Accountability for grid computing systems

Accountability is defined as “A is accountable to B when A is obliged to inform B about A’s past or future actions and decisions, to justify them, and to suffer punishment in the case of eventual misconduct”

Accountability is an important aspect of any computer system for assuring that every action executed in the system can be traced back to some entity

The dynamic and multi-organizational nature of grid systems requires effective and efficient accountability system
**Contributions**

- **We devise a distributed mechanism to capture provenance information available during the distributed execution**

- **We introduce the concept of accountability agents**

- **We develop a simple yet effective language to specify the accountability data**

- **We implement the prototype of the accountability system on an emulated grid testbed**

---

**Example 1.** Pete, a participant of the open science grid (Virtual Organization, VO) which links shared resources, performs a multidisciplinary simulation, *nwFluid linux*, that uses programs and data located at multiple locations. Even though Pete is affiliated with Purdue University, he can run program A at A-state University, and B at B-state University using input data from C-state University.
Overall Architecture of Accountable Grid Systems

Two approaches

- **Job-flow based approach**
  - Jobs flow across different organizational units
  - Long computation is often divided into many sub-jobs to be run in parallel
  - A possible approach is to employ point-to-point agents which collect data at each node that the job traverses

- **Grid node based approach**
  - Only interested in a given location in the flow and at a given instant of time for all jobs
  - Viewpoint is fixed

Combination of two approaches give complementary information
**Combination of two approaches**

**Service Provider**  Compute Node  Compute Node

**Job-Graph**

**Definition 1 (Job-Graph).** Let $N$ be a non-empty set of grid nodes. A job graph $G = (N, E)$ is a directed graph satisfying the following conditions: 1) each node $n \in N$ corresponds to a grid node characterized by indexes $i, j$, where $i$ denotes the unique node identifier and $j$ the computational units of the nodes 2) each edge $e = (n_i, n'_j) \in E$ denotes a sub-job assignment from the parent node $n_i$ to the child node $n'_j$; 3) there is a unique root node of the graph, that is $\exists n \in N \text{ s.t. } (n', n) \notin E$. 
Log sharing mechanism in multiple domains

- When the whole graph structure to be completed for a split or forwarded job, the agents relay job information to their father nodes upon requests of father nodes.

- The job-Graph is a single rooted structure, and the root node is the job entry point. Thus root node is finally able to assemble the whole job information of a job-Graph.

- Merging the local view represented by cover records completes a whole graph when each agent correctly shares the job information.

- Agents maintain complete view of a job, without generating a huge amount of overhead at a single node.

Accountability Policy Specification

- The accountability policies specify what and when to track.

- The accountability policies specify how each agent has to coordinate with agents in other locations.

- Policies are expressed by actions, capturing the main activities of an agent.
**Actions’ Representation**

**Example 2.** The following action specification states that agent AA@Purdue at Purdue University will collect user’s handle, job_id, process_id, and time stamp when a job is either transferred or completed.

```
collect_job_data(AA@Purdue, {Transferred, Completed}, {handle, job_id, process_id, timestamp}, purdue_db)
```

**Example 3.** Next actions specify the collection of the resource data associated with agent AA@Purdue to be executed every week day once an hour.

```
collect_resource_data(AA@Purdue, DATA, Week+{2,...,6}+1h, purdue_db)
DATA:={CPU cycle, memory consumption, network bandwidth}
```
Accountability Policy

- Policies can either be local to given location or shared by the agents.
- Local policies capture data as required by the grid node based approach.
- Shared policies apply to the job-flow based approach, and specify which job information has to be sent or received upon different job states.

**Definition 2 (shared, local policy)** An accountability policy is an expression of one of the following form: 1) A **shared policy** (among n organizations) is an action expression \( AE = \{A_1, \ldots, A_m\} \), specified according to Definition 1, such that \( \forall i \in [1, m], \forall j, k \in [1, n], j \neq k, A_i.\text{Site}_j = A_i.\text{Site}_k \)

2) A **local policy** (among n organizations) is an action expression \( AE = \{A_1, \ldots, A_m\} \), specified according to Definition 1, such that \( \exists i \in [1, m], \forall j, k \in [1, n], j \neq k, A_i.\text{Site}_j \neq A_i.\text{Site}_k \)

