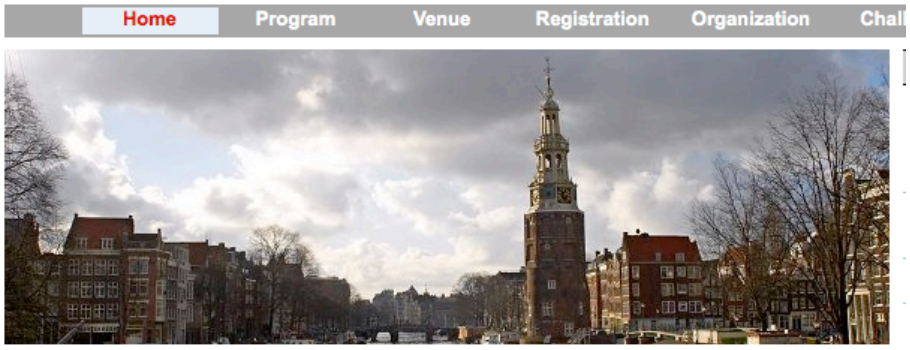
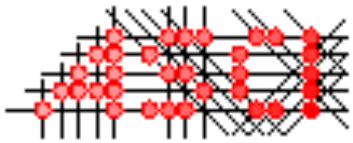
Scheduling and Resource Management in Grids and Clouds

Dick Epema
Parallel and Distributed Systems Group
Delft University of Technology
Delft, the Netherlands



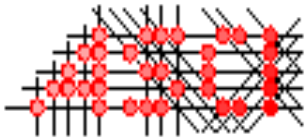
may 10, 2011



TU Delft
Delft University of Technology

Outline

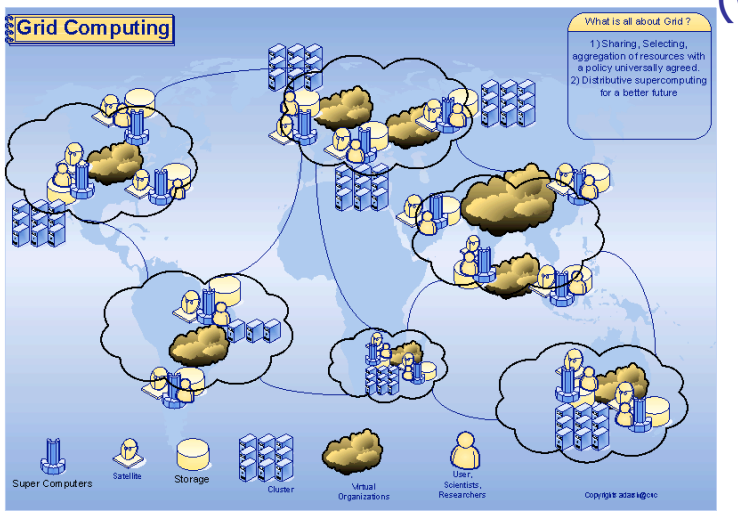
- Resource Management
  - Introduction
  - A framework for resource management
- Grid Scheduling
  - Problems in grid scheduling
  - Stages in grid scheduling
  - Service level agreements
  - Scheduling policies
- Examples of Grid Resource Management Systems
- Cloud Resource Management

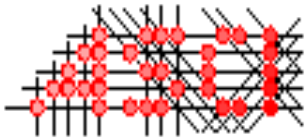


Resource Management (1)

- "***Resource management*** *refers to the operations used to control how capabilities provided by grid resources and services are made available to other entities, whether users, applications or services*"

(Czajkowski, Foster, Kesselman)





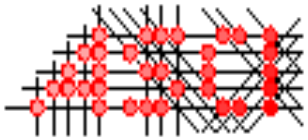
TU Delft

Resource Management (2)

- **Main goal**: establish a **mutual agreement** between resource providers and resource consumers

- **Grid Resource Consumers**
  - Execute jobs for solving problems of various size and complexity
  - Benefit by judicious selection and aggregation of resources

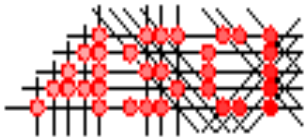
- **Grid Resource Providers**
  - Contribute ("idle") resources for executing consumer jobs
  - Benefit by maximizing resource utilization



TU Delft

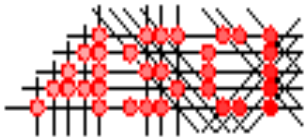
Resource Characteristics in Grids (1)

- A **resource** denotes any capability that may be shared and exploited in a networked environment
  - e.g., processor, data, disk space, bandwidth, telescope, etc.
- **Autonomous**
  - Each resource has its own management policy or scheduling mechanism
  - No central control/multi-organizational sets of resources
- **Heterogeneous**
  - Hardware (processor architectures, disks, network)
  - Basic software (OS, libraries)
  - Grid software (middleware)
  - Systems management (security set-up, runtime limits)
- **Compute power is a complex utility!!!**



