Resource Management (2)

Grid Computing (M)

Lecture 12

UNIVERSITY
of
GLASGOW

Lecture Outline

- Job scheduling and management
  - Outline technical problems
  - Effects of autonomy
- Load balancing
  - Resource provision and advertisement
  - Static load balancing
  - Dynamic load balancing
  - Job migration
  - Execution environments
  - Examples

Job Scheduling and Management in a Grid

- Grid computing applications typically follow two patterns:
  - Distributed computation
    - Exploring a large parameter space, repeating a task across a data set
      - Large amounts of data, enormous need for computational cycles
      - Master-worker model
      - Embarrassingly parallel applications
    - SETI@home, particle physics, bioinformatics, movie rendering
  - Remote resource access
    - Coordinating execution of small number of tasks, running on distributed resources managed by diverse organizations
    - Remote instrument access (scanning electron microscopes, sensors, etc.), combining large database queries

    } Much less uncommon

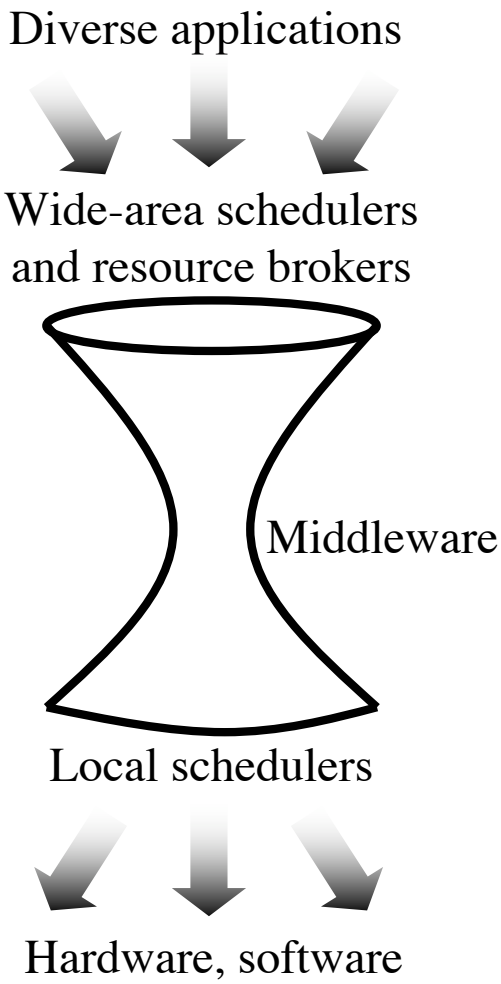
- Need grid-aware scheduling algorithms to coordinate execution of jobs across multiple sites
  - Mostly large scale distributed computation

The Job Scheduling Problem

- Job scheduling and management in a Grid is a wide-area scheduling and resource management problem
  - Discover remote resources
  - Schedule jobs on resources
  - Distribute the code and data
  - Execute jobs
  - Collect and collate results

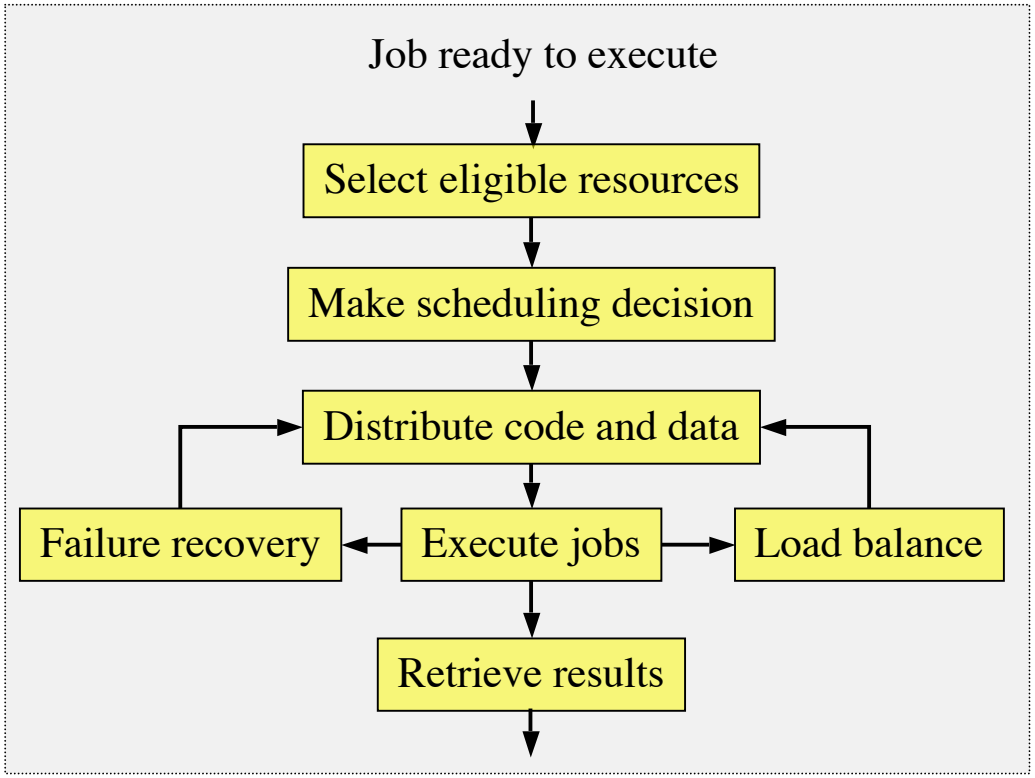
- Desirable to hide details of the local infrastructure at each site from the wide-area scheduler
  - Decouple implementations

