Agent Survivability: An Application for Strong and Weak Chain Constrained Scheduling

Axel W. Krings
Computer Science Department
University of Idaho
Moscow, ID 83844–1010
krings@cs.uidaho.edu

Abstract

In a recent two-layer approach to survivability of networked computing systems migratory autonomous agents have been used as a reactionary mechanism augmenting a low-level attack recognition scheme. This paper investigates the theoretical implications of determining the agent traversal route as specified by such survivability architecture. It introduces a five-step model that transforms survivability applications into a parameterized graph model that, together with model abstraction and representations, can be the basis for solutions derived from scheduling algorithms. The derivation of agent traversal paths will be transformed into the problem of scheduling jobs with linear precedence order in machine scheduling utilizing weak and strong modes of scheduling.

1. Introduction

Cyber attacks on computers and networks have reached epidemic proportions. Although much research has addressed the issue of increasing security of networked computer systems, problems and malicious acts are on the rise, rather than getting less [4]. One common approach to increasing the level of resistance to attacks is to minimize the functionalities accessible from the outside. Examples range from disabling macros within word processors to disabling scripting, e.g. JavaScript. However, the general philosophy of reducing accessible functionalities limits the effectiveness of technologies that rely on giving access to local functionalities through external mechanisms. Typical examples of such mechanisms are migratory autonomous agents.

Dealing with malicious act in cyber space has been the topic of much research in the security, survivability and fault-tolerance community. Security is often viewed as addressing issues of confidentiality, integrity, availability, as well as accountability and correctness. Survivability, on the other hand, goes beyond security and has been formulated with respect to *Resistance* to, *Recognition* of, and *Recovery* from attacks, with a final iteration considering *Adaptation* [9]. Whereas resistance and recognition are typically associated with security, the main consideration of survivability is recovery. Recovery is one of the prime objectives of much research in the area of fault-tolerance. However, malicious human act has traditionally not been the concern in the field. Malicious faults have been addressed in the context of fault models considering hardware that may fail in a pathological scenario, but such malicious faults have traditionally been considered orders of magnitude less likely than benign or symmetric faults [19]. Aspects of security and fault-tolerance can be implicitly seen in the common definition stating that "survivability is the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents" [9].

The lack of success in securing networked computer systems may be attributable to the missing theoretical groundwork and mathematical models [10]. Most approaches to security and survivability are ad hoc. Thus, in the absence of standardized security test procedures, claims can in general not be verified. Furthermore, it is not possible to compare relative results, as such comparisons would require a general common basis.

In an attempt to increase rigor in certain critical cyber problems, we are investigating transformation of security and survivability problems to other disciplines. Problem transformations in order to solve hard problems have been used extensively in mathematics and engineering. Well known examples include exponentiation or Laplace transformation. The general strategy is to transform the original problem into a different problem space in which known solutions exist, or solutions

can be found at lesser cost. After a solution has been derived in the new problem space, a reverse transformation is used to translate the solution found back to the original problem space.

This approach will be taken in this paper where problem transformation will be applied to agent-based systems that, due to their very nature, have been the source of many security concerns. Security problems compound in migratory agent systems where malicious agents could impact computer survivability and malicious computers could compromise agent security and survivability. The concept of resistance, recognition and recovery can therefore be seen from a computer system's or agent's point of view. This paper considers the first case, i.e. we consider survivability aspect due to potential malicious agents. The examples discussed focus on applications where response to malicious act may have high real-time sensitivity. It is desired to perform survivability actions as timely as possible.

This research is motivated by an agent based approach to survivability introduced in [14], where agents were the principal means of reacting to network based attacks. Specifically, migratory agents were the response mechanism of low-level real-time attack recognition at the kernel level. However, in [14] it was assumed that the agents responding to attacks were non-malicious.

In order to address security concerns and system survivability we assume that agent systems utilize redundancy. Redundancy in agent system is not a new concept and has been the focus of recent research [12, 13, 16, 17, 18]. The approaches discussed in this research focus on secret sharing [20, 17] rather than pure spatial or information redundancy. However, it should be noted that secret sharing using k shares can be easily extended to achieve survivability by extending it to a *m-of-k* scheme, in which the information of m out of k uncorrupted agents is needed to perform an action [17].