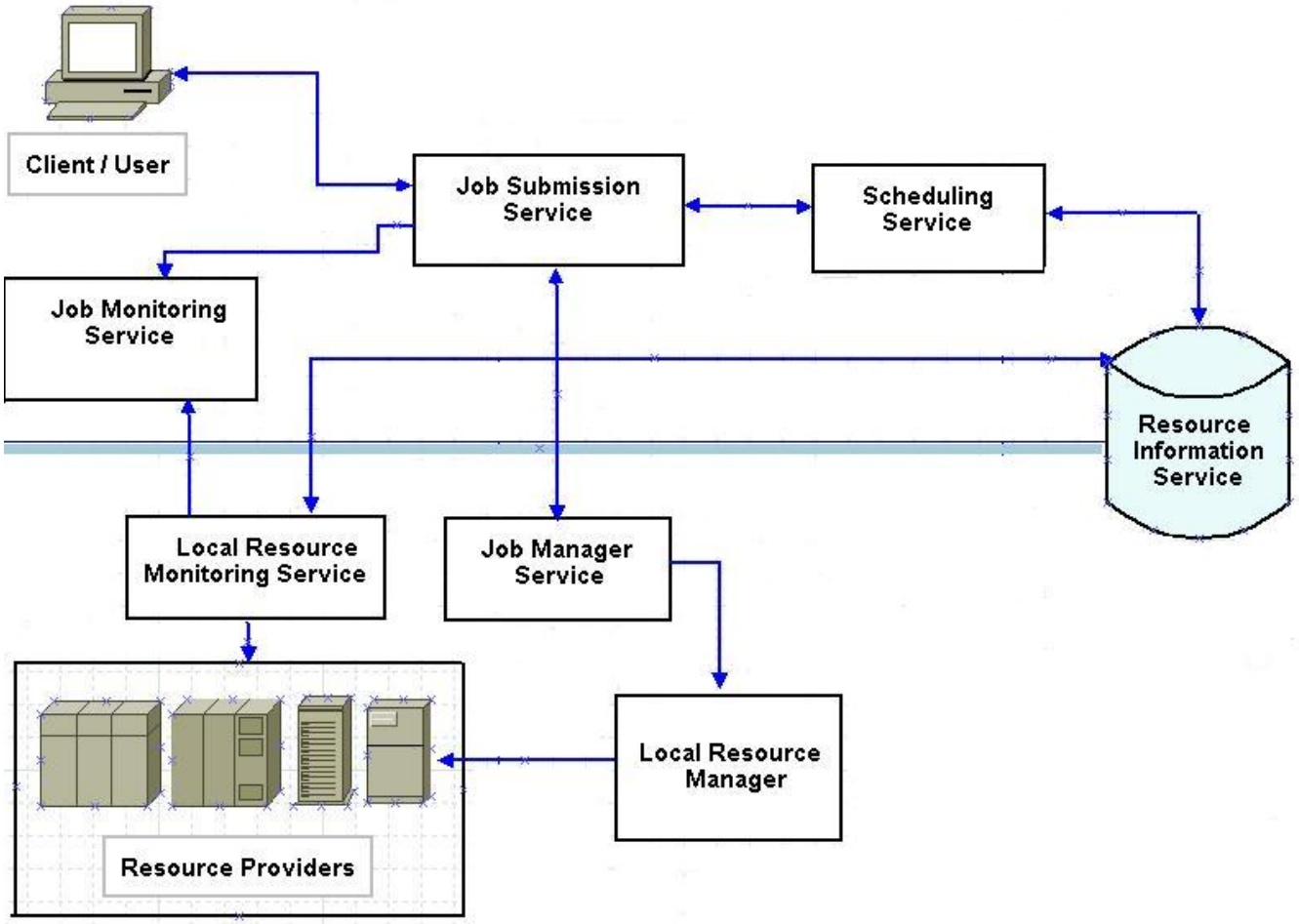
Resource Characteristics in Grids (2)

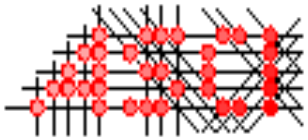
- **Size (up to exascale computing!)**
  - Large numbers of nodes, providers, consumers
  - Large amounts of data
- **Varying Availability**
  - Resources can join or leave to the grid at any time due to maintenance, policy reasons, and failures
- **Geographic distribution and different time zones**
  - Resources may be dispersed across continents
- **Insecure and unreliable environment**
  - Prone to various types of attacks



TU Delft

A General Resource Management Framework (1)



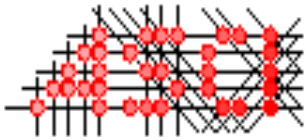


TU Delft

A General Resource Management Framework (2)

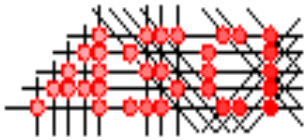
- **Local resource management system (Resource Level)**
  - Basic resource management unit
  - Provides low level resource allocation and scheduling
  - e.g., PBS, SGE, LSF

- **Global resource management system (Grid Level)**
  - Coordinates all local resource management systems within multiple or distributed sites
  - Provides high-level functionalities to efficiently use resources:
    - Job submission
    - Resource discovery and selection
    - Authentication, authorization, accounting
    - Scheduling (possibly including co-allocation)
    - Job monitoring
    - Fault tolerance



Co-Allocation (1)

