INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 12, NO. 1, MARCH 2007, 56-62

A Distributed Algorithm for Workflow Recovery

Wanyu Zang, Meng Yu, and Peng Liu,

***Abstract*— Even though the automatic recovery techniques of workflow have attracted enough attention in recent years, several critical issues regarding the distributed recovery have not been addressed. For example, if we do the recovery under sustained attacks, in which condition the recovery can terminate? Is a synchronized clock necessary for distributed recovery? In this paper, we proposed a dead-lock free attack recovery algorithm for coordinated recovery and answered related questions. We defined different IDS report orders, and discussed the termination of the recovery under the given IDS report orders. We also proved that under specific situations, we have to freeze the recovery scheme to guarantee that the recovery can make progress.**

***Index Terms*— Attack recovery, self-healing, workflow systems, transactional processes**

1. Introduction

Distributed transactional processing systems (e.g., distributed database systems and workflow systems) are important in most critical infrastructures such as financial services. These services rely on the correctness, availability, and reliability of the processing systems. Each transactional process consists of a set of transactions or tasks[1] that are related to each other in terms of the semantics of a business process. Each transaction represents a specific unit of work that the business needs to do (e.g., a specific application program, a database transaction). A consistent and reliable execution of distributed transactional processes is crucial for all organizations.

However, it is well known that system vulnerabilities cannot be totally eliminated, and such vulnerabilities can be exploited by attackers who penetrate the system. We believe in that at least four aspects should be considered while building defense mechanisms for a secure, reliable system: **protection, detection, response**, and **restore**. Current research pays more attention to the research of protection, detection, and response, e.g., access control of a secure system, intrusion detection of both systems or network, congestion control of network, than to the research of restore that recovers a system to normal service level.

The attack recovery is important because no defense, or detection system is perfect in the real world. Therefore, after the attacker successfully breaks into the system, we need to bring the system back to its original integrity level. Our work focuses on the recovery from successful attacks.

In this work, we focus on those intrusions that inject malicious transactions into a distributed transaction processing system instead of the attacks that only crash the system. These intrusions can happen in many situations, for example, when attackers access a system with stolen (guessed, cracked, etc.) passwords or when some defense mechanisms, such as access control, are broken by the attackers. Under such intrusions, transactions and data may be forged or corrupted. For one example, an attacker may forge bank transactions to steal money from accounts of others, thereby generating malicious transactions. For another example, the attacker may schedule a travel with forged credit card information that carries incorrect data in transactions.

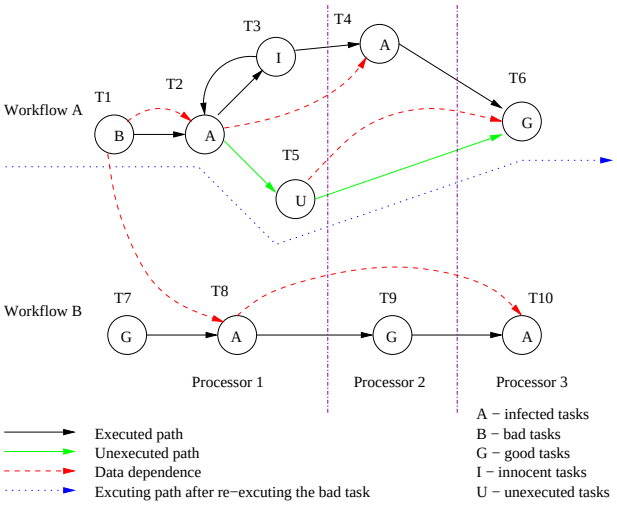


Fig. 1. An example of workflows

If these malicious transactions were not recovered in time, they would sooner or later spread misleading information or damage to a lot of legitimate transactions and other hosts, generating more trash data in the distributed system. To correct the situation, the malicious transactions must be removed from the system, and all affected transactions must be repaired.

However, the solution is not trivial. For example, the damage can spread not only among transactional processes, but also across hosts. The local view to the system is not enough for correct and complete recovery. The global knowledge of the distributed system is necessary to trace damage spreading and repair damage.

In Figure 1, two workflows are processed by three processors on three nodes (or sites). Branches in the figure are the choices of execution paths. P_1 : $t_1t_2t_3t_4t_6$ and P_2 : $t_1t_2t_5t_6$ are two possible execution paths. In each execution (or instance) of workflow A, only one path can be selected (based on the results of task t_2). In this example, P_1 is the execution path led by an attack, and P_2 is the normal execution path without corruption.

In the example, task t_1 marked with "B" is the only malicious task that is manipulated directly by the attacker and

Wanyu Zang and Meng Yu are with Monmouth University, NJ 07724, USA. Email: myu@monmouth.edu

Peng Liu is with Pennsylvania State University, University Park, PA 16802, USA. He was supported in part by NSF CCR-TC-0233324.

[1]We do not distinguish them in our paper.

is identified by the Intrusion Detection System (IDS) [23]. Due to reading corrupted data from task t_1, tasks t_2, t_4, t_8, and t_{10} calculate wrong results. They are marked with "A", indicating affected tasks. Furthermore, task t_2, based on the corrupted data it reads from t_1, makes a wrong decision to execute tasks on path P_1. In fact task t_3 and task t_4 would not have been executed at all if t_1 were not malicious.