The research presented below uses the transformation model introduced in [15] to formalize aspects of agent survivability. Section 2 gives a model overview, describing the transformation steps. Section 3 discusses the scheduling model that will facilitate the solution space of the model. Several examples of scheduling models representing agent related applications are shown. Section 4 presents a case study in which a transformation from an agent survivability problem to the strong and weak chain constraint scheduling problem is motivated. Finally, Section 5 concludes the paper.

2. Model Overview

The survivability application, i.e. agent survivability, will be addressed utilizing the five-stage transformation model introduced in [15]. Before addressing agent survivability, the basic philosophy of the model, shown in Figure 1, will be explained using a simple example of a general computer network.

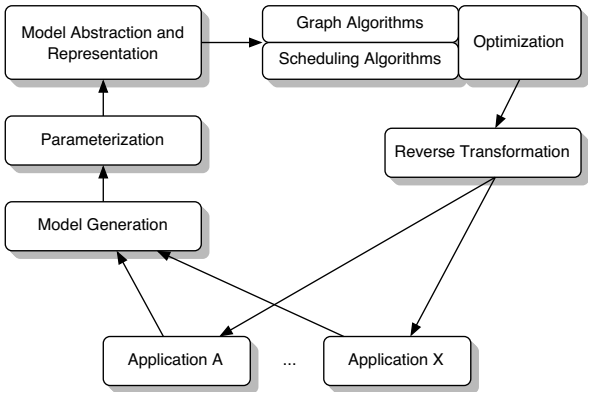

Figure 1: Model Overview

2.1. Application

Assume the application is a general computer network. One can envision the network as a collection of local area and wide area networks, consisting of typical components like workstations, switches, routers, bridges or gateways. The components themselves are connected via diverse technologies and protocols.

2.2. Model Generation

The application, i.e. the network, is transformed into a task graph together with the task model specification, if applicable. The general model is based on a directed graph $G = (V, E)$, where V is a finite set of vertices v_i and E is a set of edges e_{ij}, $i \neq j$, representing precedence relations between $v_i, v_j \in V$. In our example vertices could be the network components and edges the interconnections.

2.3. Parameterization

Now that the network is mapped to vertices and edges of G, a mapping of application specific parameters to generic parameters is needed. Examples of parameters are quality of service (QoS) parameters such as network throughput, propagation delay, communication cost, or priority. The vertices and/or edges of the

graph generated need to be assigned weights representing their characteristics. The results can be generalized by integer or real valued weights. Thus, for each vertex in V and edge in E, vertex and edge weights are defined respectively. Let w_i^v denote the vertex weight of v_i. Furthermore, let w_{ij}^e denote the weight of edge e_{ij}, where $v_i, v_j \in V$ and $i \neq j$.

2.4. Model Abstraction and Presentation

The weighted graph G can now be considered in the context of graph or scheduling problems. A graph theoretical formulation could follow directly from G together with a manipulative objective, e.g. finding the least cost path between two vertices. A scheduling theoretical formulation requires the specification of the scheduling model S, specifying the task and processing environment as well as the optimization criteria. The scheduling model will be described in detail in Subsection 3.

2.5. Algorithmic Solutions

Since a transformation to a scheduling problem is the focus of this paper, graph problems will not be discussed in the remainder of this paper. The scheduling model S is now subjected to scheduling theoretical algorithms. The goal is to find optimal or suboptimal solutions for the specific survivability criteria, applying the best suitable algorithm(s). A wealth of algorithms and heuristics of varying space and time complexity exist. Appropriate algorithms need to be identified that suit the optimization criteria. In addition, useful information about the time or space complexity may be inherited from the algorithms. This may provide valuable information about the solution space. After the application of the scheduling algorithms or heuristics, optimal or sub-optimal solutions will be available.

It should be pointed out that the focus of this research is not on scheduling algorithms, but on the identification of an appropriate scheduling model. Once the model is specified, a literature review for potential algorithms is required. However, given the semi-standardized formulations of scheduling models, this process can be efficient.

2.6. Reverse Transformation

The solutions of the graph or scheduling algorithms must now be translated back to the application. This requires a reverse transformation analogous to the transformation used in the Model Generation. This step represents the transformation from the solution space back to the application space.

