering the slack available. The security is applied to the data in two ways. First method introduces the security for the entire data at once whereas in the second method the data is divided into n uneven sections and each section is separately secured .That is at the start of the task basic security services are considered depending on the slack available. The security is increased gradually for the rest of the task but if there exist a fault, then at that point the security is maintained at the steady rate because of the limited slack. Compared to the first method the second method can provide up to a 32.3 percent higher security. While compared to the traditional checkpoint strategy, the non-uniform checkpointing makes more efficient use of slack while increasing the overall security levels by 34.4 percent for the second method.

1. Introduction

True in the realm of computers, the need for developing systems with fault tolerance and security is ever increasing. The systems have to be developed such that they can operate successfully and securely even with any fault occurrence in the execution. It is to be noted here that the task has to be completed within the given deadline. In any system when a fault occurs, it has to be detected and recovered before the time limit. The Slack that exists in the schedule of a task is utilized for the purpose of implanting the security and reliability. Achieving fault tolerance and security at the same time demands efficient exploitation of the total slack available in the system. This paper presents the study on the same issue showing how this slack can be exploited in developing a secured and reliable real-time system. We implement Checkpoint placement strategy to recover real time systems from the failures during the task's execution. These checkpoints can be inserted in a uniform or Non-uniform pattern. We already proved that the security can be increased by 385% while achieving the tasks reliability using uniform checkpoint strategy [2].

To further increase the security, in this paper, we adopt the non- uniform checkpoint strategy to develop a highly efficient fault tolerant and secure system. Here, the security can be achieved in two ways, either by providing the security over head for the entire task at once or by providing the security overhead after each checkpoint. The later method provides a higher security by up to a 32.3 percent than the former method and 34.4 percent than the uniform checkpointing. Either ways the strategies proposed here ensure more security than the ones achieved by the uniform checkpoint strategy. In our study, for simplification, we assume that only one fault can occur during the execution of any task. Once the fault is detected, we can eliminate further placement of checkpoints. This process aids us in saving the slack for

[*] Corresponding author. http://www.eng.auburn.edu/~xqin

the implementation of security. It is to be noted that in our study, we assume all deadlines for the tasks are soft and hence our algorithm is applicable for Soft-real time systems. The results we obtained are compared to the results from the uniform checkpoint approach. It is observed that our method could improve the fault-tolerance and security up to 34.4 percent.

The rest of the paper is organized as follows. Section 2 describes the related work and section 3 details the real time application followed by a fault tolerant model. Section 4 briefs the security methods that are used in this work. Section 5 gives a detailed explanation of how security and reliability can be integrated when non uniform checkpoint placement strategy is used. It also explains the two methods that are used for security over heads. Section 6 gives the performance evaluation followed by a summary in Section 7.

2. Related Work

An extensive research has been carried out in developing Fault tolerant real time systems. For the real time systems where storage was not a primary concern, conventional approaches with low over head like timeline or back up approaches are used in developing the system to be fault-tolerant [5].The transient faults were recovered by the re-execution of the tasks [4][7].

To design a fault tolerant distributed shared memory systems Sultan et al. developed an algorithm with very efficient checkpoint strategy and log Management [6]. Researchers have also worked on implementing a variety of fault tolerant activities involving both static and dynamic Scheduling. These activities assured the tasks with in the time limit and minimized the number of reliability activities [8]. On the other hand, same amount of research has been done concentrating on the security of the systems. Song et al. developed a security driven scheduling algorithms for Grids [1]. In our previous Study, We designed and implemented Security aware scheduling algorithms to protect against diversity of threats and attacks in a clustering environment [8]. Bertram et al. developed a set of algorithms for security constrained optimal power flow (SCOPF) and their configuration in an integrated package for real-time security enhancements [9].

The literature proves that very limited or no research has been done in integrating the fault tolerance and security in real-time systems .Myers et al proposed a method of building a trustworthy distributed system by the process of Construction. [10]

In our previous research, we have proposed an approach to integrate the fault tolerance and security in real-time systems by implanting uniform checkpoint placement [2]. We are extending the work further here to increase the levels of security and reliability.

The main contributions of this paper are 1.An approach to integrate the security and reliability using non uniform checkpoints is proposed. 2. Security implementation has been derived in two different methods. 3. A way to combine the confidentiality and integrity over heads has been proposed in method 2.

