Recovering from Malicious Tasks in Workflow Systems

DEXA August 2005

Yajie Zhu, Tai Xin and Indrakshi Ray
Department of Computer Science
Colorado State University
Fort Collins, Colorado

{zhuy,xin,iray}@cs.colostate.edu.
Presentation Organization

- What is a workflow?
- What can a malicious user do?
- How do we repair from malicious attacks?
- What will we do in future?
WHAT IS A WORKFLOW?
A workflow consists of a set of tasks that together achieve some business objective.

Tasks in a workflow must be properly coordinated to ensure correct execution of the workflow.

Such co-ordination is achieved through dependencies.

A workflow is formally denoted as $W_i = <T,D,C>$

- $T$: set of tasks in the workflow
- $D$: set of dependencies in the workflow
- $C$: set of completion sets in the workflow
Workflow Tasks

- Workflow $W_i$ consists of tasks $\{T_{i1}, T_{i2}, \ldots, T_{in}\}$
- Each task $T_{ij}$ performs a logical unit of work
  - It is executed atomically
  - Consists of data operations
    - read operations ($r_{ij}[x]$)
    - write operations ($w_{ij}[y]$)
  - Associated with primitives
    - Begin ($b_{ij}$)
    - Abort ($a_{ij}$)
    - Commit ($c_{ij}$)
States of Task $T_{ij}$

```
        un_{ij}   submit   in_{ij}   begin   ex_{ij}   done
                |   roll_back |
                |             |
        pr_{ij}   roll_back |
                |             |
        ab_{ij}   commit   cm_{ij}
```
Control Flow Dependencies

A control flow dependency $T_{ki} \rightarrow T_{kj}$ specifies how the execution of primitives of task $T_{ki}$ causes the execution of primitives of task $T_{kj}$.

Types of control flow dependencies

- commit dependency, abort dependency, begin dependency,
- begin-on-commit dependency, begin-on-abort dependency,
- force-begin-on-begin dependency, force-begin-on-commit dependency,
- force-begin-on-abort dependency, exclusion dependency
A data-flow dependency $T_{ik} \rightarrow_{df} T_{ij}$ signifies that there is an output produced by task $T_{ik}$ which is an input to task $T_{ij}$.

A read-write dependency $T_{kl} \rightarrow_{rw} T_{ij}$ signifies that there exists some data item $x$ such that:

- $T_{ij}$ reads $x$ after $T_{kl}$ has updated it.
- if any $T_{pq}$ updates $x$ after $T_{kl}$ has updated $x$ but before $T_{ij}$ reads it, then $T_{pq}$ is aborted.
Example Workflow

Tasks in the workflow

- $T_{10}$: Reserve a car from Company B
- $T_{11}$: Reserve a ticket on Airlines A
- $T_{12}$: Purchase the Airlines A ticket
- $T_{13}$: Cancel the airlines reservation
- $T_{14}$: Reserve a room in Resort C
- $T_{15}$: Cancel the car reservation

Example completion sets

- $\{T_{14}, T_{10}\}$, $\{T_{14}, T_{11}, T_{12}\}$,
- $\{T_{11}, T_{13}\}$, $\{T_{10}, T_{15}\}$
Example Workflow (2)
WHAT IS THE PROBLEM?
Security of a Workflow

- Vulnerabilities cannot be completely eliminated from a system.
- Preventive measures for protecting the system are not enough.
- Workflow may be subjected to attacks.
- Malicious user can create illegal tasks or execute corrupt tasks in a workflow.
- *Malicious tasks* are the committed tasks submitted by attacker.
- *Malicious workflow* contains at least one malicious task.
Malicious Tasks

- Malicious tasks may corrupt database items
- Malicious tasks may trigger some other tasks
- Presence of dependencies help spread damage
Corrupting Data Items

Malicious Task

\[ T_{ij} \text{ Writes } X \]

\[ X \text{ Reads } Y \text{ Writes } T_{kl} \]

Good Task
Dependencies Spreading Damage

Good Task \( T_{ij} \) writes to \( X \) and \( Y \).

Malicious Task \( T_{ik} \) reads from \( fbb \).

Graphical representation:
- \( T_{ij} \) writes to \( X \) and \( Y \).
- \( T_{ik} \) reads from \( fbb \).
HOW DO WE REPAIR FROM SUCH AN ATTACK?
Information needed for Repair

- Recovery algorithm needs to know
  - the actions that need to be performed during recovery
    - stored in the corresponding workflow schema
  - state of the workflow after some malicious attacks
    - stored in workflow log records
- All such information is stored in stable storage
Workflow Schema

- Workflow schema defines the type of a workflow
- Workflow is an instance of some workflow schema
- Workflow schema is specified by
  - types of inputs needed by workflow instances
  - types of outputs generated by workflow instances
  - specification of the types of tasks
  - dependencies between different types of tasks
  - set of completion sets for this type of workflow
Workflow Log Records

The following log records get written in stable storage

- Execution of begin primitive of workflow $W_i$: $< START W_i, WS_i >$
- Execution of complete primitive of workflow $W_i$: $< COMPLETE W_i >$
- Execution of begin primitive of task $T_{ij}$: $< START T_{ij} >$
- Execution of abort primitive of task $T_{ij}$: $< ABORT T_{ij} >$
- Execution of commit primitive of task $T_{ij}$: $< COMMIT T_{ij} >$
- Execution of write operation $w_{ij}[X]$: $< T_{ij}, X, v, w >$
Recovery Algorithm Overview

- Recovery algorithm proceeds in four phases
  - Phase 1: Undo malicious workflows
  - Phase 2: Find all affected tasks
    - Add good tasks that read corrupted data items to the list of affected tasks
    - Add tasks that are control-flow dependent on the affected tasks and which must be aborted to the affected list
  - Phase 3: Undo all the affected tasks
  - Phase 4: Resubmit the incomplete workflow to the scheduler
WHAT WILL WE DO IN FUTURE?
Conclusion and Future Work

Contributions
- Malicious tasks in a workflow can cause damage
- Dependencies in a workflow help spread the damage
- Our algorithm
  - finds affected tasks to assess the damage
  - repairs the damage by undoing malicious and affected tasks
  - maintains dependencies during the recovery process

Future Work
- Formalize the notion of correct execution and correct repair of a workflow
- Investigate how to recover from malicious transactions in other advanced transaction processing models