3. Scheduling Models

The key to the model transformation is an understanding of how to derive an appropriate scheduling model S and its interpretation in the context of the specific application. In order to avoid lengthy descriptions of scheduling models S, a compact classification description of the form $S = (\alpha|\beta|\gamma)$ is commonly used [3]. The fields α, β, and γ indicate the processor environment, the task and resource characteristics, and the optimization criteria respectively. Each field will be briefly introduced, together with some standard notation indicated in parenthesis. Whereas the notation might seem non-intuitive to researchers outside the field of scheduling theory, it is semi-standard and well known in the scheduling community. For general information on the notation, the interested reader is referred to [3].

3.1. Processors

The first field of S, i.e. α, specifies the assumptions placed on the processor environment. There are many different attributes associated with processors [3]. For example, processors may be identical, uniform, unrelated or dedicated. *Identical* processors (P) refer to a homogeneous processor environment. *Uniform* processors (Q) assume different speeds b_i for each processor Q_i. In classical scheduling theory, uniformity implies that processor speeds differ but individual processor speeds are considered constant over time. If the speed is dependent on the task performed, processors are called *unrelated* [3]. In the transformation model unrelated processors, denoted by (R), can be a convenient model to describe computers subjected to Denial of Service (DoS) attack, which may be distributed (DDoS). Lastly, *dedicated* processors are assumed to be specialized for the execution of certain tasks and are described in *open shop*, *flow shop* and *job shop* systems.

3.2. Tasks

The second field of S, i.e. β, specifies the task and resource characteristics. Tasks may be non-preemptive (ϕ) or preemptive ($pmtn$). Once a non-preemptive task is started, its execution can *not* be interrupted. If preemption is allowed, task execution can be interrupted. If the tasks are subjected to precedence constraints, the specific constraints are indicated, e.g. (ϕ) indicates independent tasks, ($chain$) indicates task chains.

3.3. Optimization

The last field of S, i.e. γ, indicates the optimization criteria. Optimization is often defined based on task completions times, flow times, lateness, tardiness or earliness. The completion time C_j of the last task T_j of the schedule is called the *makespan*. Thus, the makespan indicates the overall length of the schedule. Makespan optimization is denoted by (C_{max}).

Another interesting criteria is the *flow time*, which is the sum of waiting time and processing time of a task. Alternatively, flow time is the completion time minus the release time, i.e. $F_j = C_j - r_j$. *Mean flow time* optimization is expressed by

$$\bar{F} = \frac{1}{n}\sum_{j=1}^{n} F_j$$

and is denoted by (ΣC_j) in the model notation. Variations of this criteria include optimization of *weighted mean flow time* denoted by ($\Sigma w_j C_j$).

Often the execution of a task T_j is under consideration of *due date* d_j, which specifies the time by which T_j should be completed. The consequences of failure are usually measured using a penalty function. Whereas a due date is considered a soft deadline, a task *deadline*, denoted by $\tilde{d}_j$, constitutes a hard deadline. Missing the deadline may render the application useless. The terms lateness, tardiness and earliness capture issues of due dates. Lateness is defined as

$$L_j = C_j - d_j.$$

However, if a task finishes before its due date, negative values for L_j arise. This is avoided in the definition of tardiness

$$D_j = max\{C_j - d_j, 0\},$$

and earliness

$$E_j = max\{d_j - C_j, 0\}.$$

Recall that the notation in parenthesis does not represent formulas but a notation within the scheduling classification model. A common optimization criteria is (L_{max}) defined as the maximum task lateness, i.e. $max\{L_j\}$, or *mean weighted tardiness* ($\Sigma w_j D_j$), which is defined as

$$\bar{D}_w = \frac{\sum_{j=1}^{n} w_j D_j}{\sum_{j=1}^{n} w_j}.$$

3.4. Transformations

In order to further the understanding of the transformations, several simple examples of scheduling models representing agent related applications are given below.

1. Problem $1|r_i|C_{max}$
   In systems facilitating autonomous agents to perform security related operations, e.g. version tracking or diagnostics, the agent might have a set of critical tasks to perform on specific systems [14]. This can be transformed to the scheduling problem $1|r_i|C_{max}$, which indicates a single processor ($\alpha = 1$) representing the agent. The computers that the agent is traversing is represented in S by tasks T_i with release times r_i ($\beta = r_i$). Note that in this example the meaning of "processor" and "task" is reversed from the application to the transformation into S. In the application, the agent, i.e. the software task, visits the processors, e.g. the computer. In the corresponding transformation, the tasks of S represent the computers visited by the agent, and the processor is representing the agent. The optimization criteria ($\gamma = C_{max}$) indicates that an agent traversal path is sought after that minimizes the overall time to perform the tasks.