3. Non-Uniform Distribution of Checkpoints for Fault Recovery

The basic idea behind our approach is to recover soft real-time systems from transient failures by inserting checkpoints. It should be noted that checkpoint insertions can be accomplished in a uniform or non-uniform manner. We already presented the work for uniform checkpoint placement strategy [2]. In this work we are considering the non uniform checkpoint placement strategy to further increase the security and reliability of the task.

3.1 Real-time applications

A real-time application in this study is envisioned as a set of tasks, which have to be completed before their specified deadlines to accomplish an overall mission. We assume a real-time task *t* has a deadline *D* derived from hard real-time constraints. It is assumed that task *t* takes *C* number of CPU cycles for its worst-case execution. Cycle consumption of memory references vary with processor speed. Hence *C* is considered as the independent of the processor speed. Abundant research has been done to prove the degree of pessimism in the definition of *C* [3].

The time remaining to reach deadline *D* after considering the worst-case execution time *C* of the real-time task is the slack time denoted as S_l. This slack time S_l can be used to recover from transient faults in a real-time system.

3.2 Non-uniform checkpoints

Checkpoints are inserted in a real-time task to efficiently recover a task from any transient failure occurred during the execution. If there is a failure in the execution of the task then the failure is detected before the next checkpoint is placed. Hence it makes easy to re-execute the task from the previous checkpoint, where the failure occurred. Transient failures during the course of the task's execution can be readily detected by applying any of the already existing fault-detection techniques. Without loss of generality, it is assumed that no more than one transient failure might occur during the execution of the task. Similar assumptions were imposed in other studies [3]. The process we describe here (see Fig. 1.)

may not be applicable to a real-time system where a real-time task must be re-executed entirely when a failure occurs. This is because available slack S_l may not be sufficient to accommodate such a re-execution.
Our goal, of course, is to efficiently leverage slack time S_l to improve quality of security and to provide fault recovery. The overhead involved with generating checkpoints is modeled as *r*. It is assumed that in task *t*, *n* checkpoints are inserted to recover the system from failures if there exists any.

Therefore, the total time spent in creating all the checkpoints and to do the corresponding diagnosis is expressed as *n*r*, but we did consider that no more than one fault can occur during the task's execution so if a fault is detected, then there after it is not necessary to insert checkpoints. Hence, the over ahead involved with checkpoints relies on where the fault occurred. Fig 1(a) shows the uniform distribution of checkpoint [2].

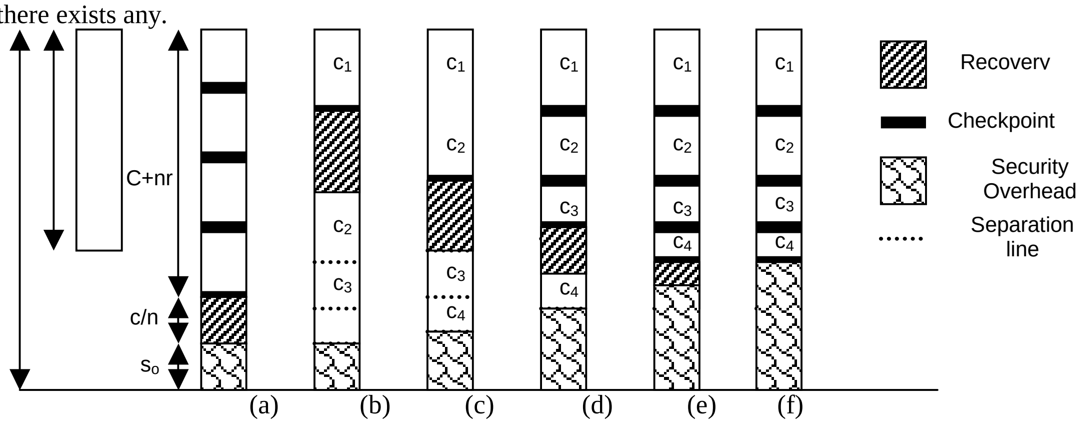

Fig.1 Uniform and Non-uniform distribution of checkpoints with security over head

As the approach we considered here is non-uniform, the *n* checkpoints are placed in the task *t* such that the *C* cycles of *t* are divided into *n* sections based on the below formulae

$$nx + (n-1)x + (n-2)x + ... + 2x + 1x = C . \quad (1)$$

$$x = \frac{2C}{n(n+1)} . \quad (2)$$

After calculating the value of *x* from (1) checkpoints are inserted in a way such that

$C_1+C_2+C_3+\ldots+C_{n-1}+C_n = C$,
where $C_1<C_2<C_3<\ldots<C_{n-1}<C_n$.

