International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-1 Issue-2, July 2012

Performance Evaluation of SDS Algorithm with Fault Tolerance for Distributed System

Sathiya Bharathi K, Kumaresan N

Abstract: ***In the recent past, Security-sensitive applications, such as electronic transaction processing systems, stock quote update systems, which require high quality of security to guarantee authentication, integrity, and confidentiality of information, have adopted Heterogeneous Distributed System (HDS) as their platforms.We systematically design a security-driven scheduling architecture that can dynamically measure the trust level of each node in the system by using differential equations and introduce SRank to estimate security overhead of critical tasks using SDS algorithm.Furthermore,we can achieve high quality of security for applications by using security-driven scheduling algorithm for DAGs in terms of minimizing the makespan, risk probability, and speedup. In addition to that the fault tolerant is included using Security Driven Fault Tolerant Scheduling Algorithm (SDFT) to tolerate N processors failure at one time, and it introduced a new global scheduler to improve efficiency of scheduling process.Moreover, the SDFT supported flexible security policy applied on real time tasks according to its security requirement and considered the effect of security overhead during scheduling. We also observe that the improvement obtained by our algorithm increases as the security-sensitive data of applications increases.***

Index Terms: ***Directed acyclic graphs, scheduling algorithm, security overheads, heterogeneous distributed systems, security-driven, fault tolerance, precedence-constrained tasks.***

I. INTRODUCTION

Heterogeneous distributed systems (HDSs) are usually composed of diverse sets of resources with different capabilities and interconnected with arbitrary networks to meet the requirements of widely varying applications. Over the last decade, HDSs have been emerging as popular computing platforms for compute-intensive security-sensitive applications, such as electronic transaction processing systems, stock quote update systems, E-commerce online services, and digital government; have started to employ HDSs as their platforms.

Traditional scheduling algorithms have paid too much attention to improving the efficiency of scheduling process and neglected the reliability and security requirement of scheduled tasks, which makes it unsuitable to real time system.

Revised Manuscript Received on 30 July 2012
*****Correspondence Author(s)
K.Sathiya Bharathi*, PG Student, Department of ECE, Anna University of Technology, Coimbatore, India.,
N.Kumaresan, Assistant Professor, Department of ECE, Anna University of Technology, Coimbatore, India.

Thereby, it's necessary to introduce a new challenge in resource allocation domain, which could adopt both fault tolerance and security technique.

Different nodes can provide different security level for users, even at the same level of security service, and different amount of computation leads to distinct security overheads. In this paper, we design and evaluate a security-driven scheduling model, which can dynamically compute security overheads and then, integrates security services into that scheduling model for parallel applications with precedence-constrained tasks in HDSs. We create a trust method formulated using differential equations to dynamically compute the trust level of nodes in such HDSs.In addition, fault tolerant is included to tolerate N processors failure at one time, and it introduced a new global scheduler to improve efficiency of scheduling process. Moreover, the Security Driven Fault Tolerant Scheduling Algorithm (SDFT) supported flexible security policy applied on real time tasks which achieved best fit security level according to its security requirement.

The rest of paper is organized as following: In Section II, we outline the related work in this domain. In Section III, we describe the system architecture and the parallel application scheduling model. In Section IV, we propose a method that can dynamically compute the trust level of security service provided to each node, security overheads, and risk probabilities. In Section V, we present a security-driven allocation scheme and investigates the properties in HDSs.Then,we present the SDFT algorithm in Section VI and its performance was analysed in section VII and conclude the paper in Section VIII.

II. RELATED WORK

Scheduling algorithms play a key role in obtaining high performance in HDSs. The objective of scheduling is to map tasks onto machines and order their executions so that task precedence requirements are satisfied with a minimal schedule length (make span). A popular representation is the directed acyclic graph (DAG) in which the nodes represent application tasks and the directed arcs or edges represent intertask dependencies, such as task's precedence. It is known as NP-complete. Most list scheduling algorithms are designed for homogeneous systems.

Several list scheduling algorithms have been proposed to deal with heterogeneous systems and ignore security issues, for example, dynamic-level scheduling (DLS) algorithms, Dynamic Critical Path (DCP), heterogeneous earliest-finish-time (HEFT) algorithm. Only few groups of researchers investigate the security-driven scheduling domain from different angles in various contexts.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Azzedin and Maheswaran suggested to integrate the trust concept into Grid resource management and are not suitable for HDS. Song et al , developed three risk-resilient strategies and a Space Time Genetic Algorithm (STGA), to provide security assurance in grid job scheduling. However, their algorithms cannot be applied in HDSs for parallel applications with security requirements for two reasons: first, fails to support nodes trust manager; second, their algorithms only consider batch scheduling, where jobs are independent of each other, and hence it cannot schedule parallel applications, where precedence constraints and communications among tasks exist in a single application.

