WHESL: A Workflow Hierarchy-aware Fault Tolerance System

NC STATE UNIVERSITY

Subhendu Behera[1], Dong H. Ahn[2], Stephen Herbein[2], Frank Mueller[1], Barry Rountree[2]

[1]North Carolina State University [2]Lawrence Livermore National Laboratory

Lawrence Livermore National Laboratory

Motivation

- Scientific workflows suffer greatly from faults/exceptions
  - Complexity
  - Dependency
- Existing solutions
  - Resource over-provisioning and checkpoint/restart
    - Not scalable
    - Resource wastage
    - Applicable to individual components in workflows
    - Lack of coordination

Solution

- Holistic approach
- Requirements for an efficient fault tolerance system
  - Detection (both temporal and spatial exceptions)
  - Propagation (notification to relevant workflow entities)
  - Recovery without conflicts (coordination among entities)

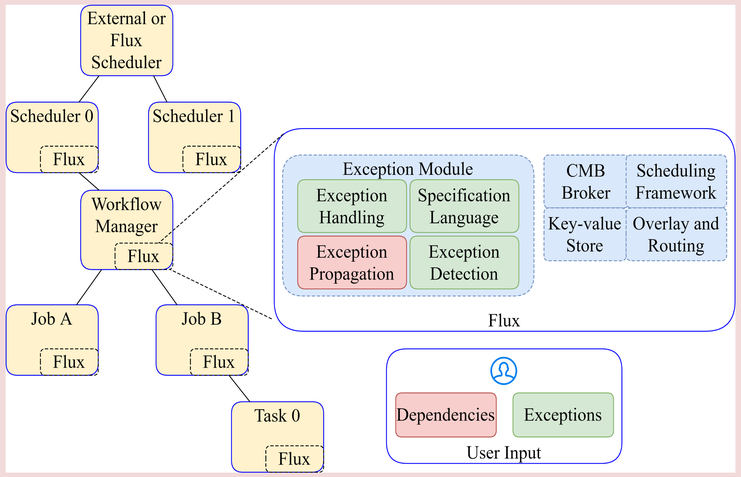
Contribution

- WHESL: Workflow Hierarchy-aware Exception Specification Language
  - Specify and handle faults among disparate components in a coordinated and programmable manner

Approach

- Flux
  - Scalable, hierarchical resource and job manager
  - High job submission and scheduling throughput
- Implemented as a module within Flux
- Exception specification language
  - Exception-component relationship
  - Detection method
  - Handling/recovery method
  - Dependencies within a workflow
- Detection
  - Componentization of vulnerable node-local features
    - HWLOC library: CPU, GPU, RAM etc.
    - pre-defined components: VM, process, files etc.
  - Node-local log parsing → temporal exception patterns
  - Exception transformation → spatial exceptions

Architecture



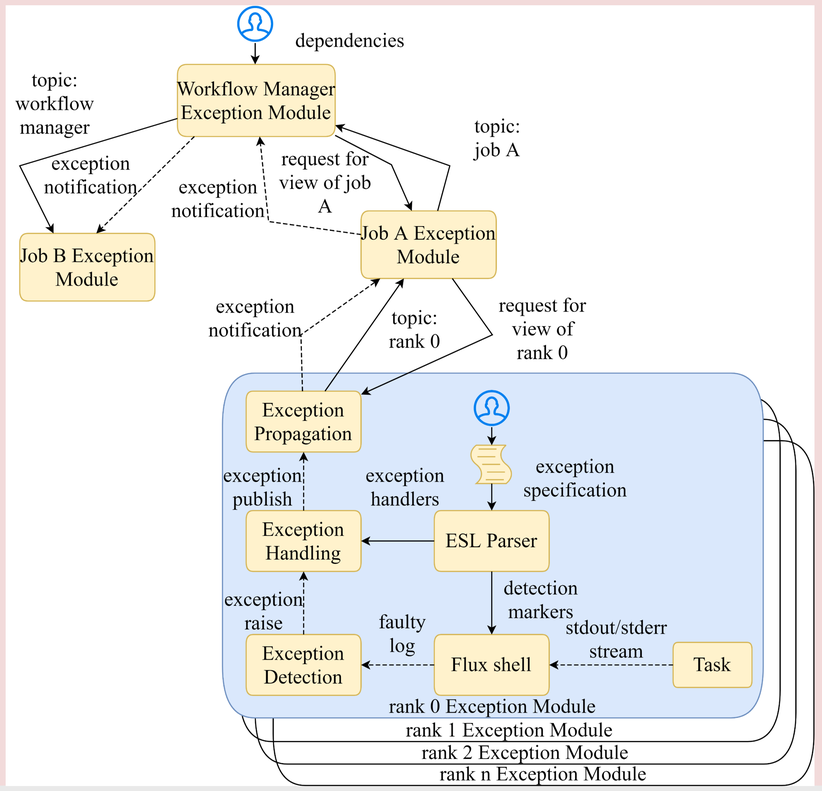
Exception Specification Language

```
<system> -> { <component-list> } { <workflow-roles> } { <exception-list> } { <dependency-list> }

<component-list> -> <component> , <component-list> | EMPTY

<workflow-roles> -> <workflow-role> , <workflow-roles> | EMPTY
<workflow-role> -> TASK | JOB | WORKFLOW | DEPENDENT_TASK | DEPENDENT_JOB

<exception-list> -> <exception> ; <exception-list> | EMPTY
<exception> -> { ID; <origin-component>; <detection>; <handler-list> } //ID: exception name

<detection> -> { <location> , markers }
<location> -> STDOUT | STDERR
<markers> -> <marker-series> | <marker-timeseries>

<handler-list> -> { <handler> } ; <handler-list> | EMPTY
<handler> -> <workflow-role> , <source-exception> , <transformation-exception> , { <action-sequence> }
<action-sequence> -> <action-key> , <action-sequence> | EMPTY
<action-key> -> PUBLISH | WAIT | KILLTASK

<dependency-list> -> dependency, <dependency-list> | EMPTY
<dependency> -> { <source> ; <dependent>; <dependency-level> , <dependency-type>}
<source> -> { <jobid> , <startrank> , <endrank> }
<dependent> -> { <jobid> , <startrank> , <endrank> }
<dependency-type> -> ALL | TASK | <component> | <component-type>
<dependency-level> -> JOB | NODE
```

Approach Continues

- Propagation
  - Based on dependencies within workflow
  - Using Flux's pub-sub APIs
  - Depth first search to find exception source and dependents
  - View (grouping) of components at source node or job level
- Recovery
  - Role-based recovery paradigm
  - Each notified entity performs a series of actions
    - Example: PUBLISH, WAIT, KILLTASK

Operational View



Case Study (Node Failure)

- Without our solution
  - Job detects unresponsive node during communication
  - Flux detects through heartbeat timeout
  - Workflow manager detects if job fails
  - Independent recovery → conflicts such as duplicate task
- With our solution
  - Peer node detects node failure, notifies job
  - Job
    - Option 1: Recovers node and restarts task
    - Option 2: Request node from workflow manager
  - Dependent jobs get notified -> throttle

Acknowledgments: This research was supported by an appointment to the Lawrence Livermore National Laboratory's Compute Scholar Program.

This work was also performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-POST-813928).