2. Problem $Pm|d_i|\sum w_i D_i$
   Whereas the previous problem considered a single software agent, this problem relates to an application where a group of agents are tasked to counter the affects of a wide spread attack. The m identical agents are represented in S by processor ($\alpha = Pm$), and the computer systems to be traversed are the tasks. Deadlines d_i are considered for each task indicating time sensitivity. The optimization criteria is $\sum w_i D_i$, which is the weighted tardiness $D_i = max\{C_i - d_i, 0\}$, attempting to patch the maximal number of systems before they are victimized. Recall that C_i denotes the task completion time.

3. Problems $R|r_i|C_{max}$ or $Rm|d_i|\sum w_i D_i$.
   The previous two cases can be viewed in the context of DoS or DDoS attacks, which usually target resources, e.g. CPU, network bandwidth, or connectivity. For example, consider a DDoS attack that floods a transaction server of an enterprise with bogus requests. The transaction server, unable to distinguish between legitimate and unlegitimate requests, experiences high resource utilization. To the legitimate user it appears that the system slows down. This can be modeled using unrelated processors (R) in S. Recall that the speed of unrelated processors is dependent on the task performed. Fast response to the DoS is crucial when implementing survivability measures using

agents [14], as the defensive measures race in time against the increasing affects of the attack.

4. Problems $1|r_i, p_i|C_{max}$ or $Pm|r_i, p_i|\sum w_i C_i$. If agents are used for patch management, installing patches has to be coordinated with the usage of the system in order to avoid conflicts with programs executing on the system. This is very obvious in operating systems that require programs to be terminated before installing the patches or rebooting after installation. This kind of agent application can be transformed to scheduling problems such as $1|r_i, p_i|C_{max}$ or $Pm|r_i, p_i|\sum w_i C_i$. Note that release time r_i indicates the time after which it is feasible to install the patch and p_i indicates the processing time, i.e. the patch installation time.

4. Agent Survivability

Section 3 presented some agent related applications and suggested scheduling models that could be applied in the transformation model of Figure 1. This section addresses a multi-agent application in which agent redundancy is the assumed survivability mechanism [12, 13, 16, 17, 18]. Specifically, we assume that secret sharing requires that a certain number of agents need to be present at a processor in order to fulfill the security and survivability requirement. With respect to security, confidentiality may be achieved using simple secret sharing. In order to achieve survivability secret sharing may be extended to a *m-of-k* scheme, in which the information of m out of k uncorrupted agents is needed.

4.1. Application

Assume an application uses a multi-agent system to implement survivability functionalities on a set of networked computer systems. The relative importance of each computer to the organization or overall mission is prioritized, e.g. indicated by an integer numbered priority index scheme. Next, assume that a secret sharing scheme is implemented requiring k_i shares for a specific processing element, i.e. computer, PE_i. Thus, the multi-agent systems requires that all k_i agents must be present at computer PE_i to perform its specific function. Such a system could facilitate an agent based survivability approach analogous to the (p, m, k) system for survivable storage [20], where data is encoded into k shares, any m shares can reconstruct the data, and less than p shares reveal no information about the encoded data. Given this secret sharing agent survivability scheme, it is important to find efficient agent traversal paths, e.g. paths that maximize the efficiency of an agent system charged with reactionary response to malicious act.

4.2. Strong-Weak Scheduling

The application above will be transformed to a scheduling problem considering strong and weak chain precedence. Before stating the transformation we want to introduce strong-weak chain constrained scheduling, partially restating the notation and formulation of [8]. Assume there are K job chains $C = \{C_k\}_{k=1}^{K}$, where $C_k = \{J_{k,i}\}_{i=1}^{m_k}$, and the jobs in each chain C_k are linearly ordered by precedence relation $<$. Thus in each chain

$$J_{k,1} < J_{k,2} < ... < J_{k,m_k}$$

with

$$\sum_{k=1}^{K} m_k = n.$$

Each job $J_{k,i}$ has processing time $p_{k,i}$, release time $r_{k,i}$ indicating the time the job becomes ready for execution, due date $d_{k,i}$ specifying its deadline, and weight $w_{k,i}$. The K chains are to be scheduled on m parallel processors $P_1, ..., P_m$ to optimize a specified objective function. It is assumed that each processor can handle at most one job at a time.