Dhal and Liu extended the RM

Algorithm to Rate Monotonic First-Fit (RMFF) Algorithm, which supported scheduling real time tasks on multiprocessor system. Moreover, Alan and Luigi proposed Fault Tolerant RMFF (FFRMFF) algorithm based on RMFF which introduced fault tolerance technique to RMFF, but it didn't consider the security requirement of real time tasks and tolerated only one processor failure.Actually, most HDS applications are intertask-dependent. Hence, we are motivated to incorporate security and fault tolerant into intertask dependency scheduling, and to propose a scheduling algorithm to improve security of HDSs while minimizing the computational overheads.

III SYSTEM ARCHITECTURE

In this section, we first introduce the architecture and design of the Security-Driven Scheduling (SDS) and Security Driven Fault Tolerant Scheduling Algorithm (SDFT). Finally, we compare the performance of SDS and SDFT that is used in this paper. For ease of understanding, we summarize the notations and their meanings used throughout of this paper in Table 1.

A. Security-Driven Scheduling Architecture:

We propose a security-driven scheduling architecture wherein the main difference from general architecture is that it includes the security evaluation modules such as Trust Manager, Trust Value Computation, security Overhead Controller, as depicted in Figure.1

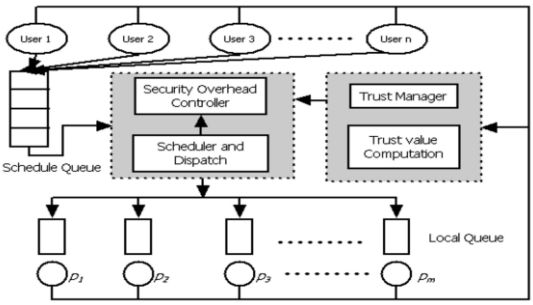


Figure. 1. Security-driven scheduling architecture.

Trust Manager Module is used to discover and collect security threats and performance metrics of every service provider (computational node) in HDSs. Some of the performance metrics aresuccess rate, speedup, memory, bandwidth, reliability, and so on. Trust Value Computation is used to compute trust level based on the information from Trust Manager module, which uses some trust models, such as Bayesian, Eigen Trust , and our new approach.

TABLE 1 DEFINITION OF NOTATIONS

Notation	Definition
ID	user identification
T	the set of v weighted tasks in the application
t_i	the ith task in the application
$w(t_i)$	the computational cost of task t_i
E	the set of weighted and directed edges representing communications among tasks in T
$e_{i,j}$	the directed edge from ith task to jth task
$w(e_{i,j})$	the communication cost of edge $e_{i,j}$
P	the set of m heterogeneous processors
p_i	the ith processor in the system
u_i	the execution speed of p_i
$SD_{i,j}^k$	the kth security requirement of task t_i for node p_j
$TL_{j,id}^k$	the kth trust level the node p_j provides for user id
$C_{i,j}^k$	the kth security overheads of task t_i for node p_j
$Pr(t_i^k, p_j)$	the kth risk probability of task t_i for node p_j
$Pr(t_i, p_j)$	the risk probability of task t_i for node p_j
$Pr(t_i)$	the risk probability of task t_i
$SRank(t_i)$	the security upward rank of task t_i
$succ(t_i)$	the set of immediate successors of task t_i
$pred(t_i)$	the set of immediate predecessor tasks of task t_i
$EST(t_i, p_j)$	the earliest computation start time of task t_i on processor node p_j
$EFT(t_i, p_j)$	the earliest computation finish time of task t_i on processor node p_j

Security Overhead Controller evaluates the security overheads for a task assigned to a node. Scheduler and Dispatch module is deployed to generate resource allocation decisions for each task in a parallel application to improve security of HDSs and to minimize the computational overheads, and then dispatches the task to the destination node. The Schedule Queue maintained by the admission controller is deployed to accommodate incoming applications. If workload is extremely high, the Schedule Queue becomes bottleneck of the system. This problem can be resolved by multiqueue techniques.

