Work Binder
Application Service

www.see-grid-sci.eu

4th EGEE User Forum / OGF 25 and OGF Europe's 2nd International Event

2-6 March 2009

Branko Marović, Milan Potočnik
Belgrade University Computer Centre
Miloš Ivanović (University of Kragujevac)
Serbia
branko.marovic rcub.bg.ac.rs

The SEE-GRID-SCI initiative is co-funded by the European Commission under the FP7 Research Infrastructures contract no. 211338

Simplified Access to Computing Resources

- Provide simple and extensible communication and parallel programming paradigm
  - Quickly deliver worker jobs/processes
  - Facilitate communication between clients and workers
  - Make the infrastructure details transparent to users - abstract the infrastructure providing computing resources
- Build upon the producer-consumer paradigm to create associations between processes/jobs, but also involves resource allocation on top of external schedulers (grid workload managers and local queues)
- The current execution environment is gLite. The interaction with the infrastructure can be easily mapped to other grid and cloud computing platforms

Is This Another Pilot-job Infrastructure?

SEE-GRID-SCI
SEE-GRID eInfrastructure for regional eScience

- Both start placeholder jobs asynchronously and match them with the actual work
- Pilot infrastructures
  - Matching of job with the data describing the work to be done
  - If there is no work, the job exits immediately
  - The pilot-infrastructure only influences the job upon selection of work item from the repository
- Our approach
  - Matching is not done with the data from the work repository, but by establishing association with a live client interacting with the job
  - A job without a matching client can wait for some time in the pool of ready jobs
  - The communication between the client and worker can be intercepted (for logging, filtering, or altering by the mediator)

Users' Perspective of Applications and Processing Infrastructure

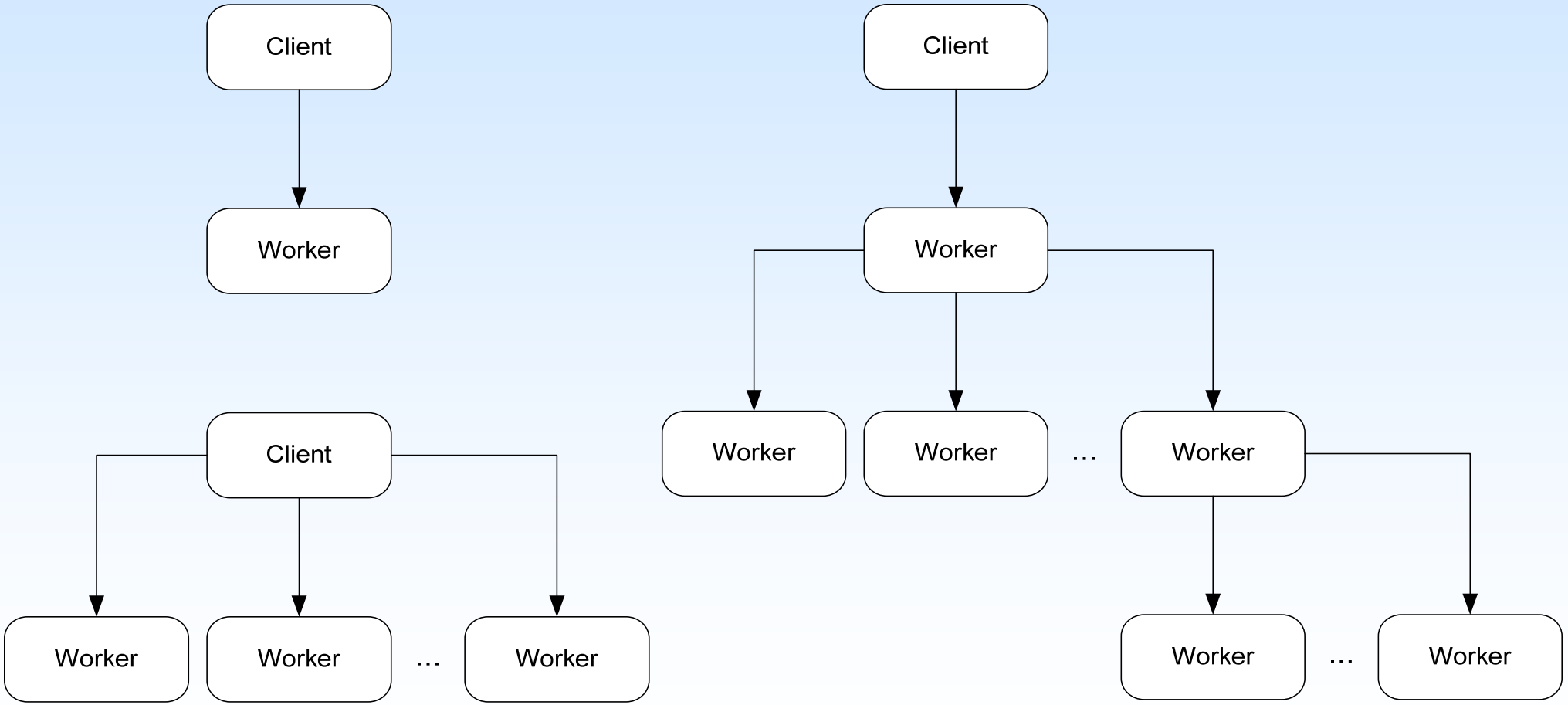
- Communication between clients and workers is the key feature
  - Required communication amount and timing should be suitable for TCP/IP network infrastructure
- Simple plug-in style programming model
  - Build interaction through a clean programming API, not through external descriptions (JDL) or scripting
  - Simple client, binder, and worker APIs and configurable client and worker code, and support for external (non-Java) worker programs
  - Support for direct (not through the binder) and application managed client-worker communication
  - Authorised users can easily install and manage individual applications on grid sites
- Security
  - User authentication via grid certificates (proxy, PKCS#12 bundles)
  - VOMS-based authorisation
  - Jobs are currently executed on behalf of Work Binder, not user - could apply glexec to change grid credentials of running jobs

