Sphinx: A Scheduling Middleware for Data Intensive Applications on a Grid

Richard Cavanaugh
University of Florida

Collaborators:
Janguk In, Sanjay Ranka, Paul Avery, Laukik Chitnis,
Gregory Graham (FNAL), Pradeep Padala, Rajendra Vippagunta,
Xing Yan

18.09.2003 Data Mining and Exploration Middleware for Distributed and Grid Computing – University of Minnesota
The Problem of Grid Scheduling

- Decentralised ownership
  - No one controls the grid
- Heterogeneous composition
  - Difficult to guarantee execution environments
- Dynamic availability of resources
  - Ubiquitous monitoring infrastructure needed
- Complex policies
  - Issues of trust
  - Lack of accounting infrastructure
  - May change with time

- Information gathering and processing is critical!
A Real Life Example

- Merge two grids into a single multi-VO “inter-grid”
- How to ensure that
  - neither VO is harmed?
  - both VOs actually benefit?
  - there are answers to questions like:
    - “With what probability will my job be scheduled and complete before my conference deadline?”
- Clear need for a scheduling middleware!
Some Requirements for Effective Grid Scheduling

- Information requirements
  - Past & future dependencies of the application
    - Persistent storage of workflows
  - Resource usage estimation
  - Policies
    - Expected to vary slowly over time
  - Global views of job descriptions
  - Request Tracking and Usage Statistics
    - State information important

- Resource Properties and Status
  - Expected to vary slowly with time
  - Grid weather
    - Latency measurement important
  - Replica management

- System requirements
  - Distributed, fault-tolerant scheduling
  - Customisability
  - Interoperability with other scheduling systems
  - Quality of Service
Incorporate Requirements into a Framework

- Assume the GriPhyN Virtual Data Toolkit:
  - Client (request/job submission)
    - Globus clients
    - Condor-G/DAGMan
    - Chimera Virtual Data System
  - Server (resource gatekeeper)
    - Globus services
    - RLS (Replica Location Service)
    - MonALISA Monitoring Service
    - etc
Incorporate Requirements into a Framework

- Framework design principles:
  - Information driven
  - Flexible client-server model
  - General, but pragmatic and simple
    - Implement now; learn; extend over time
  - Avoid adding middleware requirements on grid resources
    - Take what is offered!

- Assume the GriPhyN Virtual Data Toolkit:
  - Client (request/job submission)
    - Globus clients
    - Condor-G/DAGMan
    - Chimera Virtual Data System
  - Server (resource gatekeeper)
    - Globus services
    - RLS (Replica Location Service)
    - MonALISA Monitoring Service
    - etc
The Sphinx Framework

- **Sphinx Server**
  - Data Warehouse
  - Request Processing
  - Data Management
  - Information Gathering

- **Sphinx Client**
  - Chimera Virtual Data System
  - Condor-G/DAGMan

- **VDT Server Site**
  - Globus Resource
  - Replica Location Service
  - MonALISA Monitoring Service

- **VDT Client**
Sphinx Scheduling Server

- Functions as the Nerve Centre
- Data Warehouse
  - Policies, Account Information, Grid Weather, Resource Properties and Status, Request Tracking, Workflows, etc
- Control Process
  - Finite State Machine
    - Different modules modify jobs, graphs, workflows, etc and change their state
  - Flexible
  - Extensible

Data Warehouse

Sphinx Server

Control Process
- Message Interface
- Graph Reducer
- Job Predictor
- Graph Predictor
- Job Admission Control
- Graph Admission Control
- Graph Data Planner
- Job Execution Planner
- Graph Tracker

Data Management

Information Gatherer
Policy Constraints

- Defined by Resource Providers
  - Actual grid sites (resource centres)
  - VO management
- Applied to Request Submitters
  - VO, group, user, or even a proxy request (e.g. workflow)
- Valid over a Period of Time
  - Can be dynamic (e.g. periodic) or constant

- Global accounting and book-keeping is necessary
Quality of Service

- For grid computing to become economically viable, a Quality of Service is needed
  - “Can the grid possibly handle my request within my required time window?”
  - If not, why not? When might it be able to accommodate such a request?
  - If yes, with what probability?

- But, grid computing today typically:
  - Relies on a “greedy” job placement strategies
    - Works well in a resource rich (user poor) environment
    - Assumes no correlation between job placement choices
  - Provides no QoS
Quality of Service

- As a grid becomes resource limited,
  - QoS becomes even more important!
  - “greedy” strategies may not be a good choice
    - Strong correlation between job placement choices

- Sphinx is designed to provide QoS through time dependent, global views of
  - Requests (workflows, jobs, allocation, etc)
  - Policies
  - Resources
Resource Usage Estimation

- **User Requirements**
  - Upper limits on CPU, memory, storage, bandwidth usage

- **Domain Specific Knowledge**
  - Applications are often known to depend logarithmically, linearly, etc on certain input parameters, data size or type

- **Historical Estimates**
  - Record the performance of all applications
  - Statistically estimate resource usage within some confidence level
Data Management

- **Smart Replication:**
  - **Graph based**
    - Examine and insert replication nodes to minimise overall completion time
    - **Distribute** and **collect** required data
    - Particularly useful in data parallelism
  
  - **“Hot Spot” based**
    - Monitor current and historical data access patterns and replicate to optimise future access
Data Management

- Smart Replication:
  - Graph based
    - Examine and insert replication nodes to minimise overall completion time
    - Distribute and collect required data
    - Particularly useful in data parallelism
  - “Hot Spot” based
    - Monitor current and historical data access patterns and replicate to optimise future access
Early Sphinx Prototype Test Results

- Simple sanity checks
  - 120 canonical virtual data workflows submitted to US-CMS Grid
- Round-robin strategy
  - Equally distribute work to all sites
- Upper-limit strategy
  - Makes use of global information (site capacity)
  - Throttle jobs using just-in-time planning
  - 40% better throughput (given grid topology)

- Conclusion: Prototype is working!
Some Current and Future Activities

- Policy Based Scheduling
- Quality of Service
- Graph Partitioning
- Data Parallelism
- Prediction Module
- Useful Views and Fusion of Monitoring Data
Conclusions

- Scheduling on a grid has unique requirements
  - Information
  - System

- Decisions based on global views providing a Quality of Service are important
  - Particularly in a resource limited environment

- Sphinx is an extensible, flexible grid middleware which
  - Already implements many required features for effective global scheduling
  - Provides an excellent “workbench” for future activities!