B. Parallel Application Model

Generally, a parallel application model with precedence constrained tasks is represented by a directed acyclic graph. G = <ID, T,E, SD>, where ID is the identification of user;T is the set of v tasks that can be executed on any of the available nodes (processors); E Є T X T is the set of directed arcs or edges between the tasks to represent dependency. For example, edge $e_{i,j}$ Є E represents the precedence constraint that task t_i should complete its execution before task t_j starts the execution. A task may have one or more inputs. When all of the inputs are available, the task is triggered to execute. When the task is finished, it generates the required outputs. The weight assigned to a task t_i represents the computational cost and the weight ($e_{i,j}$)assigned to edge $e_{i,j}$ represents the communication cost.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

A task with no parent tasks in DAG is called an entry task and a task with no child task in DAG is called an exit task. In this paper, we assume that DAG has exactly one entry task t_{entry} and one exit task t_{exit}.

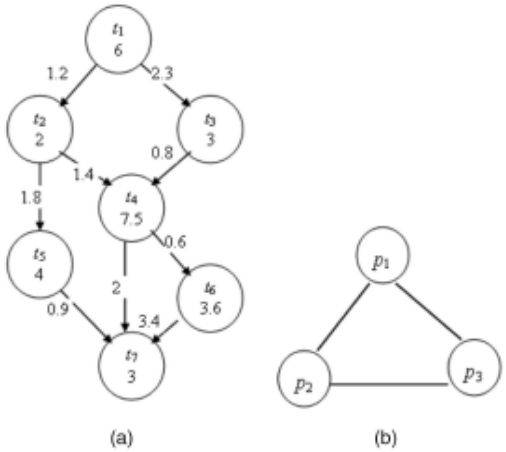


Figure. 2. An example of parallel application and heterogeneous Distributed system

Figure. 2a shows an example DAG with assigned task and edge weights. SD is security demand of the parallel task to all available nodes. The security demand of task ti for node pj can be specified as a q-tuple $SD_{i,j}=(SD^1_{i,j}, SD^2_{i,j}, \ldots SD^q_{i,j})$ represents the required security-level range of the kth security service ($1 \le k \le q$). Since snooping, alteration, and spoofing are three common attacks in HDSs .we consider three security services, which are authentication service, integrity service, and confidentiality service, to protect HDSs. Each task require three security services can be denoted as $SD^a_{i,j}$,, $SD^g_{i,j}$, $SD^d_{i,j}$ respectively. SD is specified by users from very low to very high to represent their security requirements are normalized into the range [0, 1].SD of seven tasks for three nodes as an illustrative example.

C. HDSs Model and Assumptions

In this paper, we consider heterogeneous computing environment model in Figure. 2b, in which the parameters of each machine are shown in Table 3. We assume that:

- Any node can execute/compute the task and communicate with other machines at the same time. The communication speed between two machines is also set as 1 without loss of generality.
- Once a node has started task execution, it continues without interruptions, and after completing the execution it immediately sends the output data to all the children tasks in parallel.

TABLE 2 Examples of Task Security Demands

Task	$SD^a_{i,1}$	$SD^g_{i,1}$	$SD^d_{i,1}$	$SD^a_{i,2}$	$SD^g_{i,2}$	$SD^d_{i,2}$	$SD^a_{i,3}$	$SD^g_{i,3}$	$SD^d_{i,3}$
t_1	0.4	0.1	0.6	0.2	0.3	0.1	0.2	0.5	0.5
t_2	0.2	0.3	0.2	0.6	0.1	0.1	0.5	0.3	0.4
t_3	0.8	0.4	0.7	0.3	0.2	0.5	0.1	0.2	0.6
t_4	0.1	0.1	0.3	0.4	0.4	0.4	0.3	0.7	0.5
t_5	0.4	0.2	0.4	0.1	0.7	0.3	0.4	0.4	0.4
t_6	0.2	0.7	0.5	0.8	0.3	0.2	0.3	0.3	0.3
t_7	0.6	0.5	0.1	0.9	0.2	0.9	0.4	0.5	0.5

TABLE 3 The Computational Node Parameters

Node	u_j	$TL^a_{j,1}$	$TL^g_{j,1}$	$TL^d_{j,1}$	λ^a_j	λ^g_j	λ^d_j
p_1	2	0.5	0.4	0.8	0.4	1.1	3
p_2	1	0.7	0.2	0.9	5.1	0.2	0.7
p_3	4	0.3	0.7	0.3	1.1	0.2	1.6

IV. SECURITY AND TRUST REQUIREMENTS

A.System Node Trust Model

Trust is a firm belief (attributes such as reliability, honesty, firewall capabilities, and antivirus capabilities) in the competence of a node to act as expected. In addition, this belief is not a fixed value associated with the node but rather it is subject to the user's behavior and can be applied only within a specific context at a given time.The trust level of node p_j can provide security services, such as authentication (a), integrity (g), and confidentiality (d) for users, and normalized in the range [0, 1] which is defined as follows:

$$TL^k_{j,id} = \varepsilon_j \times H(j,id,k,t) + \beta_j \times G(id,k,t),$$
$$K \in (a, g, d)$$
$$\varepsilon_j + \beta_j = 1,\ \varepsilon_j \ge 0,\ \beta_j \ge 0 \quad \ldots..(1)$$

The kth trustworthiness at a given time t between the node and the user, for example, node p_j for user id, denoted as $TL^k_{j,id}$ is computed based on the direct relationship at time t between node p_j and user id, denoted as H(j,id,k,t) as well as the reputation of user id at time t is denoted as G(id,k,t).Let the weights given to direct trust and reputation relationships be ε and β respectively.