From this example, we learn that the damages directly caused by the attacker may be spread by the execution of normal, legitimate tasks without being detected by the IDS. Even worse, the damage will spread among hosts, like $t1, t2$ are on node 1, and t_4 is on node 2. The local view to the system, either node 1 or node 2 is not enough for correct and complete recovery. The global knowledge of all the three nodes is necessary to trace damage spreading and repair damage.

Existing techniques cannot solve above problems. For example, intrusion detection (ID) techniques, which have received much attention, can only identify the malicious tasks that are manipulated directly by the attackers, like task t_1 in the example. For the affected tasks, like t_2, t_4, t_8, t_{10}, caused by referring damaged data or incorrect execution paths, ID considers them as legitimate tasks and will not create an alert.

Previous work [1], [28], [29], [32], [27], [31], [33], especially our recent work [49], [50], has built fundamental theories and a simple prototype system for attack recovery. Based on the dependency relations among the transactions, which can be obtained through the analysis of the system log or the applications, our system finds out all the damage and generates recovery transactions to repair the damage. However, some theoretical issues regarding the distributed system have not been addressed. For example, the termination and deadlock of recovery. We will address these problems in this paper.

2. Preliminaries

We have proposed two attack recovery models for transactional processes in previous work [51], [50]. However, these models ignored some important features in distributed systems for the theoretical simplicity.

2.1. Unrecoverable Transactions

In a distributed system, we need to consider both inside operations and interactions with the outside world. The formal models in our previous work failed to formalize interactions with the outside world. Since interactions with the outside world are not recoverable, we use an OWS (outside world site) to model the outside world. All transactions happened on an OWS are *unrecoverable*. We consider all inputs obtained from users and all outputs to the users happen on a user site S_u which is a OWS. All user's transactions are called OWTs (outside world transactions). For example, a transaction that a user withdraws money from a ATM cannot be recovered. It is an OWT and the ATM is an OWS.

2.2. Notation of Distributed Transactions

Existing attack recovery models also failed to describe site bindings and process qualifiers that will be defined in the following example. Therefore, they are unable to describe distributed transactional processes accurately. By introducing new notations through an example, we will build a dependency relation based model to describe all theoretical results in a formal way.

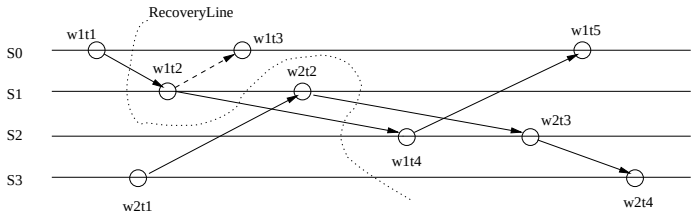


Fig. 2. Two transactional processes with forward domino effects

Figure 2 demonstrates several typical damage spreading modes. In the figure, each node (circle) is a transaction. Every transaction belongs to a transactional process, e.g., $w_1 : t_1$ belongs to process w_1 and $w_2 : t_1$ belongs to process w_2, where w_2 is a *process qualifier*. When the location where a transaction executes matters, we use a pair $(w_i : t_j, s_k)$, which is called a *site binding*, to represent transaction $w_1 : t_j$ executing on site s_k, e.g., $(w_1 : t_1, s_0)$ and $(w_2 : t_3, s_2)$ in the figure.

2.3. Dependency Relations

We define dependency relations in the following since damage spreads through dependency relations and executing orders of recovery transactions are determined by dependency relations. In Figure 2, the start node of an arrow will be executed right before the end node, which defines a *precedence* relation. For example, $w_1 : t_1$ *precedes* $w_1 : t_2$, which is denoted by $w_1 : t_1 \prec w_1 : t_2$. There are two execution paths in w_1. One is $w_1 : t_1t_2t_4t_5$. The other one is $w_1 : t_1t_2t_3$. $w_1 : t_2$ makes decision on which execution path will be selected. The selection between $w_1 : t_2$'s successor $w_1 : t_3$ or $w_1 : t_4$ is called *control dependence*, which is denoted by $w_1 : t_2 \rightarrow_c w_1 : t_3$ and $w_1 : t_2 \rightarrow_c w_1 : t_4$. In the figure, execution path $w_1 : t_1t_2t_4t_5$ was selected.

Assume $\prec$ is a relation on set $\mathcal{S}$ then we define minimal$(\mathcal{S}, \prec) = x$ where $x \in \mathcal{S} \wedge \nexists x' \in \mathcal{S}, x' \prec x$. Note there may be more than one result qualified by the definition of minimal$(\mathcal{S}, \prec)$. For example, $\mathcal{S} = \{t_i, t_j, t_k\}$, $t_i \prec t_k$ and $t_j \prec t_k$, then both t_i and t_j are qualified results for minimal$(\mathcal{S}, \prec)$. In cases like this, we randomly select one qualified result as the value of minimal$(\mathcal{S}, \prec)$. The task scheduler is supposed to choose minimal$(\mathcal{S}, \prec)$ to execute.