The main idea behind this approach is that at the beginning of the execution the task hasn't consumed much of available slack and therefore, low frequency of checkpoints is possible because most of the slack is available to accommodate a large amount of work at risk [3]. Similar to the overhead of checkpoints the recovery time also depends on where the fault occurred. That is if a fault is detected at the first checkpoint then the amount of rollback required would be *nx* because the first checkpoint is placed after *nx* as shown in fig 1(b). If the fault is detected at the second checkpoint then the rollback would be *(n-1)x* as shown in fig 1(c), similarly it would follow as *(n-2)x* and *(n-3)x* and so on for the faults that are detected there after as shown in fig 1(d) and 1(e) respectively. Fig 1(f) shows the distribution of checkpoints where a fault did not occur.

Hence a task *t* should satisfy the below equation to execute before the deadline and to guarantee the fault tolerance. Assume that a fault might have occurred at section k where $0 \le k \le n$ then

$$D \ge \sum_{i=1}^{k} (r + C_i) + C_k + \sum_{i=k+1}^{n} C_i + S_o , \quad (3)$$

where $\sum_{i=1}^{n} C_i = nx + (n-1)x + ... + 2x + 1x = C \cdot$

Here the term $\sum_{i=1}^{k} (r + C_i)$ represents the execution of a task up to a failure at a section k. The term C_k represents the roll back time and the term $\sum_{i=k+1}^{n} C_i$ represents the execution of the remainder of the task with no checkpoints because we considered that a fault can occur not more than once. Hence further insertion of checkpoints at this point is not required. And the last term S_o represents security over head which will be discussed in detail in the section 4.

It is intuitive to illustrate that the more the number of checkpoints made in a real-time task the more the task is fault tolerant. However, it is imperative to carefully choose the number of checkpoints, because there is a chance that the overhead might exceed slack time S_l, which makes it infeasible to finish the task before its deadline. Also, if the value of *n* is much smaller then the value of *nx* will be comparatively bigger, which can be inferred from (2), and the slack available might not be sufficient for the rollback if a fault is detected in this section. Hence the value of n has to be increased. This can be easily checked by the following inequality, $nx < S_l$.

If the above inequality is not true then the value of n has to be increased by 1 until the inequality is true.

4. Security Implementation

Security has become an important requirement for modern real-time systems. There are a wide variety of security mechanisms that can be applied to real-time applications. However, improving security inevitably leads to high overheads. Thus it is difficult if not impossible to implement all security mechanisms available to a real-time task. As confidentiality and integrity are the two major security issues that real-time tasks may encounter, we described the overhead model for these security services in our approach. It is worth noting that our approach can readily incorporate other types of security services like Availability to achieve higher security. In our previous work we integrated the confidentiality with reliability using uniform checkpoints here we are extending the security levels to confidentiality and integrity with non uniform checkpoints to further increase the ranking of security provided to the task [2].

Table 1: Cryptographic algorithms for confidentiality

S.No	Cryptographic Algorithms	Security Level (sl_i^c)	μ_i^c (KB/ms)
E1	SEAL	0.08	168.75
E2	RC4	0.14	96.43
E3	Blowfish	0.36	37.50
E4	Knufu/Khafre	0.40	33.75
E5	RC5	0.46	29.35
E6	Rijndael	0.64	21.09
E7	IDEA	1.00	13.5

4.1. Confidentiality model

Confidentiality counters snooping, which is unauthorized interception. Confidentiality is achieved by the implementation of cryptographic algorithms. Table 1 depicts the cryptographic algorithms [2][8] that are used in this research. Sl_i signifies the security level assigned to each algorithm in the range from 0.08 to 1. The strongest encryption algorithm IDEA is assigned the maximum-security level 1 and its performance is 13.5KB/ms.The security levels for the other algorithms are normalized by applying the following equation [2][8].Assume that l_i is the amount of data whose confidentiality has to be guaranteed for task t_i.

$$sl_i^c = 13.5 \big/ \mu_i^c \text{ where } 1 \le i \le 7 \tag{4}$$

$$l_i / \mu_i^c = P \tag{5}$$

That is P is the amount of overhead involved in encrypting the data.