Applications

SEE-GRID-SCI
SEE-GRID eInfrastructure for regional eScience

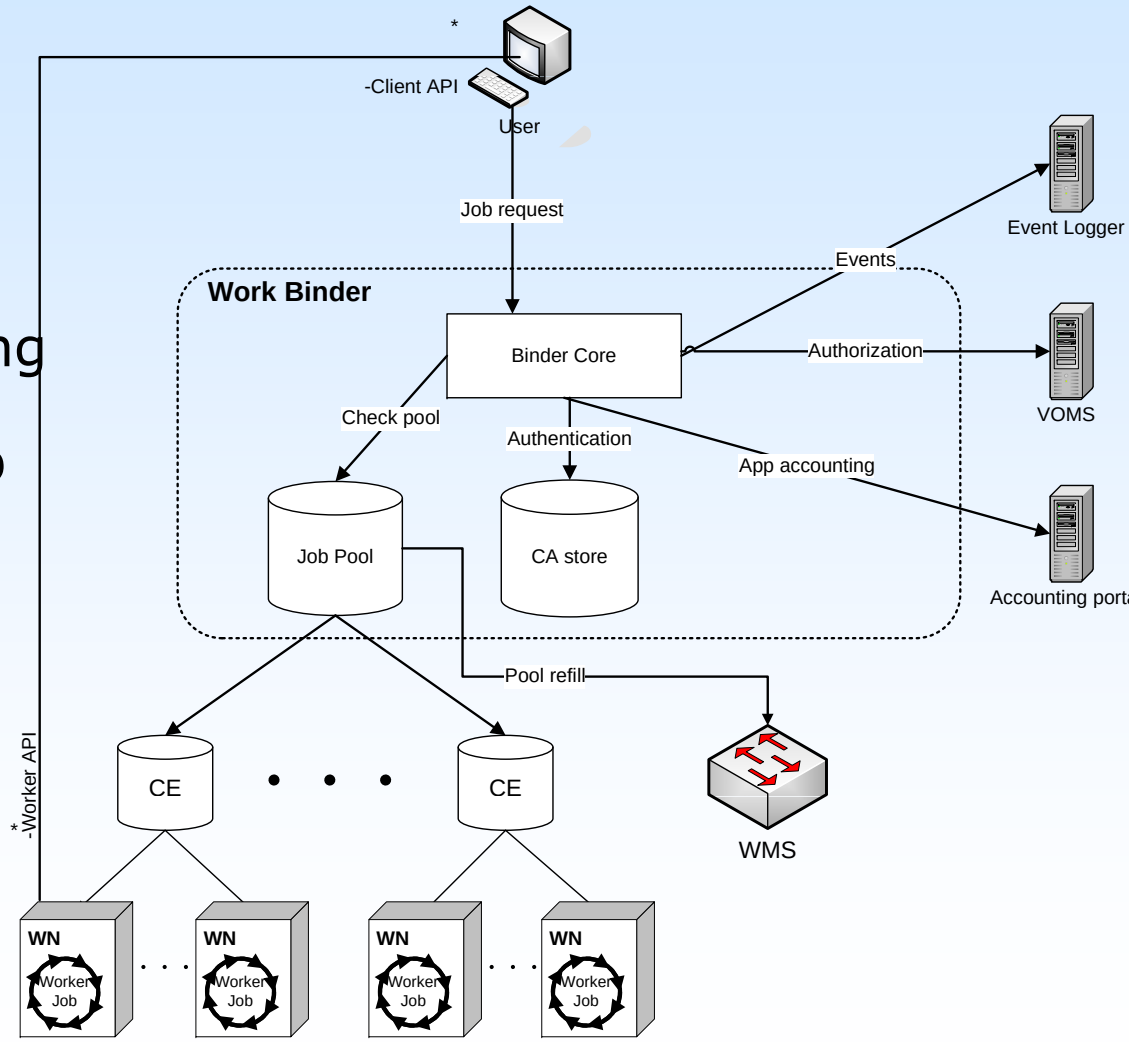
- Having readily available jobs is crucial for truly interactive applications
  - No warranties from infrastructure on job start latencies: hours, days!
- This pool also allows (almost) delay free
  - Dynamic (on demand) allocation of worker jobs
  - Iterative use of short jobs
- Several applications share the same pool of jobs – smaller overhead
- Master – Worker relationship allows creation of arbitrary dynamic hierarchies
  - Interactive applications
  - Dynamic workflows