Once $w_1 : t_2$ is identified as a transaction compromised by the attacker, and the selection of execution path $w_1 : t_1t_2t_4t_5$ was wrong, the execution of $w_1 : t_4$ and $w_1 : t_5$ need to be recovered. Furthermore, if $w_2 : t_3$ reads information generated by $w_1 : t_4$, which is *flow data dependence* denoted by $w_1 : t_4 \rightarrow_f w_2 : t_3$, the damage will be spread to $w_2 : t_3$. Note that even though no message has been sent from $w_1 : t_4$ to $w_2 : t_3$, they may still have data dependencies since they may share data objects on the same site. Similarly, if we have $w_2 : t_3 \rightarrow_f w_2 : t_4$ then $w_2 : t_4$ will be damaged as well.

In the above example, damage is spread through dependency relations. We can similarly define the other two types of data dependence. If t_j modifies data objects after t_i reads them, then t_j is *anti-flow dependent* on t_i, which is denoted by $t_i \rightarrow_a t_j$. If $t_i \prec t_j$, and they have common data objects to modify, then t_j is *output dependent* on t_i, which is denoted by $t_i \rightarrow_o t_j$.

All the relations $\rightarrow_f$, $\rightarrow_a$ and $\rightarrow_o$ are data dependency relations and are not transitive. From the well known results of concurrency and parallel computing, if t_j is data dependent on t_i, then they cannot run in parallel or concurrently, and $t_i \prec t_j$ must be satisfied. Otherwise, we will get wrong results.

2.4. Concurrency Restrictions in Recovery

We use a simple example to explain that there do exist such restrictions on executing orders of transactions in dependency relation based recovery.

Since the process qualifier and site binding do not change the results of the following discussion, we remove them in notations for the simplicity. Consider transactions $t_1 : a = 1, t_2 : b = 2$, and $t_3 : y = a + b$, which are executed in the sequence of $t_1 \prec t_2 \prec t_3$. We have $t_1 \rightarrow_f t_3 \wedge t_2 \rightarrow_f t_3$. Assume that t_2 has been identified as compromised by an IDS, so the value of b is corrupted. Therefore, t_3 is also corrupted since it reads a incorrect b. To recover, t_2 needs to be undone followed by redone. t_3 needs to be redone. We must satisfy the sequence of undo$(t_2) \prec$ redo$(t_2) \prec$ redo(t_3) in the recovery. Any other execution will get wrong results. The precedence relations introduced by dependency relations is called *concurrency restrictions*.

The concurrency restriction for transactions without process qualifiers and site bindings has been studied and formally described in [50], [51]. The concurrency restriction is also caused by dependency relations. However, we can break anti-flow dependency relations by introducing multi-version data, as described in [51].

2.5. Transactional Processes

With above notations, transactional processes can be modeled as $(T, S, \prec, \rightarrow_f, \rightarrow_a, \rightarrow_o, \rightarrow_c)$, where T is a set of transactions with process qualifiers, S is a set of sites that are corresponding to a host or a processor in the distributed system, $S_u \in S$ is an OWS, and all dependency relations among transactions. With the model, we can trace damage spreading in the distributed system, work out concurrency restrictions, and solve other important problems addressed in the rest of the paper.

3. Recovery Analysis

Recovery analysis consists of damage tracing and recovery scheme generation. Damage is spread through dependency relations, including both data dependencies and control dependencies. The damage can be spread intra-transactional processes, or inter-transactional processes. Damage tracing identifies all damaged transactions through dependency relations.

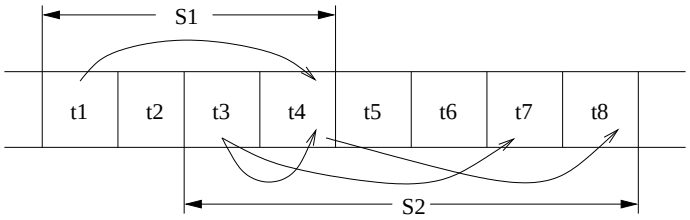


Fig. 3. Overlap of Two Segments

We consider $G\langle T, \rightarrow\rangle$ as a dependency graph. Since transactional processes can be modeled as a dependency graph, the procedure of damage tracing can be described as a procedure of chasing the dependency graph. Rules proposed in [50] describes how to chase the graph to identify all damaged transactions. A sink node which has 0 out degree will not affect any other nodes even if it is damaged.

Recovery scheme generation generates recovery transactions and execution orders according to the results of damage tracing. The recovery scheme can be modeled as $G\langle R, \prec\rangle$, where R is the set of recovery transactions, and $\prec$ is the precedence relations. The recovery scheme generation constructs the recovery graph increasingly.

3.1. Overlaps of Undetermined Segments

In a distributed system, message may arrive the destination not in the same order as they were sent. An IDS may identify damage not in the same order as the damage happened. These situations affect the progress of recovery analysis.