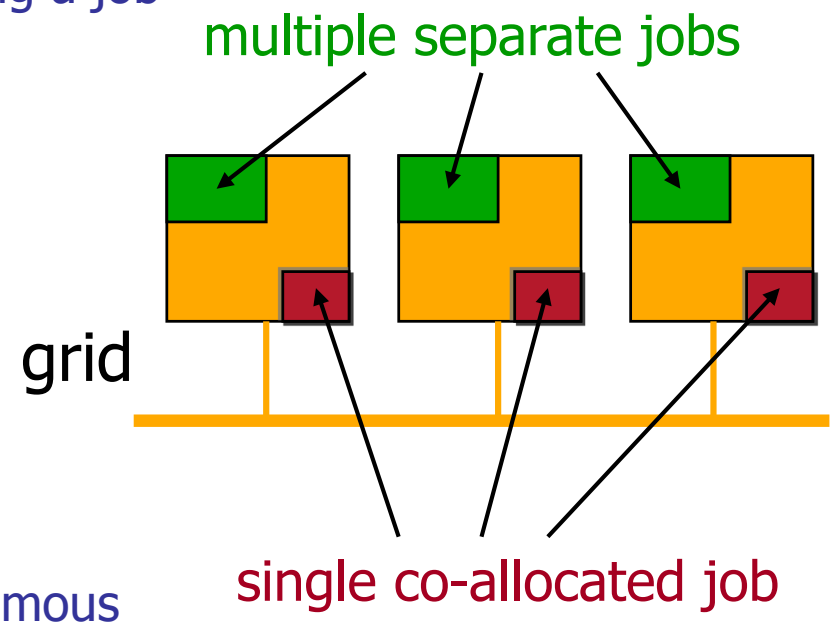
- In grids, jobs may use multiple types of resources in multiple sites: **co-allocation** or **multi-site operation**
- **Reasons:**
  - To use available resources (e.g., processors)
  - To access and/or process geographically spread data
  - Application characteristics (e.g., simulation in one location, visualization in another)
- Resource possession **in different sites** can be:
  - Simultaneous (e.g., parallel applications)
  - Coordinated (e.g., workflows)

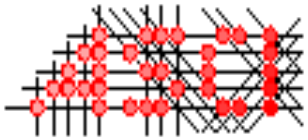


Co-Allocation (2)

- **Without co-allocation**, a grid is just a big load-sharing device with a **metascheduler** or **superscheduler**:
  - Find suitable candidate system for running a job
  - If the candidate is not suitable anymore, migrate

- **With co-allocation**:
  - Better utilization of resources
  - Better response times
  - More difficult resource-discovery process
  - Need to coordinate allocations by autonomous resource managers (**local schedulers**)

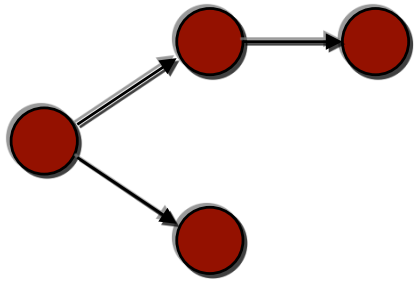


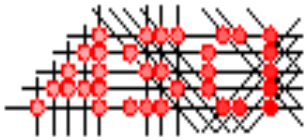


TU Delft

Job types in Grids

- **Sequential jobs/bags-of-tasks**
  - E.g., parameter sweep applications (PSAs)
  - These account for the vast majority of jobs of grids
  - Leads to **high-throughput** computing
- **Parallel jobs**
  - Extension of **high-performance** computing
  - In most grids, only a small fraction
- **Workflows**
  - Multi-stage data filtering, etc
  - Represented by directed (acyclic) graphs
- **Miscellaneous**
  - Interactive simulations
  - Data-intensive applications
  - …