The job chains can be scheduled according to two modes, strong and weak chain mode. Given any two jobs J_a and J_b, *strong* chain mode implies that if J_a immediately precedes J_b in a chain, then no other job can be scheduled between the completion of J_a and the starting of J_b. Thus, this mode strictly enforces the exclusive scheduling of the entire chain once the first job in the chain is selected. In *weak* chain mode the previous scheduling condition is allowed to be violated, as long as the schedule still conforms to the $<$ relation. This means that other jobs may be scheduled between the jobs of a chain, as long as the chain precedence still holds.

4.3. Scheduling Problem

The multi-agent application of Subsection 4.1 utilizing secret sharing with up to m shares will be transformed to the problem denoted as

$$Pm|fix_j, chain, p_j = 1|C_{max}$$

from [8]. Similar to [8], we will omit the chain subscripts for jobs in the discussion below to simplify the notation.

For ease of discussion assume that $m = 3$, resulting in

$$P3|fix_j, chain, p_j = 1|C_{max}.$$

This formulation reflects a system where three agents, each carrying a share of the secret sharing scheme, are traversing a set of computers to perform tasks that take the same amount of time. In the formulation of a scheduling problem this translates to a system that assumes 3 identical processors $(P3)$, representing the agents. Each job J_j, representing a computer, occupies a given set of processors at any moment of its processing. This set is specified by fix_j. If the number of shares in the secret sharing scheme is always 3, then $fix_j = 3$ for all J_j. If the degree of secret sharing is dependent on the computer that the agents visit, represented by J_j in the model, then fix_j represents this degree. It is assumed in the problem statement that the precedence constraint is a chain, reflecting the priority of jobs. Since each task performed by the agent takes the same amount of time, unit execution times for each J_j can be used, i.e. $p_i = 1$.

4.4. Problem Transformation

4.4.1 Model Generation:

The application of Subsection 4.1 is translated into a task graph $G = (V, E)$ consisting of K job chains C_k, each representing a group of computers. Each computer to be traversed by the agents is shown as a vertex, i.e. job, $J_{k,i} \in C_k$, for $1 \leq k \leq K$. The position of $J_{k,i}$ in C_k, i.e. the chain position, indicates the relative computer priority. Thus, if $Pr(PE_i)$ and $Pr(PE_j)$ denotes the priority of PE_i and PE_j respectively with priority values in the range $1 \leq i, j \leq m_k$, then edge $e_{i,j}$ is introduced in E if $Pr(PE_i) = Pr(PE_j) - 1$. Edge $e_{i,j}$ represents job precedence $J_{k,i} < J_{k,j}$.

4.4.2 Parameterization:

The parameters reflecting the functionality executed by the agents during job $J_{k,i}$ need to be specified. Parameterization follows directly from the definitions associated with jobs in Subsection 4.2. Hence, the job priorities are implicitly defined by the position of $J_{k,i}$ in C_k, and the processing and release times of $J_{k,i}$ are determined by $p_{k,i}$ and $r_{k,i}$ respectively. Potential cost or liabilities due to malicious act can be modeled by weight $w_{k,i}$.

4.4.3 Model Abstraction:

One possible scheduling model is [8]

$$Pm|fix_j, chain, p_j = 1|C_{max},$$

where m is the number of machines, represented by agents, in the system. Let J denote a partition of all jobs $J_{k,j}$. Furthermore, let J^{expr} denote the set of jobs that require all machines with indices indicated in $expr$ for execution. Now jobs can be partitioned by their resource requirements.

If, for example, one assumes $m = 3$, then jobs can be partitioned as

$$J = \{J^{123}, J^{12}, J^{13}, J^{23}, J^{1}, J^{2}, J^{3}\}. \quad (1)$$

The superscript indicates the machines required, e.g. J^{123} comprises all jobs that need machines M_1, M_2 and M_3 to execute, whereas jobs in J^{12} only require M_1 and M_2. The number of indices listed in the superscripts indicate the degree of secret sharing required by each of the jobs of the associated set. For example, each job in J^{123} requires that three agents need to present before their respective functionality can be executed. Similarly, each job in J^{12}, J^{13}, and J^{23} require two agents, however, each set has its specific set of associated agents. The problem

$$P3|fix_j, chain, p_j = 1|C_{max}$$

captures the notion of scheduling the tasks in J.