Definition 1: If an IDS reports an incidents sequence $i_1 i_2 \cdots i_n$, where for any i_j and i_k, $1 \leq j < k \leq n$, $i_j \prec i_k$ then the IDS reports incidents *in the temporal order*.

It is possible that IDSs do not report incidents according to the temporal order, then the situation in Figure 3 needs to be considered.

In the figure, the IDS, or a message from other sites, firstly reports t_3 as a damaged transaction, which leads to t_4, t_7, and t_8 identified as damaged transactions. All these transactions are in segment S_2. Transaction t_1 may be reported as damaged after the incident reporting t_3. According to the dependency relations denoted by curve arrows in the figure, segment S_1 will be re-scanned. There will be a overlap between S_1 and S_2, where $S_1 \cap S_2 \neq \phi$.

In the figure, since t_4 has been identified as damaged in S_1, it is not necessary to re-scan all transactions in S_1. We found the following truth regarding the local recovery analysis.

Theorem 1: A Local recovery analysis can terminate for a finite sequence of incidents no matter if the segment of incidents are reported in temporal orders or not.

Proof: For each incident, the length of the system log is limit since the number of transactions committed is limit. Therefore the scan of the system log can terminate. For a finite sequence of incidents, the scan of the system log can terminate because the possible overlaps do not affect the total number of committed transactions. ■

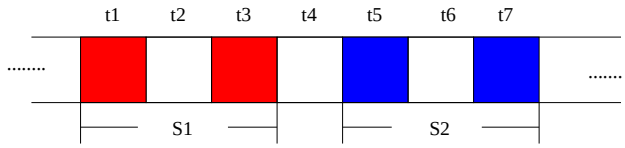


Fig. 4. An example of temporally segmented IDS reports

3.2. Termination of Recovery Analysis for Temporal Reports

If damage is reported in temporal order, we have the following results for the global recovery analysis.

Lemma 1: If all IDSs report incidents in the temporal order, the global recovery analysis of $minimal(T, \prec)$ is acyclic, and the transaction is identified by the IDS.

Proof: By contradiction. Assume that in the recovery analysis, $t \in minimal(T, \prec)$ in the distributed system has been visited twice. In other words, t is in a circle in the recovery analysis. Since the IDS reports incidents in the temporal order, the IDS will not report the incident including t twice. Therefore, the second visit to the transaction should be caused by damage spreading, say, the transaction is dependent on other transactions happened earlier, e.g., $t' \prec t$ which is contradictory to our assumption that $t \in minimal(T, \prec)$.

According to our discussion, since t is not affected by other transactions, t should be reported by the IDS. ■

Theorem 2: If all IDSs report incidents in the temporal order and there is no further incidents reported or the damage is contained, the global recovery analysis will terminate.

Proof: Since all IDSs report incidents in the temporal order, according to Lemma 1, the recovery analysis of $t \in minimal(T, \prec)$ will not be cyclic. Therefore, t can be recovered anyway. We can always choose $minimal(T, \prec)$ in the rest transactions. ■

3.3. Termination of Recovery Analysis for Temporally Segmented Reports

It is difficult to require an IDS to report all incidences according to the temporal order strictly. We have the following theorem to weaken the requirement to the IDS.

Definition 2: If an incidence sequence reported by the IDS can be segmented into several segments, $S_1, S_2, \cdots, S_n$, for any $i_j \in S_j$ and $i_k \in S_k$, $1 \leq j < k \leq n$, $i_j \prec i_k$, then the incidents reported by the IDS can be *temporally segmented*. The time span that a segment S_i covers is called the cycle α_i of S_i. The cycle of an incidence sequence is defined as $\alpha = \max(S_1, S_2, \ldots, S_n)$.

In Figure 4, the IDS report can be segmented into S_1 and S_2. All reports in S_1 are before those of S_2. However, in S_1, the IDS reports t_3 first, then t_1. In S_2, the IDS reports t_7 first, then t_5. According to the definition, the IDS reported in Figure 4 is temporally segmented, but not in the temporal order. If, in S_1, the IDS reports t_1 first, then t_3. In S_2, the IDS reports t_5 first, then t_7. Then we can say the IDS reported in Figure 4 is in the temporal order.

For task t, if there does not exist t', such that $t' \rightarrow t$, we say the recovery of t is *independent*. Therefore, for any damage spreading graph $G'\langle T', \prec\rangle$ containing t, $t \in minimal(T', \prec)$ if and only if t is independent.

Lemma 2: If all incident sequence generated by IDSs can be temporally segmented, the global recovery analysis of $minimal(T, \prec)$ is acyclic.

Proof: Since all incident sequence generated by IDSs can be temporally segmented, each of them can be denoted by $S_1, S_2, \cdots, S_n$, for any $i_j \in S_j$ and $i_k \in S_k$, $1 \leq j < k \leq n$, $i_j \prec i_k$. Assume that $e_k = minimal(T, \prec)$ and $e_k \in S_k$, then for any $e_j \in S_j$ and $e_k \in S_k$, $1 \leq j < k \leq n$, $e_j \prec e_k$.

According to the discussion of the overlap of undetermined segments, the recovery will not be pull back further than $e_k = minimal(T, \prec)$ in each segment.