4.2 Integrity model

Integrity counters the unauthorized change of information called alteration. Integrity is achieved by the implementation of Hash functions. The hash functions [2][8] that are considered in this work are depicted in Table 2. Each hash function is assigned a security level in the range from 0.18 to 1. The strongest and yet slowest Hash function Tiger is assigned a maximum security level of 1 and its performance is 4.36 KB/ms.

The security levels for other Hash functions are calculated by using the formula

$$sl_i^g = 4.36 \big/ \mu_i^g \text{ where } 1 \le i \le 7 \tag{6}$$

Table 2: Hash Functions for Integrity

S.No	Hash Function	Security Level sl_i^g	μ_i^g :KB/ms
H1	MD4	0.18	23.90
H2	MD5	0.26	17.09
H3	RIPEMD	0.36	12.00
H4	RIPEMD-128	0.45	9.73
H5	SHA-1	0.63	6.88
H6	RIPEMD-160	0.77	5.69
H7	Tiger	1.00	4.36

Assume that l_i is the amount of data whose integrity must be achieved for task t_i.

$$l_i \big/ \mu_i^g = Q\,. \tag{7}$$

The term security over head S_o is the combination of the security over heads of the confidentiality and integrity for the corresponding amount of data.

5. Integrating Reliability and Security with Non-uniform Distribution of Checkpoints

The implementation of security along with reliability in a real time task can be done in two ways. First method considers the over head involved for the entire data that has to be secured at once, that is if we are considering confidentiality then based on the requirement any algorithm is applied to encrypt the data all at once, similarly for integrity. That is security over head $S_o = P$ or Q. For simplicity, we considered the implementation of confidentiality and integrity individually for this method. The security over head that can be utilized for a particular task t can be calculated using the formulae

$$\mu_i^c = \frac{l_i}{D - \sum_{i=1}^{k}(r + C_i) + C_k + \sum_{i=k+1}^{n} C_i}. \qquad (8)$$

Equation (8) is derived from equation (3) and (5) for the confidentiality over head.

$$\mu_i^g = \frac{l_i}{D - \sum_{i=1}^{k}(r + C_i) + C_k + \sum_{i=k+1}^{n} C_i}. \qquad (9)$$

Equation (9) is derived from equation (3) and (7) for the integrity over head. Any mapping function can be used to map the values obtained from equation (8) and (9) to find the corresponding algorithm that provides maximum security to the task.

Whereas in the second method, the data is divided in to n uneven sections, same as the tasks CPU cycles, using the equation

$$nx + (n-1)x + (n-2)x + ... + 2x + 1x = l\ , \qquad (10)$$

where l is the amount of data that has be secured for the entire task. x in Eq. (10) can be written as

$$x = \frac{2l}{n(n+1)}. \qquad (11)$$

Once the data is divided in to n sections, the security is implemented to each and every section separately.
The equation below shows the implementation of reliability and security for n different sections of data using non uniform checkpoint strategy.

$$D \geq \sum_{i=1}^{k}(r + C_i + \frac{l_i}{\mu_i^c} + \frac{l_i}{\mu_i^g}) + C_k + \sum_{i=k+1}^{n}(C_i + \frac{li}{\mu_k^c} + \frac{li}{\mu_k^g}) \qquad (12)$$

For any given task t first the number of checkpoints n that are required for fault recovery are assumed, then the CPU cycles C and the amount of data l that has be to secured is divided in to n sections using the formulae (1) and (10) and then the task is executed by placing a checkpoint after each section and also applying the security mechanisms for each section separately. For the first section the basic confidentiality and integrity are provided by choosing the corresponding E1 and H1 algorithms from Tables 1 and 2. For the next section the security can be increased by one that is, E2 and H2 can be used for security. Similar way is used to increase the security for other sections, but suppose if a fault is occurred at section k then the security mechanisms Ek and Hk that are used for that particular section will be used for the rest of the section too, because at this point a portion of slack has been already utilized for the roll back so further increasing the security might decrease the slack even more resulting in an effect which might leave the task's execution beyond the deadline, which is not a feasible solution. Hence the security at this point is not increased. In equation (12) the term $\sum_{i=1}^{k}(r + C_i + \frac{l_i}{\mu_i^c} + \frac{l_i}{\mu_i^g})$ represents the execution of task by placing checkpoints for fault detection and using the algorithms of confidentiality and integrity for security. The security algorithms are increased after each section by one until a fault has been detected at section k. The term C_k represents the roll back of the section where a fault occurred. The last term $\sum_{i=k+1}^{n}(C_i + \frac{li}{\mu_k^c} + \frac{li}{\mu_k^g})$ represents the execution of the rest of the task with out any checkpoints. The security mechanisms that will be used for the rest of the sections is also going to be same as the one that has been used for the section k, because of the limited slack that is available.