- Use an hourglass model with middleware to isolate the wide-area scheduler from the local schedulers



The Scheduling Problem

- How to schedule jobs to run in parallel across a grid?
  - Where to run jobs:
    - Real time vs. non-real time
    - Batch vs. interactive
    - Load balancing
    - Need for specific resources
  - Co-allocating related jobs
  - Execution environment
  - Failure tolerance



- A standard scheduling problem...?

Effects of autonomy

- Politics and accounting
  - Who pays? Who is allowed access? Why are resources shared?
  - How to discover a resource sharing community exists? How to become a member? What are the benefits and costs of membership?
- Additional technical issues:
  - Load balancing
  - Job migration
  - Fault tolerance

- Why are these issues?
  - Things may go wrong... for political, technical or financial reasons, and force you to move a job
  - You may wish to spread your risk or your costs
  - You may wish to optimise performance

  (all three assume resources are plentiful, possible to move jobs)

Load Balancing

- What is the problem?
  - Assign jobs to resources (e.g. processors) such that no resource overloaded
  - Redistribute jobs in the event of failure of a resource

- Issues
  - How to identify and describe resources?
  - How to identify current load on a resource?
  - How to predict future load on a resource?
  - How to identify patterns of resource usage?
  - How to distribute or migrate jobs?

Question: how does organisational autonomy change the issues?

Resource Provision

- As a resource provider, how much resource should you provide?
  - Start from high level measures:
    - Transactions per unit time, peak vs. sustained rate, etc.
    - Political need to contribute, economics
  - Work down to low level measures:
    - CPU cycles, memory, disk, etc.
  - Rules of thumb, informed by prior measurement and economics

- How to do capacity planning?
  - What are the revenue and growth models?
  - What are the constraints on growth and/or revenue?
  - Economics and politics again...

Resource Description and Advertisement

- How to advertise a resource and its load?
  - Ontology for resource description
  - What information to report?
    - Low level metrics
      - CPU, disk, memory usage, etc.
    - High level metrics
      - % peak capacity, mean waiting time, etc.

    } Is the metric reported the same as that used in the planning process?
  - How much detail to report?
    - To what extent is it useful *from a business perspective* to report only incomplete or aggregate metrics?
    - What timescales to measure and to report over? Are they the same?
  - How to advertise?
    - Publish/subscribe
    - Polling and caching
    - Gossip and peer-to-peer protocols

Job Placement

- How to decide a good placement?
  - Are the needed resources only available in one location?
    - Place at that location
  - Does the job need many resources? (compute cycles, memory, disk)
    - Place on node with plentiful resources
  - Does the job need to communicate? (communication patterns)
    - Place near communication peers

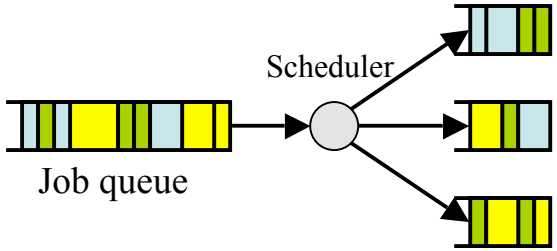
- Static or dynamic placement?
  - Static: simpler, but can't easily correct mistakes/system imbalance
  - Dynamic: more complex, migration overheads, ability to rebalance system