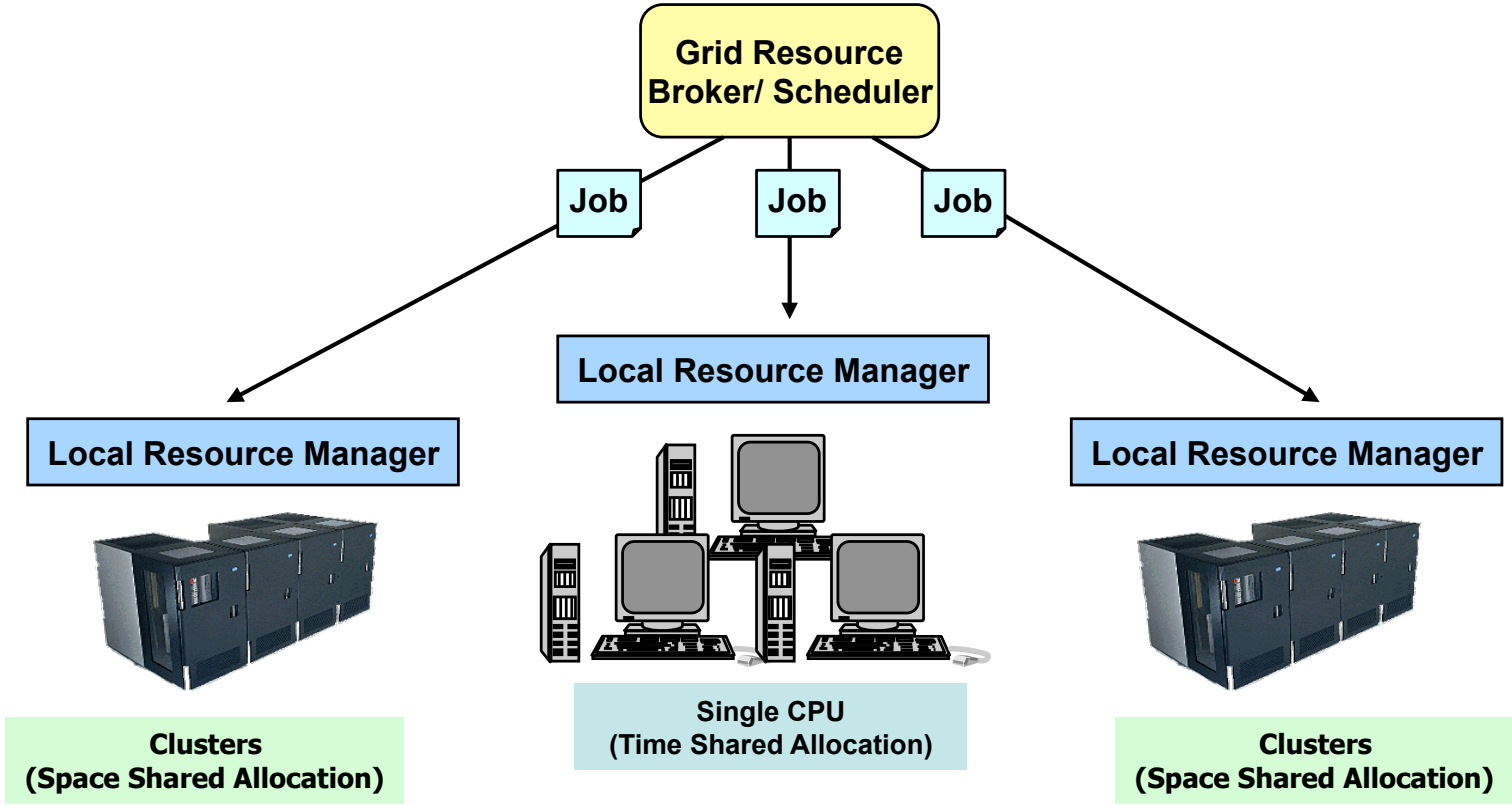


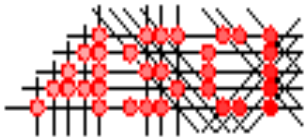


TU Delft

How to select resources in the grid?

- **Grid scheduling** is the process of assigning jobs to grid resources

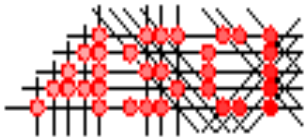




TU Delft

Different Levels of Scheduling

- **Resource level scheduler**
  - *a.k.a.* low-level scheduler, local scheduler, local resource manager
  - Scheduler controlling a supercomputer or a cluster
  - e.g.: Open PBS, PBS Pro, LSF, SGE
- **Enterprise level scheduler**
  - Scheduling across multiple local schedulers belonging to the same organization
  - e.g., PBS-Pro, LSF multi-cluster
- **Grid level scheduler**
  - *a.k.a.* super-scheduler, broker, community scheduler
  - Discovers resources that can meet a job's requirements
  - Schedules across lower level schedulers
- **System-level versus application-level scheduling**

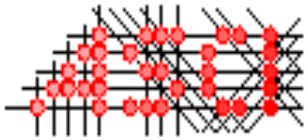


TU Delft

Grid Level Scheduler

- Discovers and selects the appropriate resources for a job
- No ownership or control over resources
- Jobs are submitted to local resource managers
- Local resource managers take care of actual execution of jobs

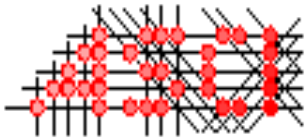
- **Architecture**
  - **Centralized**: all lower level schedulers are under the control of a single grid scheduler
    - not realistic in global grids
  - **Distributed**: lower level schedulers are under the control of several grid schedulers



TU Delft

General Problems in Grid Scheduling

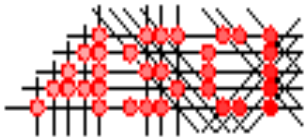
1. **Grid schedulers do not own resources themselves**
   - they have to negotiate with autonomous local schedulers
   - authentication/multi-organizational issues
2. **Grid schedulers have to interface to different local schedulers**
   - some may have support for reservations, others are queuing-based
   - some may support checkpointing, migration, etc
3. **Structure of applications**
   - many different structures (parallel, PSAs, workflows, etc.)
   - for co-allocation, optimized wide-area communications libraries are needed



TU Delft

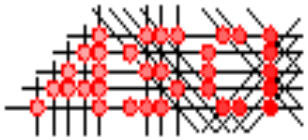
Problems (1): system

1. **Lack of a reservation mechanism**
   - but with such a mechanism we need good runtime estimates
2. **Heterogeneity**
   - hardware (processor architecture, disk space, network)
   - basic software (OS, libraries)
   - grid software (cluster scheduler)
   - systems management (security set-up, runtime limits)
3. **Failures**
   - monitor the progress of applications/sanity of systems
   - only thing we know to do upon failures: (move and) restart
   - we may interface to fault-tolerance mechanisms in applications



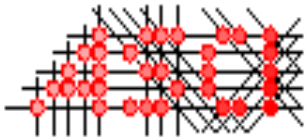
Problems (2): applications

4. **Communication in wide-area applications**
   - may need an additional initialization step to allow applications to communicate across multiple subsystems
5. **Structure of applications**
   - many different structures, can't deal with all of them
   - introduce application adaptation modules
   - allow application-level scheduling



