Grid Workflows
Current Stage and Future Directions

Institute of Computer Science, University of Innsbruck, Austria

Radu Prodan
radu@dps.uibk.ac.at
Outline

- Scientific workflows
- Workflow specification
  - End-user programming
  - Automatic composition
- Workflow interoperability
- Workflow provenance
- Workflow scheduling
- Workflow fault tolerance
- From scientific to industrial applications
- Summary
Simulation of Danube Flooding

- Workflows are complex applications dynamically constructed from existing services.
- Different organisations cooperate to predict the flooding behaviour of the Danube by using Grid sensors, computing and data storage resources, as well as modelling and simulation services.
State-of-the-Art in Scientific Grid Workflows

- **Grid workflow** = collection of computational, communication, and interaction tasks that are processed in a well-defined order to achieve a specific goal
- Mostly static workflows
- Fixed control flow dependencies
  - DAGs, if, switch, while and for loops
- Fixed data flow dependencies
- Mechanisms for expressing parallelism
  - Independent tasks, parallel for loops
- Two levels of workflow specification
  - XML-based programming
  - Graphical modeling
- No widely accepted workflow modeling and programming standard in the Grid community
“Small” Scientific Workflows

- WIEN2k
  - Material Science

Existing Scientific Workflows with thousands of tasks remind a lot of compiler-internal Abstract Syntax Tree representations

- Hydrology

- Montage
  - Astronomy

Third EchoGrid Workshop, Athens, June 9-10, 2008
User-level Workflow Composition

- Paper in CCGrid conference, 2007
- Workflows follow an imperative programming model
- Java, C, Fortran, assembly are imperative programming languages
  - Programs are a workflow of instructions
  - Skilful reuse of data stores is the key for **performance**
    - Registers, caches, main memory, hard disks, storage systems
  - Assignments are destructive ⇒ **programming errors**
    - A data store containing a wrong value = bug

Grid workflow languages
- Instruction = component, service, task, activity
- Data ports = variables (data stores)
- Data flow dependencies = assignment statements
- Control flow dependencies: sequence, while, for, if, switch, goto

Grid workflow languages are Turing complete
- Same complexity and prone to the same **programming errors** as assembly languages
Two Stage Programming

- **Formal functional specification ("what")**
  - Written by the application scientist
  - Based on expression (function) evaluation model
  - No explicit memory allocation
  - No explicit variable assignment

- **Imperative workflow coordination ("how")**
  - Written by the computer scientist (manually or through a compiler)
  - Grid distribution and parallelism

- **Correctness checker**
  - Ensures correct coordination
  - Augment data store with semantics (value names)
Recursive definition rather than sequential loop

```haskell
fun wien2k : string * int -> string
    wien2k(in_pack, i) =
        if i = 0 then wien2k_iter(in_pack)
        else wien2k_iter(wien2k(in_pack, i-1))
```

Material science workflow application

```
output wien2k(in_pack(), n())
precondition n() >= 0
postcondition converged(wien2k(in_pack(), n()))
```
WIEN2k Case Study: Correct and Efficient Coordination

coordination wien2k
input in_pack : () -> string
A = wien2k(in_pack, 0)
i = 0
while not converged(A) do
    assert i >= 0
    B = wien2k(in_pack, i) <- A
    A = wien2k(in_pack, i+1) | wien2k < B
    i = i + 1
    assert converged(wien2k(in_pack, i) <- A)
output wien2k(in_pack, i) <- A

- Full activities
- Fetch activities
- Step activities

- Annotate data ports with semantic information from the specification to ensure its correctness (value names)
Automatic Workflow Composition

- Automatic creation of (partial) workflows at
  - Resource level (runtime)
  - Developer level (offline)
- Build a domain-specific ontology comprising semantic descriptions of activity types
- Start from semantic description of the input and output data
- Use the ontology to automatically compose activities in a workflow that produces this I/O mapping
  - Automatically resolve data dependencies using the ontology information
- Drawbacks
  - Shifts the complexity from workflow composition to ontology creation
  - Semantics technology is very slow which is at odds with dynamic workflows
- Build comprehensive domain-specific ontologies as community efforts

MeteoAG workflow

Third EchoGrid Workshop, Athens, June 9-10, 2008
Grid Workflow Interoperability

- Most workflow systems disallow execution of a workflow with a different system
- Workflow interoperability is crucial for the survival of most workflow systems
  - Work cannot be justified in application development for a specific system that might be terminated when funding runs out
- Interoperability facets
  - Common high-level workflow language (XML-based)
    - $O(1)$ complexity
  - Intermediate language seen by enactment engine
    - $O(n)$ complexity
  - Transformation bridges among workflow systems
    - $O(n^2)$ complexity
  - Interoperable middleware services
    - Needs standard protocols like Web services
Workflow Provenance

- Provenance = origin or the source of something, or the history of the ownership or location of an object
- For workflows, provenance is any kind of historical data related to the development and execution of a workflow together with its source
- Provenance is principally needed for characterization, reproducibility and verification of results
- Currently provenance information is today highly fragmented
  - emails, Wiki entries, databases, journal references, code comments, compiler options
- Many workflow systems claim support for provenance if they have some monitoring services
- Provenance is much more than monitoring
  - Constraint specification (pre- and post-conditions)
  - Monitoring and recording
  - Provenance store
  - Policy management
  - Audit reports
- Provenance can support many Grid middleware services
  - performance analysis, scheduling, resource management, fault tolerance
Workflow Scheduling