Establishing Dynamic Hierarchies



Key Components

- Client – a program that requests and gets jobs, and then communicates with them. It may also be a job!
- Binder – a mediator providing a level of abstraction above the actual scheduler and job management infrastructure
- Worker – a placeholder job contacting the binder and, after being paired with the client, executes application code (class or separate program)

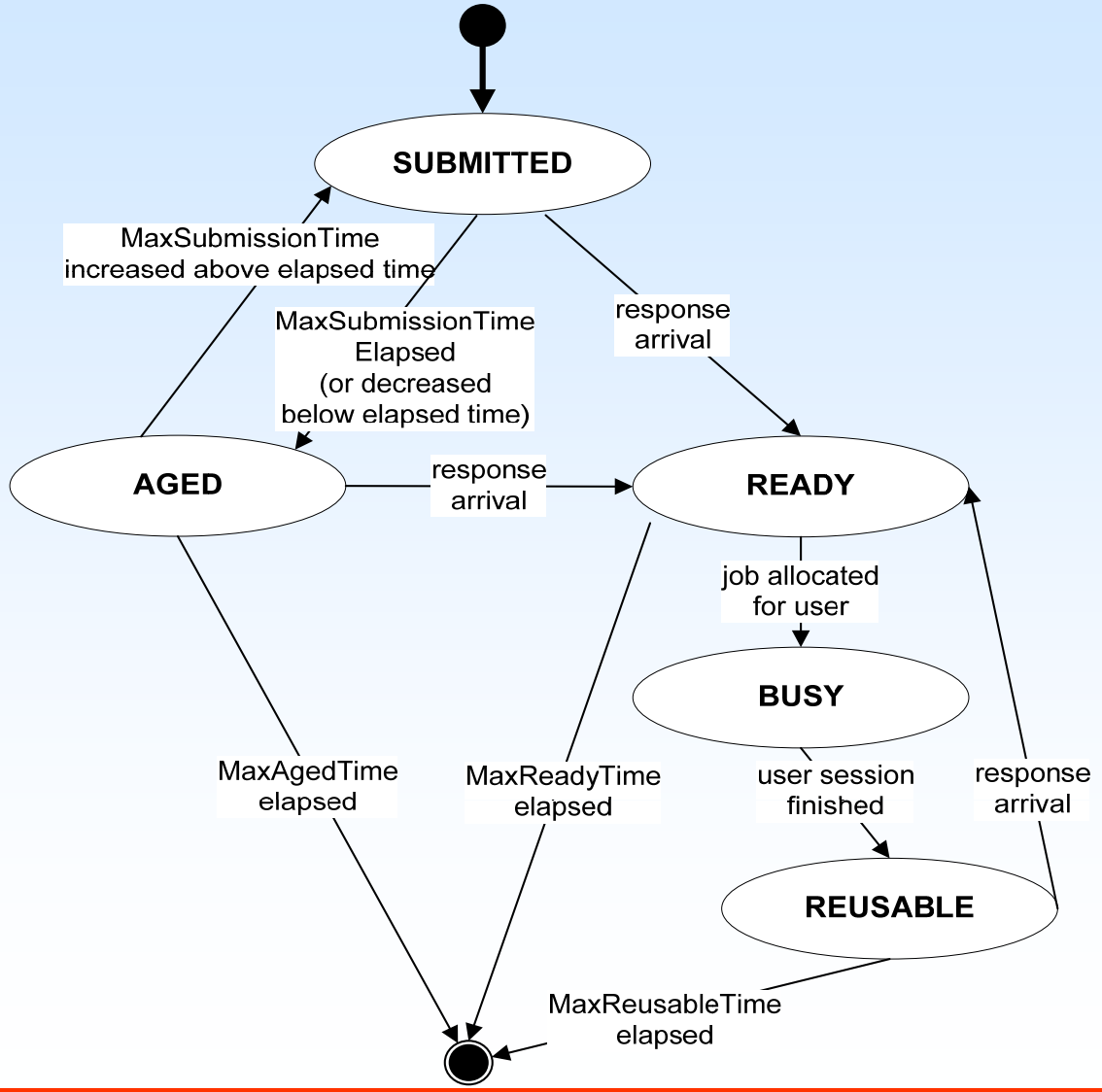


Pool of Jobs

SEE-GRID-SCI
SEE-GRID eInfrastructure for regional eScience

- Pre-emptive submission prevents start delays
- Adapt to site responsiveness
- Prevent flooding of job queues
- Job lifetime management
- Monitor consumption rate and bulk load
- Maintaining the satisfactory number of jobs in the pool
- Adhering to site policies

Job states within binder



Pool Management

- Complex job submission and pool maintenance strategies are employed in order to
  - Minimise the number of idle jobs
  - Reduce the chance for a client to face the empty pool
- Operational modes
  - Idle (no clients) – only willing CEs (Computing Elements) provide the minimal reserve
  - Active (during actual usage) – reserve maintained by all CEs
- Job allocation strategies