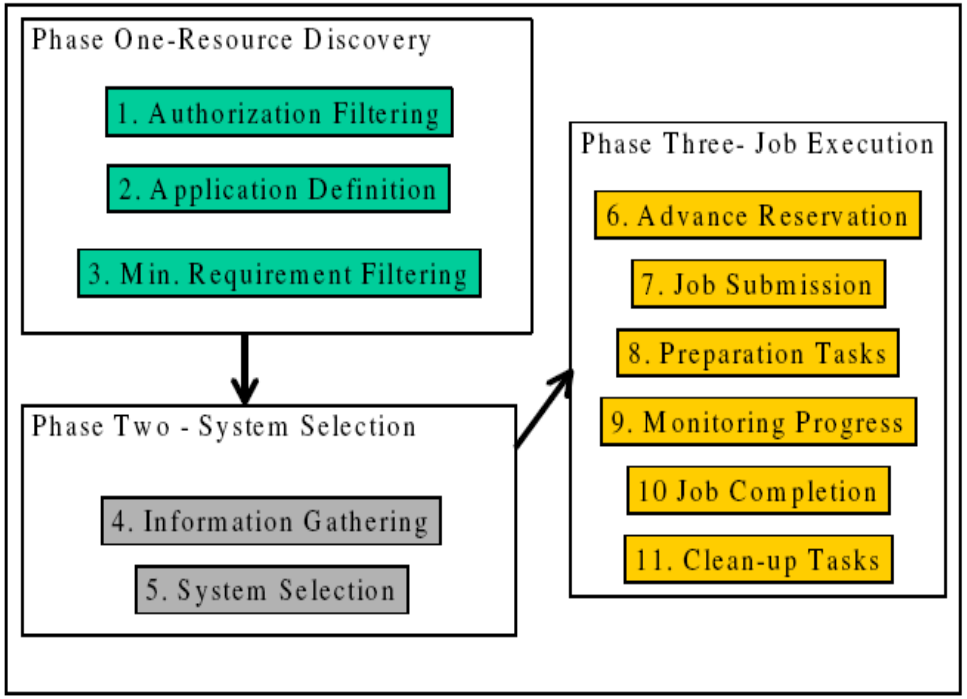
Problems (3): testing/performance

6. **Need for a suite of test applications**
   - for functionality and reliability testing
   - for performance evaluation
   - should run on "all grids"
7. **Reproducibility of performance experiments**
   - never the same circumstances
   - tools for mimicking same conditions

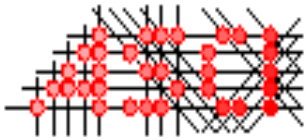


Stages of Grid Scheduling

1. **Resource Discovery**
2. **System Selection**
3. **Job Execution**



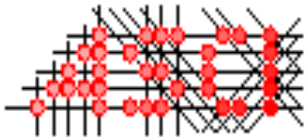
www.mcs.anl.gov/~jms/Pubs/sched.arch.2002.pdf



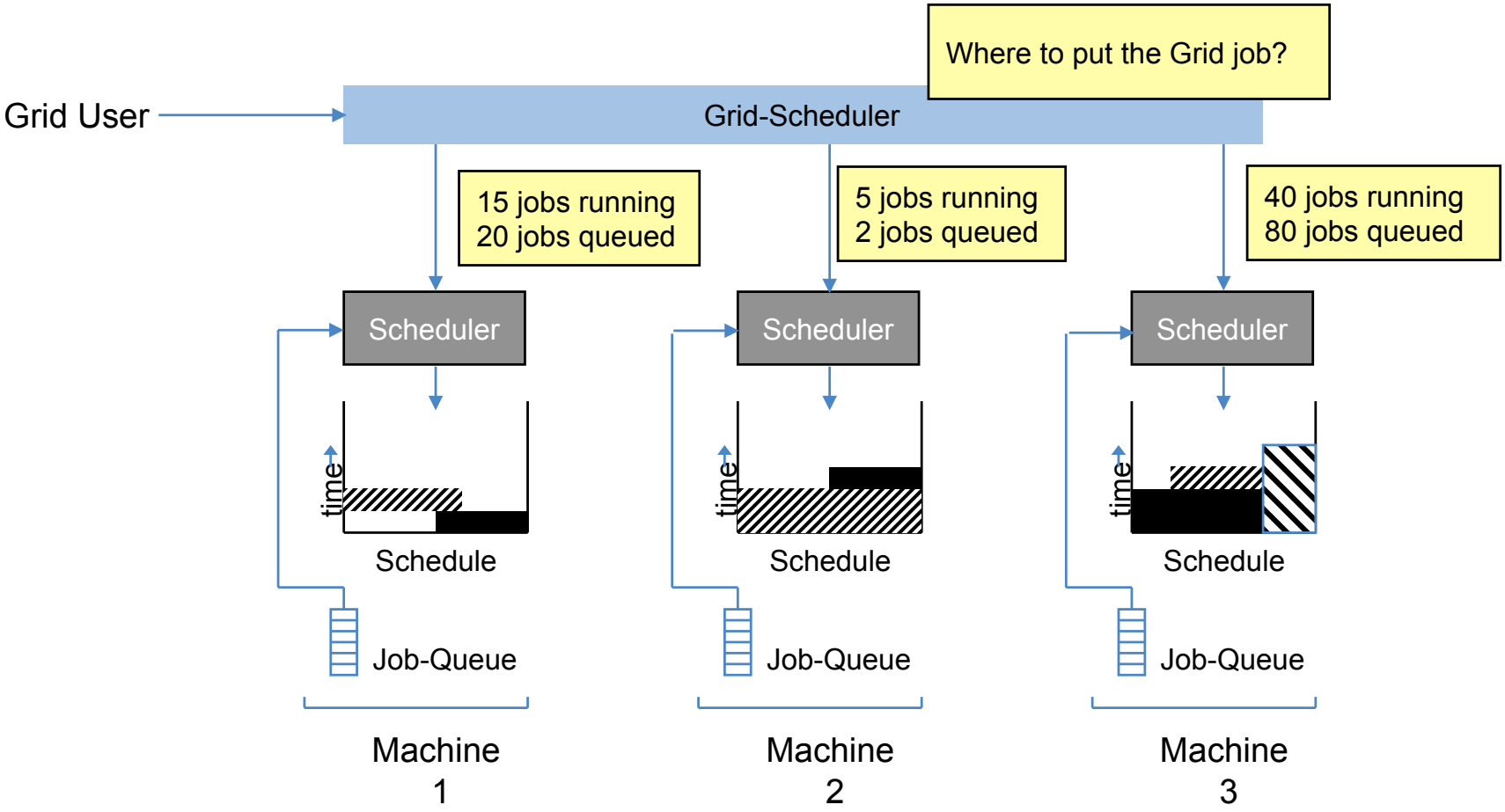
TU Delft

Resource Discovery

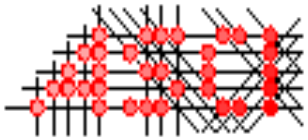
- **Describe jobs and resources:**
  - Job description files
  - Classads
- Determine which resources are available to the user:
  - **Authorization Filtering**
    - A list of machines or resources to which the user has access
  - **Application Requirement Definition**
    - Job requirements can be specified with job description languages (e.g., Condor ClassAd, GGF JSDL, Globus RSL)
  - **Minimal Requirement Filtering**
    - A reduced set of resources to investigate in more detail