B. Security Overhead Model

The same security level value in different security services may have various meanings. As $SD^k_{i,j}$ is the security level the task t_i requires including authentication, integrity, and confidentiality services provided by node p_j and is specified by the user. Let $C^k_{i,j}$ be the security overhead of the kth security service experienced by t_i on node p_j can be computed . The overall security overhead of ti on node p_j with security requirements for the three services above is modeled.

$$C^k_{i,j} = \begin{cases} 0, if\ SD^k_{i,j} \le TL^k_{j,id} \\ \dfrac{S_k\left(SD^k_{i,j} - TL^k_{j,id}\right)}{u_j}, \quad k \in (a,g,d) \\ if SD^k_{i,j} > TL^k_{j,id} \end{cases}$$

$$\begin{cases} C_{i,j} = \sum_{k \in (a,g,d)} w_k C^k_{i,j} \\ \sum_{k \in (a,g,d)} w_k = 1 \end{cases}$$

…….(2)

TABLE 4 The Parameters of Our Trust Model

parameter	value	parameter	value	parameter	value
ε_1	0.3	β_1	0.7	ε_2	0.4
β_2	0.6	ε_3	0.6	β_3	0.4
ξ	0.1	φ	0.5	ρ	0.55
μ	0.6	ν	0.7	δ	0.12

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

In our security overhead model, some techniques are used to ensure security service and lead to security overhead. Such as DES algorithm for confidentiality, HMAC-MD5 technique for authentication, RIPEMD-128 hash functions for integrity, and so on.

TABLE 5 Authentication Methods

Authentication Methods	$S_a(x)$:Security Level	Computation Time(ms)
HMAC-MD5	0.55	90
HMAC-SHA-1	0.91	148
CBC-MAC-AES	1	163

V. SECURITY-DRIVEN LIST SCHEDULING ALGORITHM

A.Task Priorities

Our security-driven scheduling algorithm will use security upward rank (SRank) attribution to compute tasks priorities starting from the exit task to entry task.

Definition: Given a DAG with v tasks and e edges and a system with m heterogeneous processors, the SRank during a particular scheduling step is a rank of task, from an exit task to itself, which has the sum of communication costs of edges, computational costs, and security overheads of tasks over all processors and is computed as follows.

$$SRank(t_i) = \frac{(\sum_{j=1}^{m} SRank(t_i, p_j)}{m} + max_{t_x \in succ(t_i)}(w(e_{i,x}) + SRank(t_x)),$$

$$SRank(t_i, p_j) = \frac{w(t_i)}{u_j} + c_{i,j} \quad \ldots\ldots\ldots(3)$$

Where SRank (t_i,p_j)is the security upward rank of task t_i on node p_j, $C_{i,j}$ is the security overhead computed and succ(t_i) is the set of immediate successors of task ti.

TABLE 6 The SRank Value Of Each Task

Task	t_1	t_2	t_3	t_4	t_5	t_6	t_7
SRank	52.608	37.714	39.797	32.664	14.266	17.662	6.321

B. Design of Algorithm

The main idea of our proposed algorithm is to incorporate the security overheads and the risk probability into scheduling. EST(t_i,p_j) and EFT(t_i,p_j) are the earliest start execution time and the earliest execution finish time of task t_i on processor node p_j, respectively. For the entry task t_{entry}, the EST is defined as

1. Compute *SRank* for all tasks by traversing Graph from *exit* task
2. Sort the tasks into a Scheduling list by Non-increasing order of *SRank*
3. **While** the scheduling list is not empty **do**
4. Remove the first task ti from the Scheduling list
5. **For** each node p_j ϵ P do
6. Compute EFT (t_i,p_j)
7. Compute Pr (t_i,p_j)
8. **End**

For the other tasks in DAG graph, EST and EFT values are computed recursively, starting from

EST (t_{entry}, p_j) = 0 ……(4)

Algorithm 1. The proposed SDS algorithm.