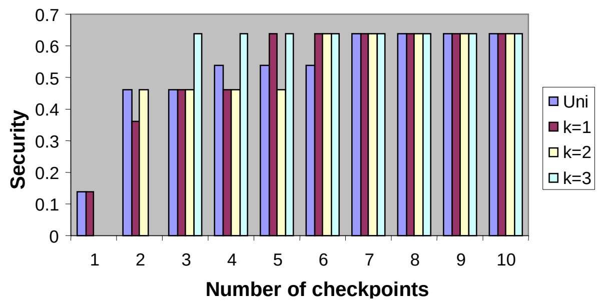

Fig. 2. Confidentiality levels for uniform checkpointing and Non-Uniform checkpointing. Fault occurred at K=1,2,3.

6. Performance Evaluation

Fig.2 depicts how the confidentiality levels vary when the checkpoints are placed in Uniform and Non-Uniform pattern. Here K=1, 2, 3 represents that fault might have occurred in section 1, 2, 3, respectively in Non-Uniform checkpoint strategy. Uni is used to represent the uniform checkpoint strategy. For uniform checkpoints, the confidentiality level gradually increases with increase in the number of checkpoints from a minimum, whereas in the case of Non-Uniform checkpoints, we observe a lot of variation depending up on where the fault is occurred. The higher the values of k the higher the security levels are. However, the average confidentiality level guaranteed in non-uniform checkpoint strategy is either higher or equal to the average confidentiality levels in uniform checkpoint strategy. It is to be noted from the graph that the Confidentiality levels corresponding to k=2, n=1 and k=3 and n=1, n=2 are zero because when we assume only one checkpoint in a task, the possibility of K=2 is impossible because $0 \le k \le n$.

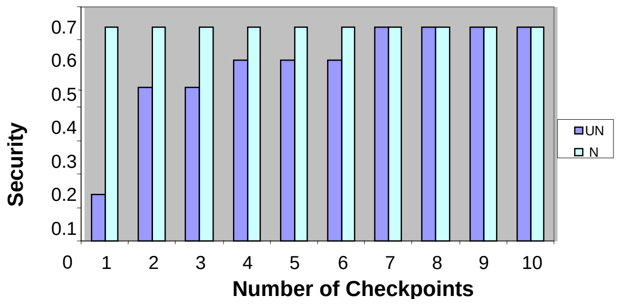

Fig. 3. Uniform Vs Non Uniform Checkpoints for confidentiality

Fig.3 indicates the improvement of confidentiality levels achieved by adopting Non-Uniform checkpoint placements over the Uniform checkpoint strategy. Here security for the entire data is considered at once. NU represents average security levels for non uniform checkpoint strategy where different k is considered.

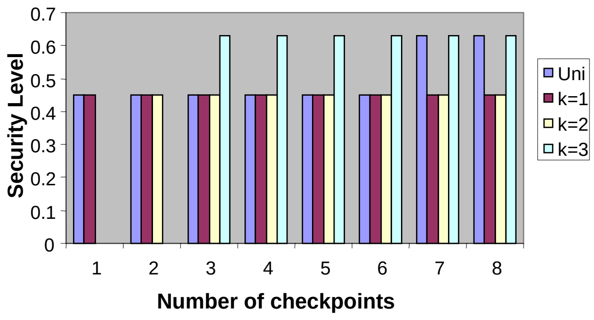

Fig. 4. Integrity levels for Uniform checkpointing and non uniform checkpointing when fault occurred at k=1,2,3.