$minimal(T, \prec)$ in the global recovery analysis will not be visited twice for the same reason in Lemma 1. ■

As long as the damage chasing speed is faster than the spreading speed, the recovery analysis can terminate.

Theorem 3: If 1)all incident sequence generated by IDSs can be temporally segmented, , 2)there is no further incidents reported, and 3)the damage is contained, then the global recovery analysis will terminate.

Proof: According to Lemma 2, the recovery analysis of $minimal(T, \prec)$ is acyclic, so its recovery analysis can be done. We can always choose $minimal(T, \prec)$ from the rest transactions. Therefore, the global recovery analysis can make progress, which reduces the total number of incidents to be analyzed. Since incidents reported has limit number and damage is contained, which means the total number of damaged transactions will not increase. Therefore, the global recovery analysis will terminate. ■

Assume there are n sites in the system and $M_i = minimal(T_i, \prec)$ for the ith site, then $M = M_1 \cap M_2 \cap \ldots \cap M_n$ is the *recovery line*. For any transaction $t \prec t'$, $t' \in M$, t does need recover. The recovery line indicates the "left" boundary of recovery. Any transactions on the left of the recovery line, in other words, happened earlier then the recovery line, does not need to be recovered.

4. Damage Containment and Repair

Definition 3: A *recovery graph* is a directed graph containing all recovery tasks as vertices and precedence relations as edges.

The procedure of repair is the traversal of the recovery graph.

If referring to the data in the recovery graph is not allowed, we say the damage is *contained*. If no further revision is allowed on the recovery graph, we say the recovery graph is *frozen*. Damage contain requires that all data items generated by damaged transactions not to be referred until these transactions are recovered.

When the damage chasing is done, the recovery graph can be frozen. The condition to freeze a recovery graph can be determined by several times longer then the length of IDS report segments. By this way, even though overlaps between recoveries may occur, we can guarantee the progress of recovery.

Theorem 4: If a transactional process is acyclic, then the repair procedure is dead-lock free.

Proof: Figure 5 shows the recovery graph (the right one) constructed from $a \prec b \wedge a \prec c$ (the left one). In the figure,

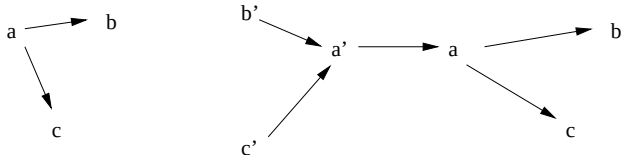


Fig. 5. The construction of a recovery graph

a', b', c' are undo transactions of a, b, and c. The recovery graph is acyclic.

According to our previous conclusions [50], [51] (theorems to generate precedence relations), all possible precedence relations between undo and redo transactions are displayed in the figure. Since the original transactional process is acyclic, the recovery graph will also be acyclic. Therefore, the repair procedure will be dead lock free. ■

Theorem 5: The repair procedure can make progress and terminate if 1)the damage is contained, 2)the recovery graph is frozen, and 3) all transactional processes are acyclic.

Proof: According to the definition of containment and freezing of the recovery graph, damage will not be spread by referring the recovery graph and the recovery line has been identified. Since we can choose to recover any transaction in the recovery line, the recovery line will move forward. In other words, the recovery can make progress. Since the recovery graph has been frozen. The number of recovery transactions is limit. Furthermore, the recovery is dead-lock free according to Theorem 4 since all transactional processes are acyclic. Therefore, the recovery can terminate. ■

5. The Integrated Recovery Algorithm

Based on the theory of recovery analysis and repair, we proposed an integrated recovery algorithm in this section. Our algorithm includes recovery analysis as the first part and damage repair as the second part.

We assume that the recovery components of the system, and all related information and communication are trustable. For example, we assume that all information is authenticated.

The following algorithm runs on each site for the attack recovery.

1) Wait σ seconds, where $\sigma < \alpha$, for the set of damaged transactions B_i reported by an IDS or the set B_s reported by other sites
2) Local recovery analysis generates $\langle R, \prec \rangle$ based on $B_s \cap B_i$;
3) Send all $s_j|w_k|t_l \in R$ as set B, where $j \neq i$, and $\prec$ to site s_j;
4) If $minimal(R \cap T, \prec)$ has been changed in α seconds, then go to step 1;
5) do the repair according to $G\langle R, \prec \rangle$
6) go to step 1;

In the algorithm, T contains all transactions submitted by users. We assume that recovery transactions in R consist of undo transactions and redo transactions. $R \cap T$ includes damaged transactions that need to be redone.

We assume that dependency relations are carried by transactional processes. There are two kinds of information being transferred among different sites during the recovery: 1) identified damage transactions, and 2) recovery transactions and executing orders.

In the algorithm, step 1 to step 4 are for recovery analysis, which generates recovery a recovery scheme denoted by $G\langle R, \prec \rangle$. Step 5 repairs the damage.

Our recovery algorithm does not need a global synchronized clock because the recovery, including the recovery analysis and the repair, depends only on the precedence relations among transactions. The precedence relations usually are integrated in the specification of transactional processes.