- Map a workflow of \( N \) tasks onto \( M \) Grid sites
- Only workflow makespan addressed so far
  - NP-complete optimization problem
  - Many heuristic algorithms that converge to good solutions
- Real overheads not considered in evaluation of the objective functions
- Simulation not based on real workloads
  - Execution completely different from the plan
- **Quality of Service** negotiation and enforcement
  - The challenge is to build QoS enforcement strategies over best effort protocols
  - Best effort TCP/IP protocol
  - Local job queuing systems operate in best effort mode
- SLAs (business) → QoS (resource management) → metrics (fabric)
Multi-Criteria Scheduling

- Optimization of multiple non-functional parameters
  - execution time, cost, reliability, security, availability
- Taxonomy of workflow scheduling

workflow structure dependence

structure dependent (e.g., execution time)  structure independent (e.g., economic cost)

workflow structure

cost calculation method

additive (e.g., economic cost)  multiplicative (e.g., security)  concave (e.g., bandwidth)

intradependence

intradependent (e.g., execution time)  non-intradependent (e.g., economic cost)

\[ \prod_{i=1}^{n} x_i \equiv \sum_{i=1}^{n} \ln x_i \]
Bi-criteria Scheduling

- Primary criterion plus a flexible constraint for the secondary criterion
- Two-phase optimization based on dynamic programming
- Optimize the schedule for the primary criterion
  - NP-complete for intradependent criteria (execution time using the HEFT algorithm)
  - Trivial for non-intradependent criterion (simple greedy approach) (cost)
  - Result is a preliminary solution
- Modify the preliminary solution, optimizing the secondary criterion
  - The primary criterion kept within the flexible limit
- Problem described as multiple choice knapsack problem
Fault Tolerance Questionnaire

- Questionnaire sent to workflow system developers
  - Fault detection, recovery, and prevention
- ASKALON, Chemomentum, Escogitare, GWEE, GWES, Pegasus, P-GRADE, ProActive, Triana, UNICORE 5

<table>
<thead>
<tr>
<th>Hardware level</th>
<th>Task level</th>
<th>Middleware level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine crashed/down</td>
<td>Memory leak</td>
<td>Authentication failed</td>
</tr>
<tr>
<td>Network down</td>
<td>Uncatched exception</td>
<td>Job submission failed</td>
</tr>
<tr>
<td>Operating system level</td>
<td>Deadlock / Livelock</td>
<td>Job hanging in the local resource manager queue</td>
</tr>
<tr>
<td>Disk quota exceeded</td>
<td>Incorrect output data</td>
<td>Job lost before reaching the local resource manager</td>
</tr>
<tr>
<td>Out of memory</td>
<td>Missing shared libraries</td>
<td>Too many concurrent requests</td>
</tr>
<tr>
<td>Out of disk space</td>
<td>Job crashed</td>
<td>Service not reachable</td>
</tr>
<tr>
<td>File not found</td>
<td>Incorrect output data</td>
<td>Failure</td>
</tr>
<tr>
<td>Network congestion</td>
<td>Job crashed</td>
<td>File staging failure</td>
</tr>
<tr>
<td>CPU time limit exceeded</td>
<td>Workflow level</td>
<td>User level</td>
</tr>
<tr>
<td>Inifinite loop</td>
<td>User-definable exceptions</td>
<td></td>
</tr>
<tr>
<td>Input data not available</td>
<td>User-definable assertions</td>
<td></td>
</tr>
<tr>
<td>Input error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data movement failed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Faults Detected by an Average System

- Hardware: 80% current version, 100% future version
- Operating System: 40% current version, 60% future version
- Middleware: 60% current version, 80% future version
- Task: 20% current version, 40% future version
- Workflow: 60% current version, 80% future version
- User: 40% current version, 60% future version
Fault Recovered by an Average System

- Hardware: 40% (current version), 80% (future version)
- Operating System: 20% (current version), 60% (future version)
- Middleware: 20% (current version), 60% (future version)
- Task: 20% (current version), 60% (future version)
- Workflow: 20% (current version), 60% (future version)
- User: 20% (current version), 60% (future version)
From Scientific to Industrial Applications
Massively Multiplayer Online Games
Over the last 10 years the market size has increased by 20 fold
60 million people by 2011
Entertainment Software Association (ESA)
  Size: 7 billion USD
  300% growth in the last 10 years
MMOG Challenges for the Grid

- Real-Time Online Interactive Applications (ROIA) as a new class of Grid applications
  - Multiple users share the same application instance
  - Impact the dynamics of the application as a community
- Increase the maximum number of players in one session
  - 64 in fast-paced First Person Shooter (FPS) action games
- Virtual Organisations
  - Ad-hoc and dynamic
  - Anonymous users sharing pseudonyms
  - Cheating prevention
- On-demand provisioning of compute servers to game sessions based on user load
  - Avoid over-provisioning
- Real-time QoS requirements
  - State update rate per second from game servers to game players
  - 10 – 60 Hz in fast-paced FPS action games

RISGE-RG Meeting, OGF 23, Barcelona, Spain, June 3, 2008
Summary

- Workflow is a low-level paradigm for application scientists
  - Workflow parallelisation and distribution are runtime optimisations that should be hidden to the application scientists

- Automatic workflow composition
  - Limited by building domain-specific ontologies

- Existing workflow systems comply to no standards and do not interoperate

- Scheduling makespan objective function does include large Grid overheads

- Scheduling validations are not based on real workloads

- QoS support on top of best effort protocols

- Multi-criteria scheduling

- Online games as a new class of socially important applications with huge market potential

- Invisible Grid is still far away
  - Move from programming to abstract modeling
  - Dynamic binding of model to implementation
  - Dynamic software deployment