EST (t_{entry}, p_j) = 0 ……(4) In order to compute EST of a task t_i, all immediate predecessor tasks t_x of t_i must be scheduled.After all the tasks in the graph are scheduled, the schedule length (i.e,overall completion time) will be the actual finish time of exit task t_{exit}, thus the schedule length (which is also called makespan) is defined as follows:

Makespan = EFT (t_{exit}, p_j) …..... (5)

In heterogeneous systems, the risk coefficient λ^k_j is different from one to another.For example, node p_j can provide authentication with coefficient equals 1.3, integrity coefficient equals 3.2 and confidentiality coefficient equals 0.3.While the other node p_x may offer values 2.3, 0.1, 0.9, and 3.3, respectively. The negative exponent indicates that failure probability grows with the difference $SD^k_{i,j}$ -$TL^k_{j,id}$.

$$P_r(t_i^k p_j) = \begin{cases} 0 \ if \ SD_{i,j}^k \le TL_{j,id}^k \\ 1 - e^{-\lambda_j^k (SD_{i,j}^k - TL_{j,id}^k)}, \quad k \in (a,g,d) \\ if SD_{i,j}^k > TL_{j,id}^k \end{cases} \quad \ldots\ldots(6)$$

Speedup is computed by dividing the sequential execution time (i.e., cumulative computation costs and security overheads of the tasks in the graph) by the parallel execution time (i.e., the makespan of the output schedule). If the sum of the computational costs and security overheads is maximized, it results a higher speed up, but ends up with the same rank of the scheduling algorithms.

$$Speedup = \frac{min_{u_j \varepsilon U}\{\sum_{t_i \varepsilon T} w(t_i)/u_j + C_{i,j})\}}{makespan} \quad \ldots\ldots..(7)$$

Atlast, SDS finds a node with the earliest execution finish time and assigns task ti to this node.

VI. SDFT SCHEDULING ALGORITHM

This section provides a top-level framework of the proposed Security Driven Fault Tolerant (SDFT) scheduling algorithm which is mainly composed by four core components: Security Policy Loader, Global Task Assigner, Running State Monitor and Processor Failure Handler.

Step1. Load the Security Policy and compute the security overhead for all the tasks or copies.

Step2. (Global Task Assigning)Let the set of tasks Γ be arranged by RM priority decreasing ordering: τ_1, τ_2; Set processor set P= {p_1},and set the number of processors needed *M* =1; Repeat the following steps for every task in $\tau_{i \text{ in } \Gamma}$.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

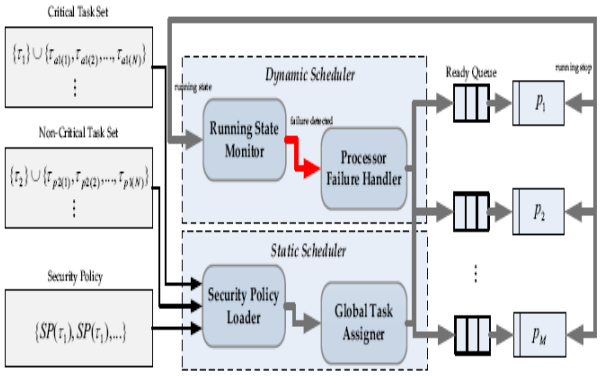


Figure3 Security Driven Fault Tolerant (SDFT) scheduling architecture

Step2.1. (Assign task or active copies) Let the *P* arranged by processor length increasing order, and search the processor p_j which can meet such conditions: (C1) The set { τ_i }U task(p_j) U *active* (p_j) is schedulable by means of Theorem 1. (C2) In case of S=1,2,...,N, for each p_f ={ $p_{f(1)}, p_{f(2)}, \ldots p_{f(s)}$} and $p_j \varepsilon p_f$, the set { τ_i}U task (p_j)*copy*(p_j, p_f) is schedulable by means of the Theorem 1. If processor p_j satisfied conditions, then set $P(\tau_i) = p_j$,*length* (p_j)+=C_i ; If no such processor exists, then assign task τ_i to new processor p_{M+1} , and set $P(\tau_i)=p_{M+1}$,*length*(p_{M+1})=0,$P=P\{p_{M+1}\}$; If task τ_i choices active copies then go to Step 2.2, else go to Step 2.3;