  - Panic/full throttle – global: get jobs ASAP from the best responding sites
  - Regular – on per-site basis, distribute the load fairly among CEs (within limits of their commitment)

Simple Producer-Consumer Problem?

- Rules
  - Clients randomly request and consume worker jobs
  - Some time is required to produce new jobs
  - Job creation time (job start latency) is significant and provider (CE) dependant, but not fully deterministic
  - The infrastructure and local policies of participating resource providers (CEs) impose some constrains
  - Minimise the number of idle jobs
  - Reduce the chance for a client to face the empty pool, and thus be blocked or rejected
  - Active worker cannot be reassigned to a new client until finished (allowing this would permit having a smaller pool or reassignment of jobs used by abortable applications!)
- The resulting problem is far from being trivial!

Pool Size Calculation

- In idle mode fixed by per-site commitments
- In regular mode depends on the predefined minimum, number of active clients, client arrival rate
- Client arrival rate is regularly sampled
- Old values are being attenuated exponentially

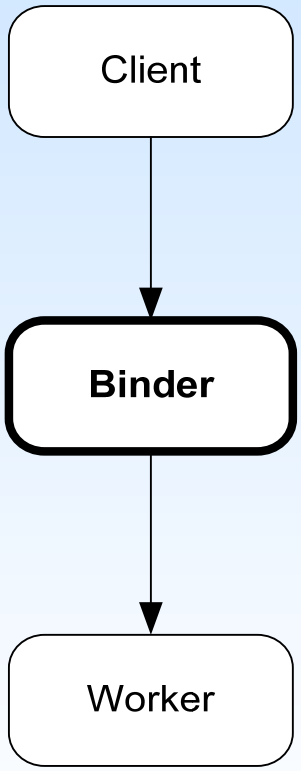
$$x_{residual} = x_{original}\, e^{-\lambda t} \qquad \lambda = \frac{\ln 2}{T_{half-decay}}$$

