Workflow Fault Tolerance for Kepler

Sven Köhler, Thimothy McPhillips, Sean Riddle, Daniel Zinn, Bertram Ludäscher
**Scientific Workflows**
- Automate scientific pipelines
- Have long running computations
- Often contain stateful actors

**Workflow execution can crash because of …**
- Hardware failures
- Power outages
- Buggy / malicious actors, …

**Current approach:** Start workflow from the beginning
Current Fault Tolerance Solutions ...

- Manage actor failures or sub-workflow failures AND their effects
  - Atomicity and provenance support for pipelined scientific workflows [Wang et al.]
  - Ptolemy’s “Backtrack” [Feng et al.]

- Use caching strategies for faster re-execution
  - W.A.T.E.R.S. memoization [Hartman et al.]
  - “Skip over” strategy [Podhorszki et al.] (CPES)
Our Fault Tolerance Approach

- Recovery based on readily available Provenance

1. Create a uniform model for workflow descriptions and provenance
2. Record actor state in provenance in relation to invocations
3. After a workflow crash: Use provenance data in our uniform model and start recovery

- Different strategies for recovery
## Our Recovery Strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Description</th>
</tr>
</thead>
</table>
| Naive        | - Restart the workflow without using provenance  
               - Re-executes everything                                                                                                                   |
| Replay       | - Use **basic provenance** to speed up recovery  
               - Re-execute stateful actor with input from provenance (*replay*)  
               - Restore all queues  
               - Resume the workflow according to the model of computation                                                                                   |
| Checkpoint   | - **extension** of *replay* strategy  
               - Use **checkpoints** (state of actors stored in provenance)  
               - Reset stateful actors to recorded state  
               - Replay successful invocations after the checkpoint  
               - Restore queue content  
               - Resume the workflow                                                                                                                      |
Example: Checkpoint in SDF

Workflow with a mix of stateful and stateless actors

Corresponding schedule of the workflow with a fault during invocation B:2
Execution with a Failure

Execution of the previous workflow

Checkpoints for actor B and D but not for C

At invocation B:2 - Crash

Tokens $t_4$ and $t_7$ - in queue

Token $t_9$ - to be restored

Token $t_{10}$ - to be deleted
Stages of Checkpoint Recovery

Actor A (stateless)  Actor B (stateful)  Actor C (stateful)  Actor D (stateful)  Actor E (stateless)

1. (done)  (new instance)  (new instance)  (new instance)  (new instance)
   - State B1
   - t1
   - iterate
   - C:1
   - t7
   - State D1
   - t4
   - t10

2. restore states
   - replay actors
   - restore queues
   - reset scheduler & continue execution

3. t9
   - running
   - B:2

4. t
Prototype Implementation in Kepler

- Using Kepler with the Provenance Recorder
- Extensions to the Provenance Recorder:
  - Record serialized tokens
  - Extend the provenance schema
  - Add queries
- Recovery Extension in the SDF Director:
  - Serialize states after one iteration of the SDF schedule
  - Black-list to prevent capturing transient actor information
  - White-list if actors are annotated with state-information
Prototype Implementation in Kepler

- **Upon restart:**
  - SDF director checks provenance information
  - SDF director calls the recovery engine

- **Recovery:**
  - Restore the internal state of actors
  - Replay successful invocations using input tokens from provenance
  - Restore content of all queues
  - Return to SDF director with information about where to resume
Evaluation

Synthetic Workflow

Results

- Completion
- Recovery
- Time to Crashpoint

Bar chart showing times in seconds for different scenarios:
- Successful Run
- Crashed Run
- Complete ReRun
- Replay
- Checkpointing

Graph showing times for each scenario.

UC Davis: S. Koehler, T. McPhillips, S. Riddle, D. Zinn, B. Ludaescher

2/16/2011
Conclusion

- **Advantages of our strategy:**
  - Efficient workflow recovery using readily available information
  - Quick constant time recovery (checkpoint strategy)
  - Generalized approach, saving labor
  - Robustness

- **Disadvantages of previous strategies:**
  - Required labor-intensive customized systems
  - Failure required restarting long-running workflows from the beginning
  - Caching only works for stateless actors
  - Caching only provides a partial recovery