Step2.2. (Assign active copies) Do the following steps for every active copy ($\tau_{ai(K)}$) of task τ_i : Let the *P* arranged by processor length increasing order, and search the processor p_j ($p_j \neq P(\tau_i)$ *and* $p_j \neq P(r_a)$) which can meet below conditions: (C1) The set { τ_i}U task (p_j)*copy*(p_j, p_f) is schedulable on by means of the Theorem 1. (C2) In case of R=1,2,...,N, for each p_f ={ $p_{f(1)}, p_{f(2)}, \ldots p_{f(R-1)}$} U{ $P(\tau_i)$},the set { τ_{ai}}U task (p_j)*copy*(p_j, p_f) is schedulable by means of the Theorem 1. If processor p_j satisfied conditions, then set $P(\tau_i) = p_j$,*length* (p_j)+=C_i ; If no such processor exists, then assign task τ_i to new processor p_{M+1} , and set

$P(\tau_i)=p_{M+1}$,*length*(p_{M+1})=0,$P=P\{p_{M+1}\}$;

Step2.3. (Assign passive copies) Do the following steps for every active copy $\tau_{pi(k)}$ of task τ_i : Let the *P* arranged by processor length increasing order, and search the processor p_j ($p_j \neq P(\tau_i)$)*and* $p_j \neq P(r_p)$which can meet below conditions: (C1) In case of R=1,2,...,N, for each p_f ={ $p_{f(1)}, p_{f(2)}, \ldots p_{f(R-1)}$} U{ $P(\tau_i)$}, the set { τ_{pi}}U task (p_j)*copy*(p_j, p_f) is schedulable by means of the Theorem 1. If processor p_j satisfied conditions, then set $P(\tau_i) = p_j$,*length* (p_j)+=C_i ; If no such processor exists, then assign task τ_i to new processor p_{M+1} , and set $P(\tau_i)=p_{M+1}$,*length*(p_{M+1})=0,$P=P\{p_{M+1}\}$;

Step3. (Running State Monitoring (RSM)) Each task and its copies would be scheduled by RM algorithm on the assigned processor and send a message to notify RSM, after it completed. When received the message, RSM would operate by the following rules: If first message comes from task τ_i, then stop running all the copies of τ_i ; If first message comes from active copy τ_{pi} , then RSM stop running all the active copy and task τ_i ; If first message comes from passive copy τ_{pi}, then RSM stop running the other copies of τ_i. When processor failure occurs, go to Step4.

Step4. (Processor Failure Handling) Do the following steps in parallel for all the processors p_j and $p_j \varepsilon p_f$: if *copy* (p_j , p_f) $\neq \emptyset$ then replace the schedule of p_j with the set task(p_j) U *copy* (p_j , p_f) ; if *copy* (p_j , p_f) =$\emptyset$ then continue to schedule task(p_j) U *active* (p_j) on p_j .

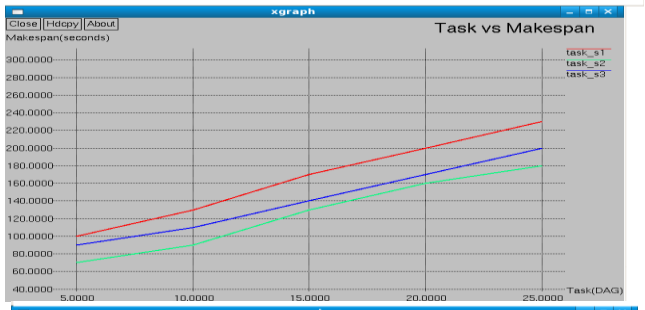
VII. PERFORMANCE EVALUATION

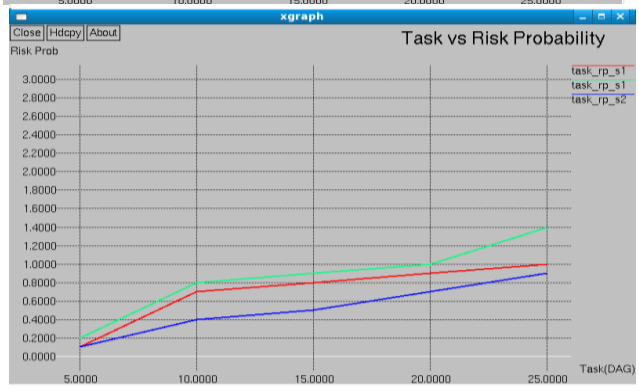
In this section, we compare the performance of SDS and SDFT algorithm in HDSs.To test the performance of these algorithms; we have built an extensive simulation environment such as heterogeneous processors in which the transmission rates of links are assumed to be 100 Mbits/second.