It should be pointed out that the problem formulation is extremely flexible. Whereas secret sharing is usually assume constant for a given application, this problem abstraction allows for the parameters of the secret sharing scheme to be specified for each individual computer.

4.4.4 Algorithmic Solution:

Solutions for several scheduling problems involving chains exist. Next solutions for weak and strong constrained scheduling will be discussed.

If all jobs of a chain k belong to the same partition in equation (1) then they obey the so-called agreeable machine configuration. This also implies that the degree of secret sharing of the entire chain is the same. The problem

$$Pm|fix_j, weakchain, p_j = 1|C_{max} \quad (2)$$

with agreeable machine configurations can be solved in $O(n)$ time [8]. They also provide a way to obtain C_{max}.

On the other hand, the problem in (2) without the agreeable machine configurations is NP-hard in the

strong sense for $m > 1$. A reduction from the 3-PARTITION problem is given in [8].

For strong chain constraints, the problem

$$P3|fix_j, strongchain, p_j = 1|C_{max}$$

has been shown to be NP-hard in the strong sense, even with agreeable machine configurations [1, 2]. Table 1 gives a summary scheduling problems and their complexities. Note that *amc* in the table indicates that the problem assumes the agreeable machine configuration.

If the requirement for secret sharing is dropped, a wider range of research results exist. With the elimination of secret sharing, i.e. fix_j is omitted in the scheduling problem formulation, this problem eliminates redundant agent traversal paths. Table 2, modified from [8], gives a overview of diverse problems and indicates the respective complexities. For more detail about the individual scheduling problems the reader is referred to the references identified. Note that b in the second problem of the table is the maximum number of jobs in a chain. The scheduling problems are sorted by the optimization criteria. The optimization criteria of the different problems identified are makespan C_{max}, mean flow time ΣC_j, and maximal lateness L_{max}.

4.4.5 Reverse Transformation:

The schedule produced by any scheduling algorithm directly reflects the agent traversal path leading to optimal behavior as defined by the optimization criteria of the scheduling algorithm.

5. Conclusion

This paper presented a transformation to determine agent paths in multi-agent systems using secret sharing. In an attempt to increase rigor in an area that traditionally has been criticized for its lack of formalism, a transformation model was introduced and applied to transform the agent problem into a scheduling problem.

The specific agent problem transformed assumed secret sharing as an approach to increasing agent security. Thus, a certain number of agents, each carrying a share of the secret sharing scheme, have to be present at a computer in order to execute the agent's functionality. Whereas using secret sharing to increase agent security is not new, the transformation to a scheduling problem is. By transforming the agent security problem to strong and weak chain constrained scheduling, new insight and a deeper understanding of the solution space with respect to run-time complexities of determining agent traversal paths has been achieved.

We hope that through this contribution researchers from the areas of security and survivability will realize that many problems can be mapped to well established research areas facilitating a wealth of inside and potential solutions.

References

[1] J. Blazewicz, P.W. Dell'Olmo, M. Drozdowski, and M.G. Speranza, "Scheduling Multiprocessor Tasks on Three Dedicated Processors", *Information Processing Letters* 41, pp. 275-280, 1992.

[2] J. Blazewicz, J., P.W. Dell'Olmo, M. Drozdowski, and M.G. Speranza, "Scheduling Multiprocessor Tasks on Three Dedicated Processors: Corrigendum", *Information Processing Letters* 49, pp. 269-270, 1994.

[3] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, "Scheduling Computer and Manufacturing Processes", Springer-Verlag, 1996.

[4] CERT/CC Statistics 1988-2002, CERT Coordination Center, http://www.cert.org/stats/cert_stats.html.

[5] E.G. Jr. Coffman, and R.L. Graham, "Optimal Scheduling for Two Processors", *Acta Informatica*, 1(3), pp. 200-213, 1972.

[6] M. Dror, J.Y-T., Leung, and C-H. Lin, "Uniform and Identical Machines with Unit Jobs and Chain Precedence", MIS Department, University of Arizona, 1996.

[7] M. Dror, Kubiak, W., and Dell'Olmo, P., " Scheduling chains to minimize mean flow time" *Information Processing Letters*, Vol. 61, 1997, pp. 297-301.

[8] M. Dror, Kubiak, W., and Dell'Olmo, P., " 'Strong' - 'Weak' chain constrained scheduling," *Ricerca Operativa*, Vol. 27, 1998, pp. 35-49.