Accountability Grammar in BNF

- \(<\text{policy}\_\text{set}> := <\text{policies}>\)
- \(<\text{policies}> := <\text{policy}> <\text{policies}> | <\text{policy}>, <\text{policy}> := <\text{action}\_\text{specification}> | <\text{representatives}>, <\text{representatives}>, <\text{accountability}\_\text{data}> := <\text{symbol}>, <\text{accountability}\_\text{data}> := <\text{job}\_\text{flow}\_\text{based}> | <\text{grid}\_\text{node}\_\text{based}> | <\text{resources}> := <\text{resource}> <\text{period}> | <\text{constraints}>, <\text{state}> := \text{submitted} | \text{created} | \text{started} | \text{completed} | \text{pending} | \text{aborted} | \text{queued} | \text{suspended} | \text{active} | \text{done}, <\text{job}\_\text{type}> := \text{computational} | \text{transfer}, <\text{job}_{\text{flow}}\_\text{based}> := \text{handle} | \text{job}\_\text{id} | \text{process}\_\text{id} | \text{executable} | \text{SP}\_\text{id} | \text{IP}\_\text{id} | \text{file}\_\text{names} | \text{platform} | \text{time}\_\text{stamp}, <\text{grid}\_\text{node}\_\text{based}> := \text{memory}\_\text{consumption} | \text{cpu}\_\text{time} | \text{network}\_\text{bandwidth} | \text{disk}\_\text{bandwidth} | \text{IP}\_\text{destination} | \text{port}, <\text{constraints}> := \text{all}\_\text{process} | \text{life}\_\text{time} | \text{all}\_\text{day} | \text{weekdays} | \text{weekend}, <\text{job}_{\text{assigns}}> := \text{SP}\_\text{id} \leftarrow \text{job}\_\text{id}, <\text{strings}> := \text{authorization}\_\text{policy} | \text{usage}\_\text{policy}, <\text{resources}> := \text{authorization}\_\text{decision}, <\text{symbols}> := (\text{AND}) | (\text{OR}), <\text{period}> := \text{NUM} | \text{null}, <\text{resources}> := \text{null}, <\text{constraints}> := \text{null}.

- Shared and local policies are specified according to the grammar.
- Our grammar mainly consists of action specification, accountability data, which is job_flow_based or grid_node_based, and terminal variables such as state, names of data supported in the languages.
**ACCOUNTABILITY POLICY**

**Example 3.** A job is submitted to Purdue University SP and then assigned for execution to the RPs, A-state University, and B-state University. Purdue agrees to send job relation data (handle, job-id, subjob-id, RP-id, timestamp) to A-state and B-state when the processed job enters into active state. Additionally, A-state locally collects resource data (memory consumption, cpu time, network bandwidth, disk bandwidth) every day during the week.

The policies for such scenario are as follows:

**[at Purdue University]**

shared_policy :=
send_job_data (agent@Purdue, agents_in_job_relationPurdue, active, dataSetactive, job-id)
collect_job_data (agent@Purdue, active, dataSetactive, DBPurdue)

agents_in_job_relationPurdue := agent@A-state (AND) agent@B-state
dataSetactive := handle (AND) job-id (AND) subjob-id (AND) RP-id (AND) timestamp

**[at A-state University]**

local_policy :=
collect_resource_data (agent@A-state, dataSetlocal, time_constraintsA-state, DBA-state)
dataSetlocal := memory consumption (AND) cpu time (AND) network bandwidth (AND) disk bandwidth
time_constraintsA-state := weekdays (AND) all.days

---

**EXPERIMENTAL EVALUATIONS**

**Exp. 1 / Scalability with respect to the number of computing nodes**

![Graph showing scalability with respect to the number of computing nodes](image)

- The response time is computed as the difference between the time at which the user receives the result and the time at which a user submits the job.
- Blue bars show the overhead introduced by accountability, which is negligible.
Exp. 2 / Scalability with respect to the data volume

Data volume is the main concern for administrators. If a data volume is a concern, administrators can trade off accountability accuracy for performance.

Exp. 3 / Evaluation of shared vs local policies

Policy processing includes reading the policy from the XML file at local computer, and collecting or searching elements specified at the policy. Shared policies vs local policies with the same policy complexity. Support guideline to administrators.
Experimental Evaluations

Exp. 4 / Searching time for policy elements

- Search time required for the policy elements specified in different policies.
- Rightmost four elements collected by shared policies takes only 3 to 35 microseconds while others 1 to 18 milliseconds.

Conclusion

- We presented a distributed approach for end-to-end accountability in grid systems.
- We introduced the concept of accountability agents performing a wide range of information gathering and keeping track of submitted jobs and users.
- We showed the accountability policy specification so that only suitable information for accountability can be gathered.
- We conducted extensive experiments on a large set of nodes to overcome scalability issues and showed that our accountability system does not interfere with ordinary computations.