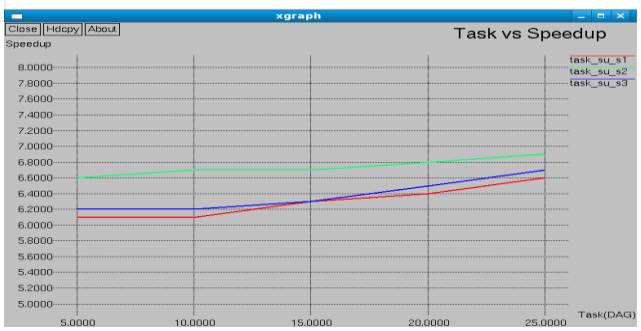
A. Experimental Results

The goal of the experiments is to compute the makespan, risk probability and speedup for SDS algorithm and then incorporate the fault tolerant into SDS algorithm and its performance metrics such as throughput and energy consumption was analysed.Figure. 3 show the simulation results for SDS algorithm that has been proposed above.

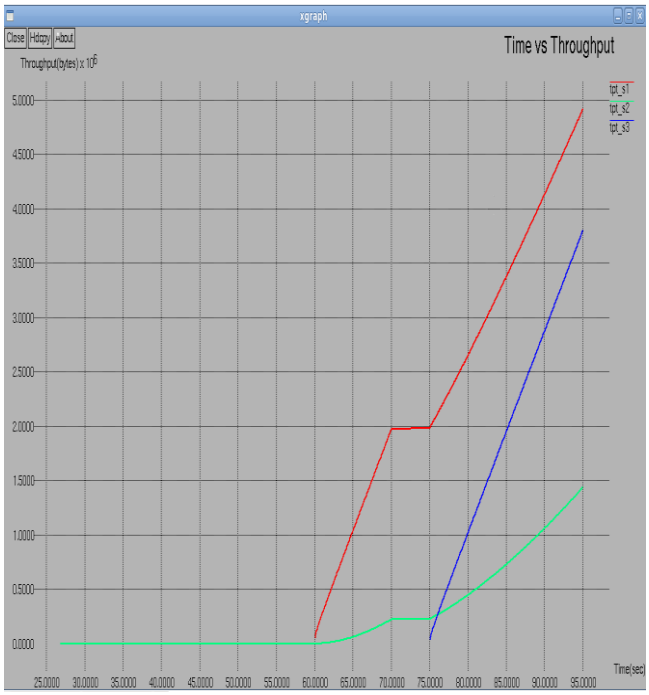
We observe from Figure. 3a shows that make span of SDS algorithms. Figure.3b plots the risk probability of the SDS algorithms. Another improvement with the speedup could be concluded from Figure. 3c.Thus, SDS algorithm is suitable for security-sensitive application with high computation demands.







Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)



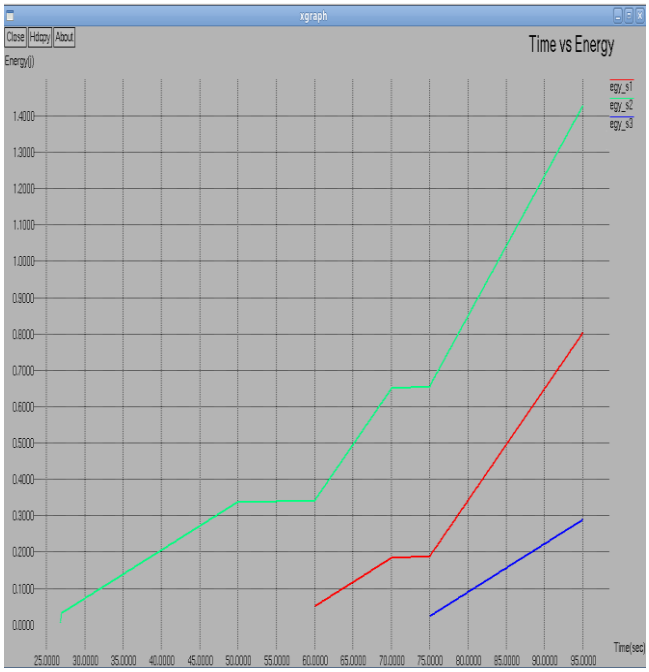
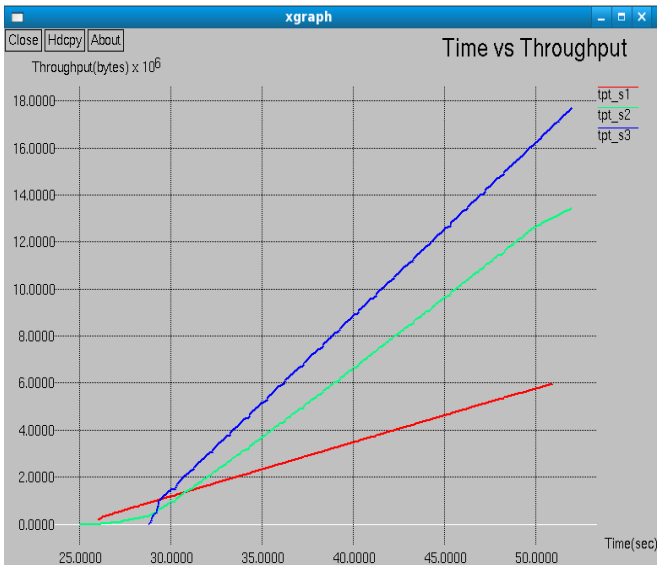