Static Job Placement

- Typical supercomputing mechanism:
  - Large computational jobs submitted to a batch queue
    - Central queue runner task
  - Queue serviced according to some constraints
    - FIFO queue
    - Priority queue
    - Weighted fair queuing
  - Job at head of queue assigned to resource for execution
    - Large cluster of worker nodes

- Makes a single scheduling decision
  - Use knowledge of static job parameters; current and predicted state of the system load
  - Does not move jobs after assigned to resource
    - Requires predictable job pattern to ensure load balancing

Static Job Placement: Queuing Models



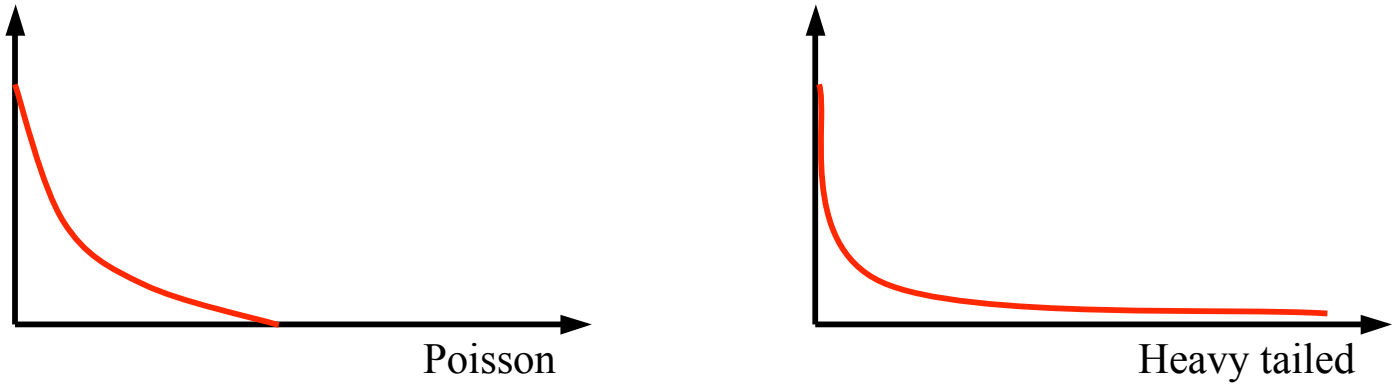
Jobs executing on several resources

Queuing discipline?
Execution speed?
Distribution of job arrivals?
Distribution of job sizes?

- Model static scheduler as a simple queuing system
- Many well-formed analytic models for optimal performance
  - Balance load across queues
  - Optimise throughput
  - Predict waiting time
- But: need statistical distribution of job sizes and inter-arrival times; queue service discipline; speed at which jobs execute
  - E.g. Poisson arrival processes likely appropriate for many large systems

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

Static Job Placement: Heavy Tails



- Unfortunately job sizes, execution times likely heavy tailed
  - E.g. Pareto distribution
  - Most jobs are small, but most of the work in the large jobs
    - E.g. one study saw 80% of bytes in FTP traffic in largest 2% of transfers
  - Makes analytic queuing models difficult
    - E.g. mean execution time not typical; maths gets hard!
    - Expectation paradox: longer we have executed, longer likely to execute

- Disrupts naïve models; few systems implement complex models

Dynamic Job Placement

- Monitor dynamic system load; migrate jobs to rebalance if needed
  - How to monitor?
    - Within a cluster:
      - Periodic polling of system load
      - Alerts on node overload
      - Failure checks for fault tolerance
    - Across autonomous sites:
      - Much higher monitoring overhead
      - Sites may not allow monitoring, or provide only restricted information
  - How to migrate?
    - Need to move a running process from one node to another
    - Likely an expensive operation

- *Question: is dynamic migration needed if nodes are dedicated?*

Dynamic Job Placement

- Heavy tailed distributions can help dynamic migration:
  - Expectation paradox: longer job has executed, longer it is likely to execute
  - Most of the weight is in a few large jobs

- Implications:
  - If a node is unbalanced and has jobs which have just started, assume will finish soon ⇒ static load balancing will solve the imbalance
  - If a node is unbalanced and has long running jobs, assume will run for much longer ⇒ dynamic load balancing + migration appropriate

- If job sizes known to be heavy tailed, gives simple predictor to mitigate overheads of job migration

Job Migration

- To perform dynamic load balancing, must be able to migrate jobs between nodes

- What type of jobs?
  - Native processes (e.g. Unix processes)
  - Virtual machine programs (e.g. Java, Python, etc.)