6. Related Work

Intrusion Detection. One critical step towards intrusion-tolerant systems is ID, which has attracted much attention over the past years [7], [34], [15], [13], [36], [39], [35], [20], [22]. The existing methodology of ID can be roughly classed as anomaly detection, which is based on profiles of normal behaviors [16], [40], [21], [41], and misuse detection, which is based on known patterns of attacks, called signatures [14], [11], [42]. However, existing anomaly detection techniques focuses on identifying attacks on OS and computer networks, which cannot be directly used to detect malicious transactions. Although there are some works on database ID [6], [44], these methods are neither application aware nor at transaction-level, which are the two major design requirements of the ID component of the attack recovery system.

An IDS [23] can detect some intrusions. But, in a distributed computing system, the damage directly caused by the attacker may be spread by executing normal transactions without being detected by the IDS. The IDS is unable to trace damage spreading and cannot locate all damage to the system.

Survivability and Intrusion Tolerance. The need for intrusion tolerance, or survivability, has been recognized by many researchers in such contexts as information warfare [12]. Recently, extensive research has been done in general principles of survivability [18], survivable software architectures [43], survivability of networks [38], survivable storage systems [47], [10], [45], [48], etc. These researches are helpful for database survivability, but the techniques cannot be directly applied to build intrusion tolerant database systems.

Some research has been done in database survivability. For instance, in [2], a fault tolerant approach is taken to survive database attacks where (a) several phases are suggested to be useful, such as attack detection, damage assessment and repair, and fault treatment, but no concrete mechanisms are proposed for these phases; (b) a color scheme for marking damage (and repair) and a notion of integrity suitable for partially damaged databases are used to develop a mechanism by which databases under attack could still be safely used. This scheme can be viewed as a special damage containment mechanism. However, it assumes that each data object has an (accurate) initial damage mark, our approach does not. In fact, our approach focuses on how to automatically mark (and contain) the damage, and how to deal with the negative impact of inaccurate damage marks. There are also some work on OS-level database survivability. In [37] a technique is proposed to detect storage jamming, malicious modification of data, using a set of special detect objects which are indistinguishable to the jammer from normal objects. Modification on detect

objects indicates a storage jamming attack. In [4], checksums are smartly used to detect data corruption. Similar to trusted database system technologies, both detect objects and checksums can be used to make our techniques more resistant to OS level attacks.

Fault Tolerance. The failure handling of workflow has been discussed in recent work [8], [5], [46]. Failure handling is different from attack recovery in two aspects. On one hand, they have different goals. Failure handling tries to guarantee the atomic of workflows. When failure happens, their work find out which transactions should be aborted. If all transactions are successfully executed, failure handling does nothing for the workflow. Attack recovery has different goals, which need to do nothing for failure transactions even if they are malicious because failure malicious transactions have no effects on the workflow system. Attack recovery focuses on malicious transactions that are successfully executed. It tries to remove all effects of such transactions. On the other hand, these two systems active at different times. Failure handling occurs when the workflows are in progress. When the IDS reports attacks, the malicious transactions usually have been successfully executed. Failure handling can do nothing because no failure occurred. Attack recovery is supposed to remove the effects of malicious transactions after they are committed.

The checkpoint techniques [24], [25] also do not work for efficient attack recovery. A checkpoint rolls back the whole system to a specific time. All work, including both malicious transactions and normal transactions, after the specific time will be lost, especially when the delay of the IDS is very long. In addition, checkpoints introduce extra storage cost.

Distributed Computing Systems. De-centralized distributed computing is becoming more and more popular. In distributed computing models, transactions specifications cannot be accessed in a center node. They are carried by workflow itself or stored in a distributed style. In either case, our theories are still practical. We need to process the specifications of distributed transactional processes in a distributed style.

In some work such as [3], security and privacy is important, and the whole specification of workflows avoids being exposed to all processing nodes to protect privacy. Our theories are based on the dependence relations among transactions. The specification can be best protected by exposing only dependent relations to the recovery system.

Self-Healing Database Systems In [1], [28], When intrusions have been detected by the IDS, the database system isolates and confines the impaired data. Then, the system carries out recovery for malicious transactions. ITDB has actually implemented the on-the-fly damage assessment and repair algorithm proposed in [1]. In [30], the authors have proposed a general isolation algorithm. However, this algorithm cannot be directly implemented on top of an off-the-shelf DBMS. ITDB has developed a novel SQL rewriting approach to implement the algorithm. Finally, the design and implementation of almost every key ITDB component have been described in detail in previous publications [29], [32], [27], [31], [33], [1], Recent work considering more detailed dependency relations has been published in [51], [50].

However, this work is the first work on fundamental theories regarding the feasibility of self-healing distributed transactional processes.

7. Conclusions

In this paper, we proposed a dead-lock free self-healing algorithm for distributed transactional processes. We defined different IDS report order and discussed dead-lock, progress, and terminating problems regarding the distributed algorithm by introducing a serial theorems. Our results can serve as guidelines to design other dead-lock free and effective distributed recovery algorithms.