Figure 3. Performance impact of tasks of DAG for SDS algorithm (a) makespan in seconds; (b) risk probability; (c) speedup;(d)throughput;(e)energy consumption.



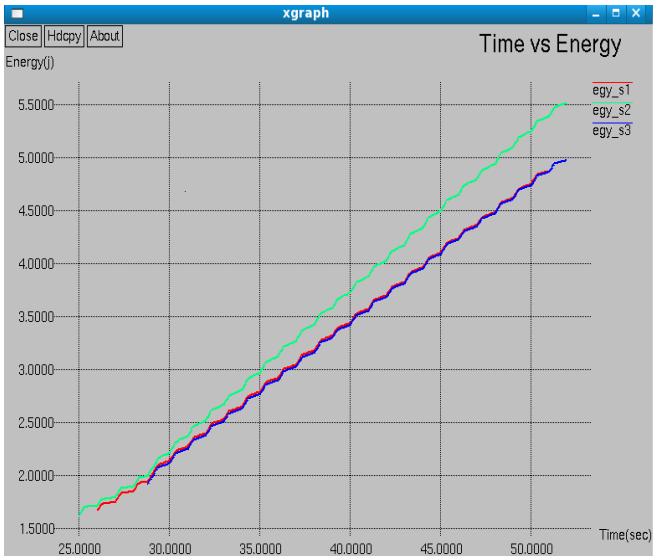


Figure. 4. Performance impact of SDFT algorithm (a) throughput ;(b)energy consumption .

VIII CONCLUSION AND FUTURE WORK

Without security-driven scheduler, the following two situations may occur. First, security-sensitive applications will run at a lower security levels, thereby leading to low quality of security. Second, security-sensitive applications will be at a higher security levels with higher security overheads, which can result in poor performance. To solve this problem, we had built a security-driven scheduling architecture and its performance was analysed in terms of scheduling length (makespan), risk probability, speedup, throughput and energy consumption. Then incorporate, the fault tolerant into SDS and its performance was analysed in terms of throughput and energy consumption.

Future studies in this domain are twofold: first it will be interesting to extend our security overhead models to multidimensional computing resources, such as network bandwidth, memory, and storage; second, it will be interesting to study the problem context under an arbitrarily interconnected HDS with security requirements.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

ACKNOWLEDGEMENT

Thanks go to the reviewers for the very detailed reviews and helpful suggestions.

REFERENCES

1. Bharadwaj Veeravalli, Kenli.Li, Xiaoyong Tang, and Zeng Zeng, (2011),"A Novel Security-Driven Scheduling Algorithm for Precedence-Constrained Tasks in Heterogeneous Distributed Systems", IEEE Trans.on Computers,Vol. 60, No. 7.
2. Xia Ping, and Zhou Xingshe, "Security-Driven Fault Tolerant Scheduling Algorithm for High Dependable Distributed Real-Time System"Fourth International Symposium on Parallel Architectures, Algorithms and Programming,2011.
3. J. Sarangapani, Wireless Ad Hoc and Sensor Networks: Protocols,Performance, and Control. CRC Press, Apr. 2007.
4. T. Xie, X. Qin, A. Sung, M. Lin, and L.T. Yang, "Real-Time Scheduling with Quality of Security Constraints," Int'l J. High Performance Computing and Networking, vol. 4, nos. 3/4, pp. 188-197, 2006.
5. G. Donoho, "Building a Web Service to Provide Real-Time Stock Quotes," MCAD.Net, Feb. 2004.
6. 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.
7. Alan A.B, Luigi V.M., and Federico R., "Fault-Tolerant Rate-Monotonic First-Fit Scheduling in Hard-Real-Time Systems", IEEE Transactions on Parallel and Distributed Systems, Vol.10, No.9, 1999, pp.934-945.
8. M.H. Klein, J.P. Lehoczky, and R.Rajkumar, "Rate- Monotonic Analysis for Real-Time Industrial Computing", Computer, pp.24-33, Jan.1994.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)