System Selection (1)



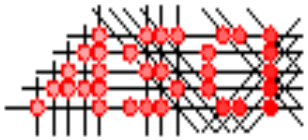
From: Grid Resource Management and Scheduling, *Ramin Yahyapour*



TU Delft

System Selection (2): Information Gathering

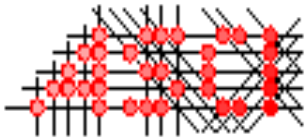
- **Grid Information Service (GIS)**
  - Gathers information from individual local resources
  - Access to static and dynamic information
  - Dynamic information include data about planned or forecasted future events
    - e.g., existing reservations, scheduled tasks, future availabilities
  - Create a database for the GIS



System Selection (3): Methods

- **Predictive state estimation**
  - Consider current and **historical** information
  - Heuristics
  - Pricing Models
  - Machine Learning

- **Non-predictive state estimation**
  - Consider only the **current** job and resource status information
  - No need to take into account the historical information
  - Heuristics



TU Delft

System Selection (4): Objectives

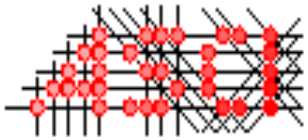
- **Application Oriented**
  - Optimizing metrics such as job response time, job wait time, etc.
  - e.g., average weighted response time

$$AWRT = \frac{\sum_{j \in Jobs} w_j \cdot (t_j - r_j)}{\sum_{j \in Jobs} w_j}$$

- r : submission time of a job
- t : completion time of a job
- w : weight/priority of a job

- **System Oriented**
  - Maximizing throughput, utilization (minimizing idle time)

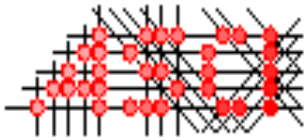
- **Conflicting objectives!!**
  - The (main) scheduling criterion is usually static for an installation and implicitly given by the scheduling algorithm



TU Delft

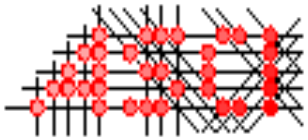
Job Execution (1)

- Advance Reservation
- Job Submission
- Preparation Tasks
- Monitoring Progress
- Job Completion
- Cleanup Tasks



Job Execution (2): Advance Reservations

- Co-allocated and other applications require a priori information about the precise resource availability
- Resource provider guarantees a specified resource allocation
  - two- or three-phase commit included for agreeing on the reservation
- **Implementations**
  - GARA/DUROC provide interfaces for Globus to create advanced reservation
  - Setup of a dedicated bandwidth between endpoints (e.g., DAS3/ StarPlane)



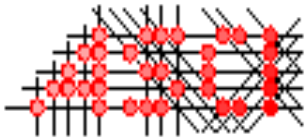
TU Delft

Job Execution (3): Job Submission

- It is the task of the resource management system to start a job on the allocated resources

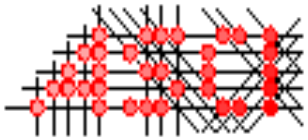
- The user 'submits' the job to the resource management system:
  - qsub jobscript.pbs
  - globus-submit-job job.rsl

- The user can control the job
  - *qsub*: submit
  - *qstat*: poll status information
  - *qdel*: cancel job



Job Execution (4): Other Phases

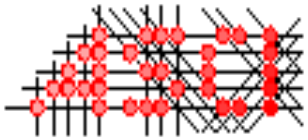
- **Preparation Tasks**
  - File staging, claiming a reservation
- **Monitoring Progress**
  - Resource conditions
  - Agreements
  - Schedules
  - Program execution
  - SLA conformance
- **Job Completion**
  - Notification
- **Cleanup Tasks**
  - File retrieval
  - Removing temporary settings



Service Level Agreements

- Old concept from the mainframe days

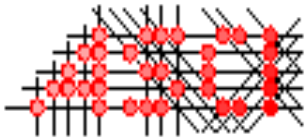
- The mapping of jobs to resources can be abstracted using the concept of Service Level Agreement (**SLAs**)
- **SLA**: Contract negotiated between
  - resource provider, e.g., local scheduler, and
  - resource consumer, e.g., grid scheduler, application
- **SLAs** provide a uniform approach for the client to
  - specify resource and QoS requirements, while
  - hiding from the client details about the resources,
  - such as queue names and current workload



TU Delft

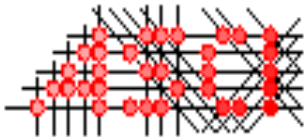
Scheduling and Replication Algorithms (1)

- **Random**
  - A randomly selected site
  - Each site has an equal or a different probability of getting selected
- **LeastLoaded**
  - The site that currently has the least load
  - A possible load definition: smallest number of waiting jobs
- **RandLeastLoaded**
  - A site randomly selected from a number of the least-loaded sites
- **DataPresent**
  - A site that already has the required data
- **Local**
  - The site where the job originated



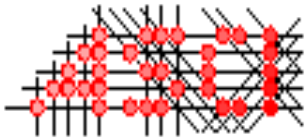
Scheduling and Replication Algorithms (2)