- A similar approach is used to calculate per-site job submission delays
- Global and CE parameters
  - Current usage
  - Response time
  - Expiration of ready jobs
  - Expected arrival of submitted and reused jobs
  - Expected new load

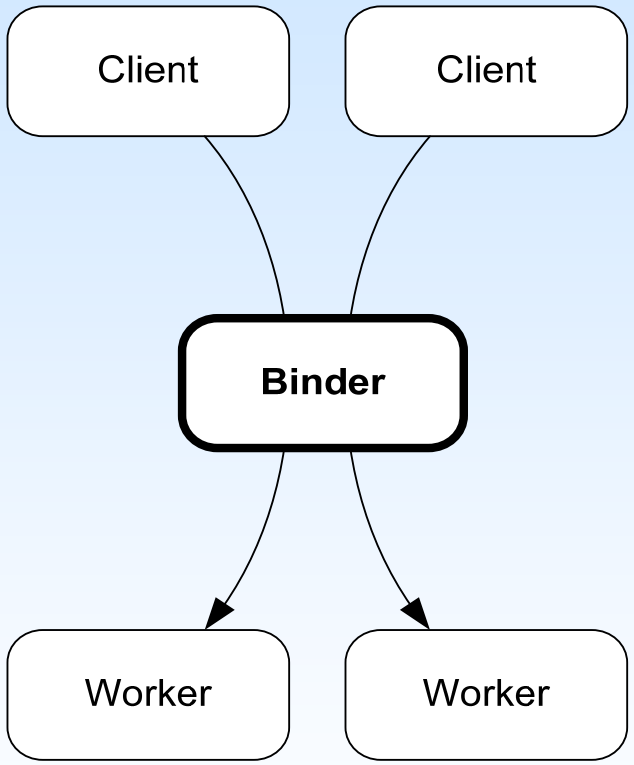
Communication

- Binder is a proxy between the client and worker
- Simple TCP-based protocol
- Client
  - Required maximum wall clock time
  - Preferred resource provider(s) (CEs)
  - Application
  - End-user credentials (for check or glExec)
  - Description of further communication (e.g. client address:port)
- Worker
  - Owning resource provider (CEs)
  - Job instance ID
  - Available time
  - Application ID
  - Could also offer an alternative access point
- Error descriptions and routing data are also communicated
- Application-dependant header may follow
- Further application dependant communication through the same channel or by establishing direct communication
- After assigned work is finished, the job reports back to the binder