- Also any resources on which the jobs depend
  - Data files (potentially very large)
  - Shared memory, locks, semaphores, mutexes, etc
  - Network connections and IPC channels
  - Shared library code (*.dll, *.so, Java class libraries, etc.)

Job Migration: Cutover Strategies

- How to move a job between systems?
  - Checkpoint and eager cutover
    - Freeze job, copy entire address space + other resources, migrate thread of control, continue
    - High initial cost, large pause during cutover
  - Copy on reference, lazy cutover
    - Freeze job, migrate minimal state and thread of control, continue and fault in data + other resources when needed
    - Low initial costs, high runtime overhead during lazy cutover
  - Pre-copy with lazy cutover
    - Copy entire address space + other resources while job running, freeze job, migrate thread of control, continue and fault in dirty data + other resources when needed
    - High initial and runtime costs, but very short "freeze" time during cutover

(each requires varying degree of operating system support)

Job Migration: Transparency

- Is a job aware that it's being migrated?
  - With a suitable virtual machine to hide machine dependencies, and an RMI implementation, might be possible to migrate some objects transparently to the remainder of the system
    - *Question: is this possible with Java?*
  - It *is* possible to checkpoint and transparently migrate standalone Unix or Linux processes
    - *Question: how to checkpoint a Unix process? why the restriction to checkpoint only standalone processes?*

- Unless using checkpoint and eager cutover, will likely need to proxies for objects, files, etc.
  - Performance hit, but may be acceptable

Execution environments

- Migrating jobs need a predictable execution environment
  - Hardware differences: virtual machine languages like Java, C#, etc.
  - Surprisingly hard to make a binary that'll execute across different versions of an operating system; need standard environment
    - Binary compatibility and library versioning issues
  - How to ensure code and data integrity and confidentiality?
    - Don't necessarily trust the host on which a job is running…
    - Trusted computing environments *very* helpful here!

- Jobs need to access data and store results
  - Cannot assume a shared file system
  - Does the scheduler automatically distribute input files and collect results? Is this done manually as part of the job?
  - Does the job have access to a file system? All or part of it?
    - How does constraining access to the file system affect ability to load shared libraries? (e.g. the **`jail`** facility on FreeBSD)

Example: Condor

- Network batch queuing system for clusters and cycle-stealing
  - Jobs have dispatch priority
  - Facilities for ordered jobs and master-worker operation
- Automatically distributes code, data and retrieves results
  - Uses trivial ClassAds to match jobs to resources
    - Processor, memory, disk, operating system, programming language, owner
  - **Static assignment of jobs to resources**
- Robustness via checkpoint and recovery
  - Requires re-linking against a modified **`libc`**; lots of space for checkpoints
  - Robust, but imposes considerable constraints
    - No IPC; no alarms, timers or sleeping; single threaded; no **`mmap()`**; file locking unreliable; files can't be opened read/write; most platforms need static linking
- Limited security and sandboxing; trusted environment

Example: Other Systems

- OpenPBS http://www.openpbs.org/
- Sun N1 Grid Engine http://gridengine.sunsource.net/
- Xgrid http://www.apple.com/acg/xgrid/

More limited… generally perform job scheduling; leave job management, data distribution, fault tolerance to the user

---

- In general:
  - Security, authentication, authorisation and accounting neglected
  - No real facilities for real-time jobs, resource reservation
  - Limited robustness and distributed coordination facilities

- *Question: why are real-world implementations so limited?*

Summary

- Static load balancing fundamentally a solved problem
  - Well developed theory, mathematical models
  - Many robust implementations
    - Could improve performance by leveraging more theory
- Dynamic load balancing difficult
  - Basic problems solved; open issues remain
  - Few implementations

- Grid computing systems currently addressing only limited scope problems; no need for complex solutions
  - Likely to change in future: growth of parallelism