References

[1] P. Ammann, S. Jajodia, and P. Liu. Recovery from malicious transactions. *IEEE Transaction on Knowledge and Data Engineering*, 14(5):1167–1185, 2002.

[2] P. Ammann, S. Jajodia, C.D. McCollum, and B.T. Blaustein. Surviving information warfare attacks on databases. In *the IEEE Symposium on Security and Privacy*, pages 164–174, Oakland, CA, May 1997.

[3] Vijayalakshmi Atluri, Soon Ae Chun, and Pietro Mazzoleni. A chinese wall security model for decentralized workflow systems. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 48–57. ACM Press, 2001.

[4] D. Barbara, R. Goel, and S. Jajodia. Using checksums to detect data corruption. In *Int'l Conf. on Extending Data Base Technology*, Mar 2000.

[5] Qiming Chen and Umeshwar Dayal. Failure handling for transaction hierarchies. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 245–254. IEEE Computer Society, 1997.

[6] C. Y. Chung, M. Gertz, and K. Levitt. Demids: A misuse detection system for database systems. In *14th IFIP WG11.3 Working Conference on Database and Application Security*, 2000.

[7] D. E. Denning. An intrusion-detection model. *IEEE Trans. on Software Engineering*, SE-13:222–232, Feb. 1987.

[8] Johann Eder and Walter Liebhart. Workflow recovery. In *Conference on Cooperative Information Systems*, pages 124–134, 1996.

[9] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, September 2002.

[10] Gregory R. Ganger, Pradeep K. Khosla, Mehmet Bakkaloglu, Michael W. Bigrigg, Garth R. Goodson, Semih Oguz, Vijay Pandurangan, Craig A. N. Soules, John D. Strunk, and Jay J. Wylie. Survivable storage systems. In *DARPA Information Survivability Conference and Exposition*, volume 2, pages 184–195, Anaheim, CA, 12-14 June 2001. IEEE.

[11] T.D. Garvey and T.F. Lunt. Model-based intrusion detection. In *the 14th National Computer Security Conference*, Baltimore, MD, October 1991.

[12] R. Graubart, L. Schlipper, and C. McCollum. Defending database management systems against information warfare attacks. Technical report, The MITRE Corporation, 1996.

[13] P. Helman and G. Liepins. Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Trans. on Software Engineering*, 19(9):886–901, 1993.

[14] K. Ilgun. Ustat: A real-time intrusion detection system for unix. In *the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1993.

[15] R. Jagannathan and T. Lunt. System design document: Next generation intrusion detection expert system (nides). Technical report, SRI International, Menlo Park, California, 1993.

[16] H. S. Javitz and A. Valdes. The sri ides statistical anomaly detector. In *Proceedings IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA, May 1991.

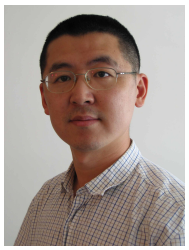
[17] David R. Jefferson. Virtual time. *ACM Transaction on Programming Languages and Systems*, 7(3):404–425, July 1985.

[18] J. Knight, K. Sullivan, M. Elder, and C. Wang. Survivability architectures: Issues and approaches. In *the 2000 DARPA Information Survivability Conference & Exposition*, pages 157–171, CA, June 2000.

[19] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In *CCS'03*, pages 251–261, Washington, DC, USA, October 27-31 2003.