[9] E. Ellison, L. Linger, and M. Longstaff, *Survivable Network Systems: An Emerging Discipline*, Carnegie Mellon, SEI, Technical Report CMU/SEI-97-TR-013, 1997.

[10] Keynote Speech of the Information Survivability Workshop, part of the International Conference on Dependable Systems and Networks, DSN-2001, by Roy Maxion, CMU, Goteborg, Sweden, 2001.

Scheduling Problem	Complexity
$Pm\|fix_j, weak\ chain, p_j = 1\|C_{max}$	with *amc* $O(n)$ [8]
$Pm\|fix_k, strong\ chain, p_j = 1\|C_{max}, \ \ m > 1$	NP-hard in the strong sense [8]
$P3\|fix_j, weak\ chain, p_j = 1\|C_{max}$	with *amc* NP-hard in the strong sense [1, 2]

Table 1: Summary of Scheduling Problems (implying secret sharing)

Scheduling Problem	Complexity
$P\|weak\ chain, p_j = 1\|C_{max}$	$O(K)$ [8]
$Pm\|strong\ chain, p_j = 1\|C_{max}$	$O(mnb^m)$ [8]
$P\|strong\ chain, p_j = 1\|C_{max}$	NP-hard in the strong sense [8]
$P2\|strong\ chain\|\Sigma C_j$	$O(nK)$ [6]
$P2\|weak\ chain\|\Sigma C_j$	NP-hard in the strong sense [6]
$P\|weak\ chain, p_i = 1\|\Sigma C_j$	polynomial time [8]
$P\|strong\ chain\|\Sigma C_j$	NP-hard in the strong sense[8]
$P\|r_j, weak\ chain, p_j = 1\|L_{max}$	polynomial time[8]
$P\|r_j, strong\ chain, p_j = 1\|L_{max}$	NP-hard in the strong sense [8]
$P2\|weak\ chain, p_j = 1\|L_{max}$	$O(n^2)$ [8, 11]
$P2\|strong\ chain, p_j = 1\|L_{max}$	$O(nK)$ [8]
$P2\|weak\ chain, p_i = 1\|\Sigma C_j$	$O(n^2)$ [5]

Table 2: Summary of Scheduling Problems (not implying secret sharing)

[11] M.R. Garey, and D.S. Johnson, " Scheduling Tasks with Nonuniform Deadlines on Two Processors", *Journal of the ACM,* 23(3), pp. 416-426, 1976.

[12] D.Johansen, R.van Renesse, and F.B.Schneider, Operating System Support for Mobile Agents, *Proc. 5th IEEE Workshop on Hot Topics in Operating Systems*, 1995.

[13] D.Johansen, K.Marzullo, F.B.Schneider, K.Jacobsen, and D.Zagorodnov, NAP: Practical Fault-Tolerance for Itinerant Computations, Technical Report TR98-1716, Department of Computer Science, Cornell University, USA, November, 1998.

[14] Krings A.W, W.S. Harrison, et.al., "A Two-Layer Approach to Survivability of Networked Computing Systems", *Proc. International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, L'Aquila, Italy, Aug 06 - Aug 12, pp. 1-12, 2001.

[15] Krings A.W, and M.H. Azadmanesh, "A Graph Based Model for Survivability Analysis," Report Series UI-CS-TR-02-024, Computer Science Department, University of Idaho, August, 2002.

[16] K.Rothermel, and M.Strasser, A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents, *Proc. IEEE Symposium on Reliable Distributed Systems* (SRDS'98), West Lafayette, USA, October, 1998, pp. 100-108.

[17] F.B.Schneider, Towards Fault-tolerant and Secure Agentry, *Proc. of the 11th International Workshop on Distributed Algorithms* Saarbrucken, Germany. September, 1997.

[18] F.M.Assis Silva, A Transaction Model based on Mobile Agents, PhD Thesis, Technical University Berlin, 1999.

[19] P. Thambidurai, and You-Keun Park, Interactive Consistency with Multiple Failure Modes, *7th Reliable Distributed Systems Symposium*, Columbus, OH, pp. 93-100, October 1988.

[20] Jay J. Wylie, et.al., Selecting the Right Data Distribution Scheme for a Survivable Storage System, Technical Report, CMU-CS-01-120, Carnegie Mellon University, May 2001.