- **DataCaching**
  - Datasets are pre-assigned to different sites
  - No dynamic replication is in place
- **DataRandom**
  - Sites keep track of the popularity of the datasets they contain
  - When a site's load exceeds a threshold, "popular datasets" are replicated to a random site on the grid
- **DataLeastLoaded**
  - Sites keep track of dataset popularity and choose the least loaded site as a new host for a popular dataset
- **DataRandLeastLoaded**
  - A random site is picked from a number of the least loaded sites



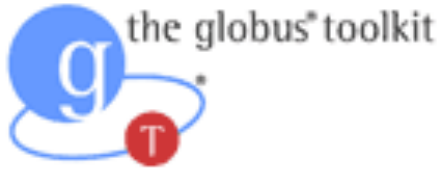
Scheduling and Replication Algorithms (3)

- Scheduling in grids resembles **scheduling in distributed systems** (1980s and 1990s)
- **Differences with grids:**
  - In distributed systems, usually, a **centralized approach** was assumed
  - In distributed systems, the system was usually assumed to be **homogeneous**
- **Important research issues** in scheduling in distributed systems:
  - Amount of state information used (global FCFS, global SJF, …)
  - Staleness of state information used
- **Main lessons from distributed systems:**
  - Use simple policies
  - Use policies that don't use much state information



TU Delft

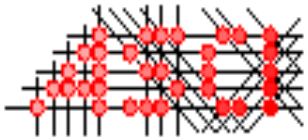
Example 1: The Globus Toolkit



- **Widely used grid middleware**
  - Negotiate and control access to resources
  - GRAM component for control of local execution

- **Globus toolkit includes software services and libraries for:**
  - resource monitoring
  - resource discovery and management
  - security
  - file management

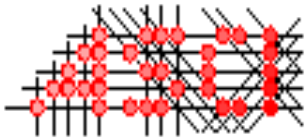
- GT2
- GT3 – Web/GridService-based
- GT4 – WSRF-based



TU Delft

Example 2: Condor (1)

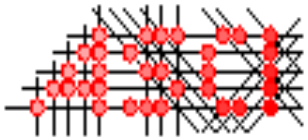
- Specialized **high-throughput** scheduling system
- Started around 1986 as one of many **batch queuing systems** for clusters (of desktop machines)
- Provides a queuing mechanism, scheduling policies, priority scheme, resource monitoring
- **Cycle scavenging**: can be set to use idle time on machines
- Condor introduced the notion of **classads**
- Cross-platform: UNIX/Linux, MacOS X, Windows
- **Condor-G(lide-in):**
  - used for submitting, queuing and monitoring jobs on Globus-managed resources
  - makes a Globus-managed resource appear as part of a Condor pool



TU Delft

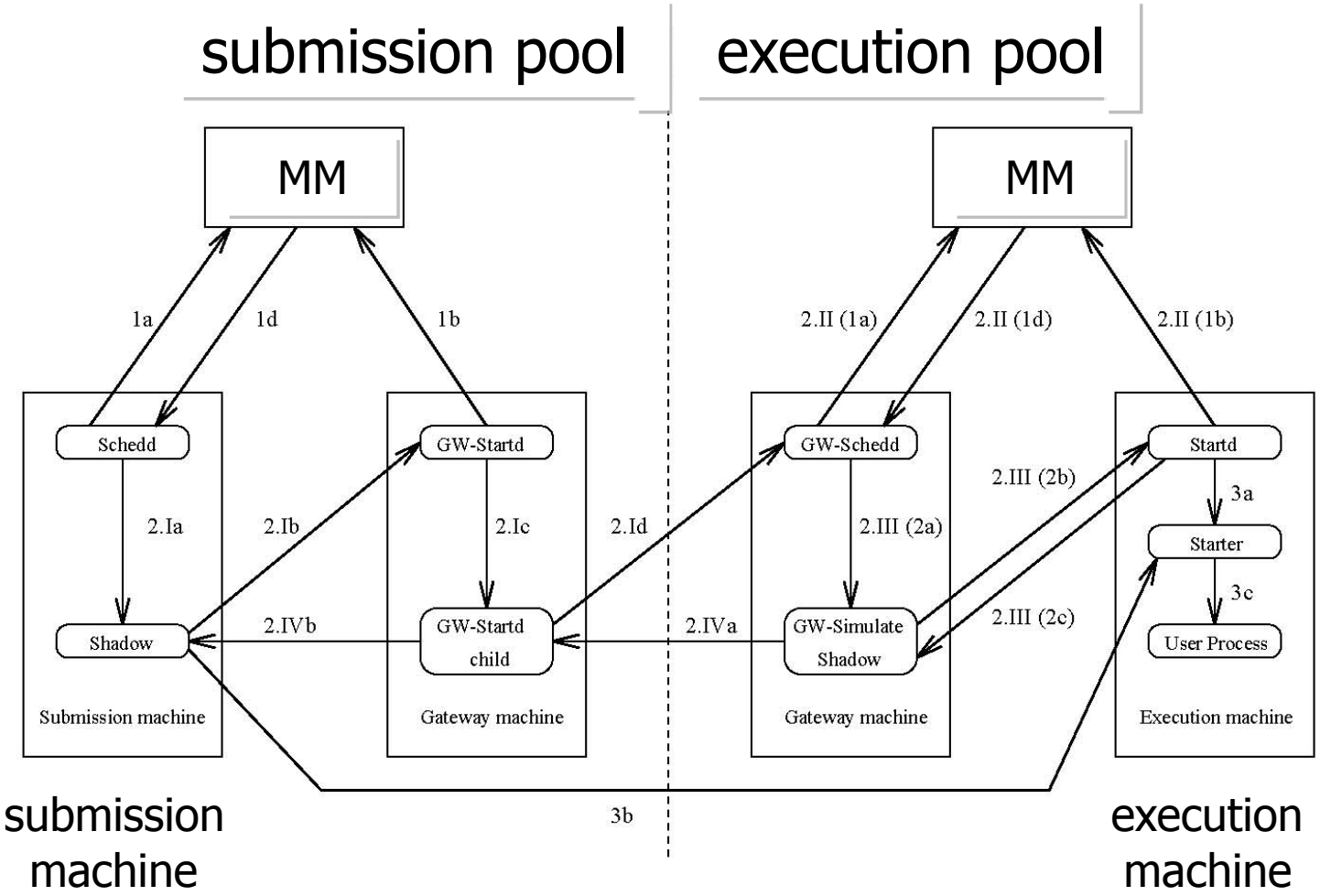
Example 2: Condor (2)