Providing non-computing services within the mediator



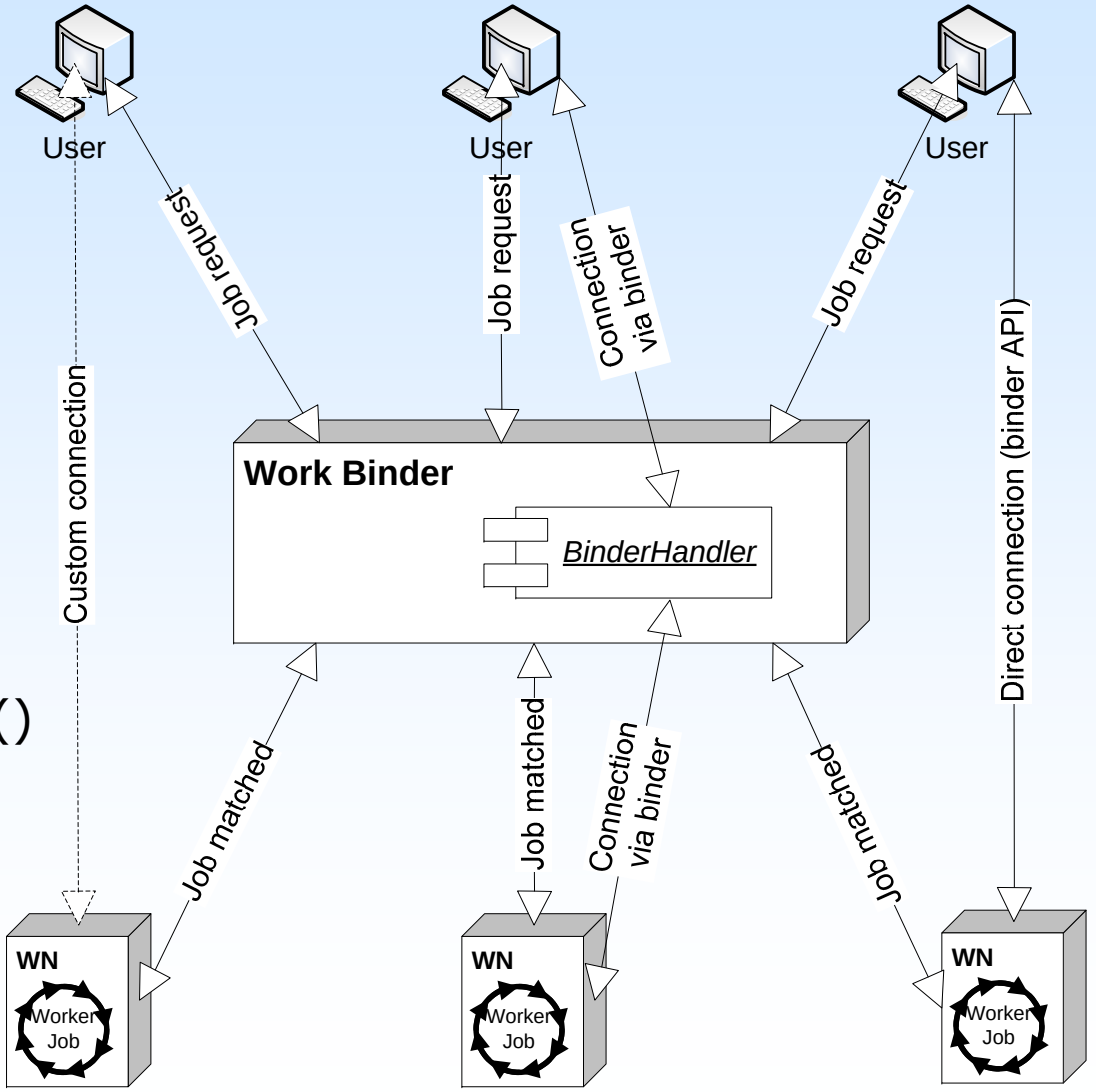
- Persistence: handling/caching of results
- Monitoring and logging/performance data
- Application-level access control
- …



- Orchestration of individual client sessions, e.g. for multi participant whiteboards
- …

Client

- Can communicate with the worker through the initial socket to binder or request a direct connection
- ClientConnector
  - connect(<initial parameters>)
  - get<connection parameter>()
- Communication primitives
  - sendData(..)/receiveData()
  - sendString(..)/receiveString()
  - getSocket()
- Can connect to several workers



Binder

- Maintains the pool of free workers by submitting new jobs whenever needed
- Enforces infrastructure and site usage policies
- Performs authentication and authorisation
- Matchmaking between clients and workers
- A generic handler simply relays traffic between the client and worker (as a proxy server)
- Can be extended with application-specific handlers mediating the communication
- Key operational data (jobs, CEs) persisted in a database
- Usage logging

Application-Specific Binder Handler

SEE-GRID-SCI
SEE-GRID eInfrastructure for regional eScience

- Can provide some non-computational services

```
public interface BinderHandler
    public void run(BinderConnector binderConnector);
```

- Connector provides communication primitives and access to underlying sockets towards the client and worker
- If the traffic is observed, the handler can track requests and responses and provide some features useful in interactive visualisation applications
  - Monitoring, logging, performance tracking
  - Persistence: handling/caching of results
    - May respond instead of worker to persistence-related requests
    - Can record some worker responses and, if the same request is later repeated, provide the response from the cache instead of letting the worker to recalculate
  - Application-level access control