[20] T. Lane and C.E. Brodley. Temporal sequence learning and data reduction for anomaly detection. In *5th ACM Conference on Computer and Communications Security*, San Francisco, CA, Nov 1998.
[21] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *2001 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.
[22] Wenke Lee, Sal Stolfo, and Kui Mok. A data mining framework for building intrusion detection models. In *1999 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1999.
[23] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4):227–261, 2000.
[24] Jun-Lin Lin and Margaret H. Dunham. A survey of distributed database checkpointing. *Distributed and Parallel Databases*, 5(3):289–319, 1997.
[25] Jun-Lin Lin and Margaret H. Dunham. A low-cost checkpointing technique for distributed databases. *Distributed and Parallel Databases*, 10(3):241–268, 2001.
[26] Yi-bing Lin and Edward D. Lazowska. A study of time warp rollback machanisms. *ACM Transactions on Modeling and Computer Simulations*, 1(1):51–72, January 1991.
[27] P. Liu. Dais: A real-time data attack isolation system for commercial database applications. In *the 17th Annual Computer Security Applications Conference*, 2001.
[28] P. Liu, P. Ammann, and S. Jajodia. Rewriting histories: Recovery from malicious transactions. *Distributed and Parallel Databases*, 8(1):7–40, 2000.
[29] P. Liu and S. Jajodia. Multi-phase damage confinement in database systems for intrusion tolerance. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 191–205, Nova Scotia, Canada, June 2001.
[30] P. Liu, S. Jajodia, and C.D. McCollum. Intrusion confinement by isolation in information systems. *Journal of Computer Security*, 8(4):243–279, 2000.
[31] Peng Liu and Ying Wang. The design and implementation of a multiphase database damage confinement system. In *the 2002 IFIP WG 11.3 Working Conference on Data and Application Security*, 2002.
[32] P. Luenam and P. Liu. Odar: An on-the-fly damage assessment and repair system for commercial database applications. In *the 2001 IFIP WG 11.3 Working Conference on Database and Application Security*, 2001.
[33] P. Luenam and P. Liu. The design of an adaptive intrusion tolerant database system. In *IEEE Workshop on Intrusion Tolerant Systems*, 2002.
[34] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, H. S. Javitz, A. Valdes, P. G. Neumann, and T. D. Garvey. A real time intrusion detection expert system (ides). Technical report, SRI International, Menlo Park, California, 1992.
[35] Teresa Lunt and Catherine McCollum. Intrusion detection and response research at DARPA. Technical report, The MITRE Corporation, McLean, VA, 1998.
[36] T.F. Lunt. A survey of intrusion detection techniques. *Computers & Security*, 12(4):405–418, Jun. 1993.
[37] J. McDermott and D. Goldschlag. Towards a model of storage jamming. In *the IEEE Computer Security Foundations Workshop*, pages 176–185, Kenmare, Ireland, June 1996.
[38] D. Medhi and D. Tipper. Multi-layered network survivability - models, analysis, architecture, framework and implementation: An overview. In *the 2000 DARPA Information Survivability Conference & Exposition*, pages 173–186, CA, June 2000.
[39] B. Mukherjee, L. T. Heberlein, and K.N. Levitt. Network intrusion detection. *IEEE Network*, pages 26–41, Jun. 1994.
[40] D. Samfat and R. Molva. Idamn: An intrusion detection architecture for mobile networks. *IEEE J. of Selected Areas in Communications*, 15(7):1373–1380, 1997.
[41] S. Sekar, M. Bendre, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *2001 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.
[42] S.-P. Shieh and V.D. Gligor. On a pattern-oriented model for intrusion detection. *IEEE Trans. on Knowledge and Data Engineering*, 9(4):661–667, 1997.
[43] V. Stavridou. Intrusion tolerant software architectures. In *the 2001 DARPA Information Survivability Conference & Exposition*, CA, June 2001.
[44] S. Stolfo, D. Fan, and W. Lee. Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, 1997.
[45] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A. N. Soules, and Gregory R. Ganger. Self-securing storage: Protecting data in compromised systems. In *Operating Systems Design and Implementation*, pages 165–180, San Diego, CA, 23-25 October 2000. USENIX Association.
[46] Jian Tang and San-Yih Hwang. A scheme to specify and implement ad-hoc recovery in workflow systems. *Lecture Notes in Computer Science*, 1377:484–498, 1998.
[47] Jay J. Wylie, Mehmet Bakkaloglu, Vijay Pandurangan, Michael W. Bigrigg, Semih Oguz, Ken Tew, Cory Williams, Gregory R. Ganger, and Pradeep K. Khosla. Selecting the right data distribution scheme for a survivable storage system. Technical Report CMU-CS-01-120, Carnegie Mellon University, 2001.
[48] Jay J. Wylie, Michael W. Bigrigg, John D. Strunk, Gregory R. Ganger, Han Kiliccote, and Pradeep K. Khosla. Survivable information storage systems. *IEEE Computer*, 33(8):61–68, August 2000.
[49] Meng Yu, Peng Liu, and Wanyu Zang. Intrusion masking for distributed atomic operations. In *The 18th IFIP International Information Security Conference*, pages 229–240, Athens Chamber of Commerce and Industry, Greece, 26-28 May 2003. IFIP Technical Committee 11, Kluwer Academic Publishers.
[50] Meng Yu, Peng Liu, and Wanyu Zang. Self-healing workflow systems under attacks. In *The 24th International Conference on Distributed Computing Systems(ICDCS'04)*, pages 418–425, 2004.
[51] Meng Yu, Peng Liu, and Wanyu Zang. Multi-version based attack recovery of workflow. In *The 19th Annual Computer Security Applications Conference*, pages 142–151, Dec. 2003.

Wanyu Zang received the M.S. degrees in Computer Science from the Northeastern University, China, in 1998 and Ph.D. degree in Computer Science from Nanjing University, China in 2001. She is currently a visiting research professor at Computer Science department of Monmouth University. Her research interests include computer security and wireless networks.



Meng Yu received the M.S. degrees in Computer Science from the Northeastern University, China, in 1998 and Ph.D. degree in Computer Science from Nanjing University, China in 2001. He has been worked as a postdoctoral research scholar at University of Maryland, Baltimore County and Penn. State University, University Park from 2002 to 2004. He is currently an assistant professor at Computer Science department of Monmouth University. His research interests include computer security, especial on distributed and database systems.

Peng Liu received his B.S. and M.S. degrees from the University of Science and Technology of China, and his Ph.D. degree from George Mason University in 1999. He is an associate professor of Information Sciences and Technology and director of the Cyber Security Lab at Penn State. His research interests are in all areas of computer and network security. Dr. Liu has published a book and about 70 refereed technical papers. His research has been sponsored by DARPA, NSF, DOE, DHS, AFRL, NSA, CISCO, HP, Japan JSPS, and Penn State. Dr. Liu is a recipient of the DOE Early CAREER PI Award.