- **Basic operation of Condor:**
  - **jobs** send a **classad** to the **matchmaker**
  - **machines** send a **classad** to the **matchmaker**
  - the matchmaker **matches** jobs and machines,
  - and notifies the **submission machine**,
  - which contacts the **execution machine**
  - on the submission machine, a **shadow** process is started, which represents the remote job on the execution machine
  - on the execution machine, the actual **remote user job** is started



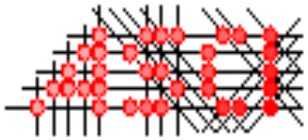
TU Delft

Example 2: Condor Flocking



Clouds (1): properties

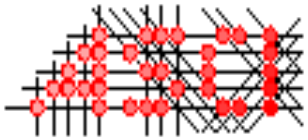
- **(Large-scale) data centers**
  - Offer compute, storage, and network resources
- **Pay per use**
  - No advance commitment, utility prices
- **Elastic capacity**
  - Scale up/down on demand (time granularity 1 hour)
- **Self-service interface**
  - Access through web services
- **Resources are abstracted/virtualized**
  - Virtualization technology
  - Provides for protection between users and for easy resource management



Clouds (2): types

- **By access:**
  - Public/private clouds

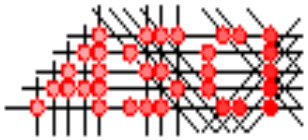
- **By level:**
  1. **Infrastructure As A Service** (IAAS):
     - basic computing infrastructure (processors, storage, network, etc)
     - resources are machine images
     - example: Amazon Elastic Compute Cloud (EC2)
  2. **Platform As A Service** (PAAS):
     - programming environment
     - execution environment
     - example: Google App Engine
  3. **Software As A Service** (SAAS):
     - application (service)s
     - example: Google Docs



TU Delft

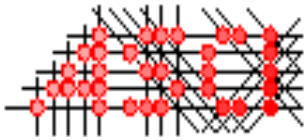
Clouds (3): Amazon EC2 features

- **SLA with 99.95% uptime**
- **Availability zones**
- **Types of "instances"**
  - **On-demand** instances (hourly basis)
  - **Reserved** instances (1—3 years)
  - **Spot** instances (specify maximum hourly price)
- **Elastic load balancing**
  - Distributes incoming application traffic across multiple EC2 instances and availability zones
- **EC2 for HPC:**
  - Cluster Compute instances (up to 128)
  - Guaranteed low-latency 10 Gbps interconnections
  - Single instance type (quad core, 23 GB MM, 1.7 TB storage)
  - Cluster GPU instances with NVIDIA Tesla GPU



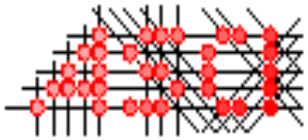
Clouds (4): IAAS systems

- **Eucalyptus**
  - Open source
  - EC2 compatible
  - Spin-off at UCSB (Rich Wolski)
- **Open Nebula**
  - Open source
  - Cloudbursting (overflow to cloud from local compute infrastructure/outsource virtual machines to a public cloud)
  - Cloud federation



References (1)

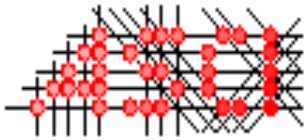
- *Grid Resource Management: State of the Art and Future Trends*, Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz (eds.), Kluwer Publishing, 2004
- R. Buyya et al., "A Taxonomy and Survey of Grid Resource Management Systems," *Software—Practice and Experience*, Vol. 32(2), 2002
- R. Yahyapour, Grid Resource Management and Scheduling--Current State and Future, Totorial at Euro-Par 2004
- Douglas Thain, Todd Tannenbaum, Miron Livny, "Distributed computing in practice: the Condor experience," *Concurrency and Computation - Practice and Experience* 17(2-4): 323-356, 2005
- Y-T. Wang and R.J.T. Morris, "Load Sharing in Distributed Systems," *IEEE Trans. on Comp.*, Vol. C-34, 204-217, 1985.
- D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, J. Pruyne, "A Worldwide Flock of Condors: Load Sharing among Workstation Clusters," *Future Generation Computer Systems*, Vol. 12, pp. 53-65, 1996.



TU Delft

References (2)

- H.H. Mohamed and D.H.J. Epema, "KOALA: A Co-Allocating Grid Scheduler," *Concurrency and Computation: Practice and Experience*, Vol. 20, 1851-1876, 2008.
- TUD grid/cloud group publications:
  - see publications database on www.pds.ewi.tudelft.nl
- KOALA: www.st.ewi.tudelft.nl/koala
- Globus: www.globus.org
- Condor: www.cs.wisc.edu/condor
- Open Grid Forum: www.ogf.org
- See the CCGrid, Grid, and HPDC conferences for current research



TU Delft