Worker

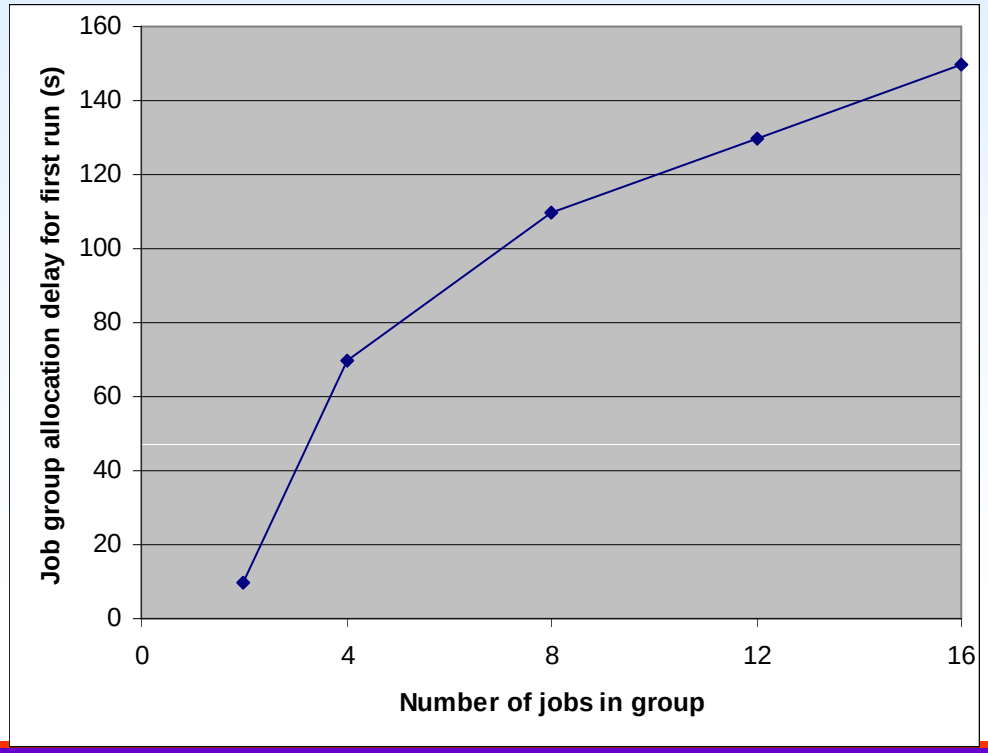
- Performs the actual computation
- Ran within worker jobs
- Information about supported applications, location of their code and binder in configuration file
- Connects to the binder and waits for a client for some time
- Starts application code
  - External program – via ExternalExecutor or ExternalListener
  - Application handler

```
public interface WorkerHandler
    public void run(WorkerConnector workerConnector);
```

- Further communication is performed directly or via binder, connector provides
  - Communication primitives and access to the socket with the client (via binder or directly), or
  - Client-provided data for establishing alternative connection
- After the work is completed, reconnects to the binder asking for more work within job time limits, or jobs of finished workers are reused

Allocation of Jobs by Lizza-PAKP Application

- Application uses groups of jobs in series of runs
- The jobs are requested until a group of desired size is created with 10s delay between tries



- There is no delay when there are already enough jobs in the pool, as in the case with 2 jobs, and after first series invocations
  - The delay is only noticed during warming up from idle mode with insufficient initial number of jobs
  - Adapts well to increasing number of requested jobs (but limited by the number of free nodes in underlying grid infrastructure)
- Subsequent delays are negligible - this needs to be re-evaluated in a concurrent setting
  - Job recycling of the jobs by the pool has dramatic positive impact on job allocation speed

Status

- Currently being tested with two SEE-GRID applications
- Monitoring of infrastructure usage and performance done with Event Logger Service
- Latest: The initial implementation of MPI support with a worker handler using mpirun
- Considering adding client utilities for file transfer and allocation and management of several workers
- May be extended with data provenance and data staging for workflows

- Availability and documentation
  - LGPL 3
  - Wiki, Javadoc, two examples (Echo & Reverse Remote Shell)
    - http://wiki.egee-see.org/index.php/Work_Binder_Application_Service
  - SVN/Trac source code repository (certificate based access)
    - https://trac.egee-see.org/trac/work-binder.see-grid-sci.eu
    - svn://svn.egee-see.org/svn/work-binder.see-grid-sci.eu
- Interested applications with their requirements are welcome!