Fig 4 shows the effect on integrity levels when uniform and non-uniform checkpoints are considered for *k*=1,2 and 3. For uniform checkpoints the security level kept increasing with the number of checkpoints whereas for the non uniform checkpoints the security levels either increase or remain constant for different k that is the values are either greater than or equal to the values in the uniform checkpoints. This shows that non uniform approach performs better than uniform approach.

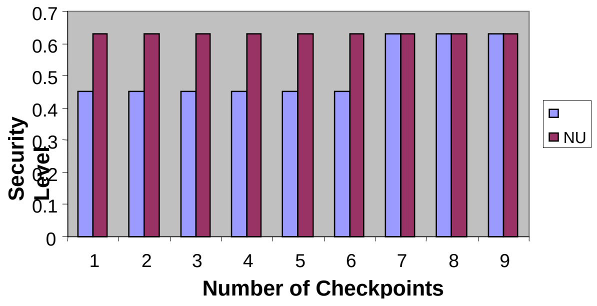

Fig. 5. Uniform Vs Non Uniform checkpoints for Integrity

Fig. 5. shows the difference of integrity levels for uniform and non uniform checkpoints when the security is applied to the entire data at once. Same as the confidentiality, the integrity levels are either the same are higher than the uniform checkpoint integrity levels. With this observation it can be stated that irrespective of the values of *k*, the non uniform checkpoint strategy is definitely better than the uniform checkpoint strategy.

7. Summary

This paper presents an approach to seamlessly integrating the security and reliability. Confidentiality and integrity services are considered for security, and non-uniform checkpoint placement strategy is employed for reliability. Two methods are proposed in combining the security with reliability in real time tasks. First method considers security overheads for an entire data at once. In the second method, the data is divided into *n* different sections; the security overhead for each section is considered separately. The simulation results show that the non uniform checkpoint strategy gives a 34.4% higher security than the uniform checkpoint when the second method is considered for security. In addition, the second method is better than the first method by 32.3%. The overheads for all the schemes are analytically modelled. The approach presented in this paper maximizes both security and reliability for real-time applications in an innovative way.

Acknowledgments

The work reported in this paper was supported by the US National Science Foundation under Grant No. CCF-0702781, Auburn University under a startup grant, New Mexico Institute of Mining and Technology under Grant No. 103295, the Intel Corporation under Grant No.

2005-04-070, and the Altera Corporation under an equipment grant.

8. References

[1] S. Song, Y. K. Kwok, and K. Hwang, "Trusted Job Scheduling in Open Computational Grids: Security-Driven Heuristics and A Fast Genetic Algorithms," *Proc. Int'l Symp. Parallel and Distributed Processing*, 2005

[2] K. Bellam, Z. Zong, M. Alghamdi, M.Nijim, X Qin, "Integrating Fault Recovery and Quality of Security in Real-Time Systemsm," *Proc. IEEE Int'l Symp. Ubisafe Computing, Ontario*, Canada, May 2007.

[3] Rami Melhem, Daniel Mosse, Elmootazbellah (Mootaz) Elnozahy, "The Interplay of Power Management and Fault Recovery in Real-Time Systems," *IEEE Trans. Computers*, vol. 53, no. 2, pp. 217-231, Feb. 2004.

[4] S. Ghosh, D. Mosse, and R. Melhem, "Implementation and Analysis of a Fault-Tolerant Scheduling Algorithm," *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 3, Mar. 1997.

[5] Liestman and R. Campbell, "A Fault-Tolerant Scheduling Problem," *IEEE Trans. Software Eng.*, Nov.1986

[6] F. Sultan, T. Nguyen, L. Iftode, "Scalable Fault-Tolerant Distributed Shared Memory," *Proc. ACM/IEEE Conf. Supercomputing*, 2000.

[7] S. Ramos-Thuel and J. Strosnider, "Scheduling Fault Recovery Operations for Time-Critical Apps," *Proc. IFIP Conf. Dependable Computing for Critical Apps*, Jan. 1995.

[8] T. Xie and X. Qin, "Scheduling Security-Critical Real-Time Applications on Clusters," *IEEE Trans. Computers*, vol. 55, no. 7, pp. 864-879, July 2006.

[9] Bertram, T.J.; Demaree, K.D.; Dangelmaier, L.C., "An integrated package for real-time security enhancement," *IEEE Trans. Power Systems*, May 1990.