Using Sun Grid Engine and Globus to Schedule Jobs Across a Combination of Local and Remote Machines

By Geoff Cawood and Paul Graham
Edinburgh Parallel Computing Centre (EPCC)

Telephone: +44 131 650 5120
Email: geoffc@epcc.ed.ac.uk

The Team

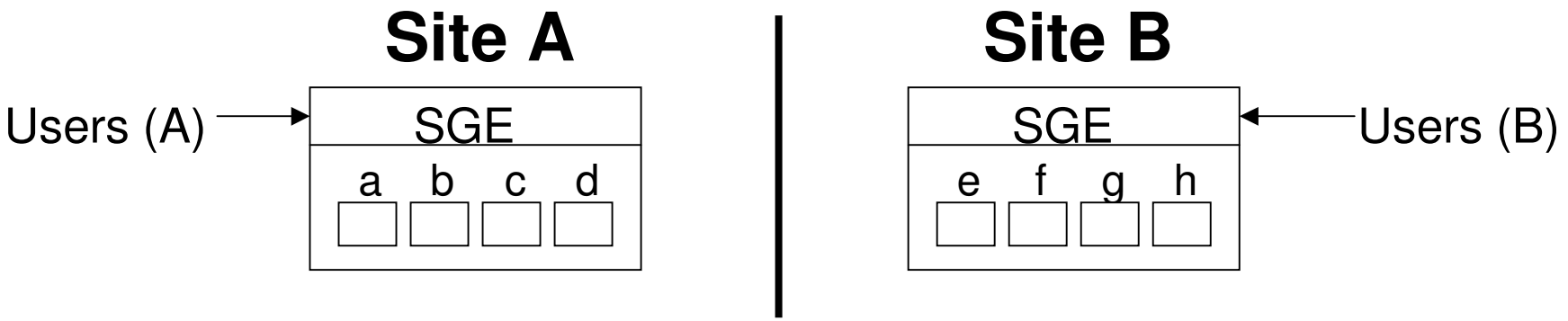
- EPCC
  - Edinburgh Parallel Computing Centre
  - Part of the University of Edinburgh in Scotland
  - “A technology-transfer centre for high-performance-computing”
  - http://www.epcc.ed.ac.uk
- Project Personnel
  - Terry Sloan
    - Project leader - tms@epcc.ed.ac.uk
  - Thomas Seed
  - Ali Anjomshoaa
  - Geoff Cawood
  - Paul Graham

The Project

- A collaboration between EPCC and Sun
  - Called simply ‘Sungrid’ within EPCC
  - 57 person months of EPCC effort
  - Significant staff contribution from Sun

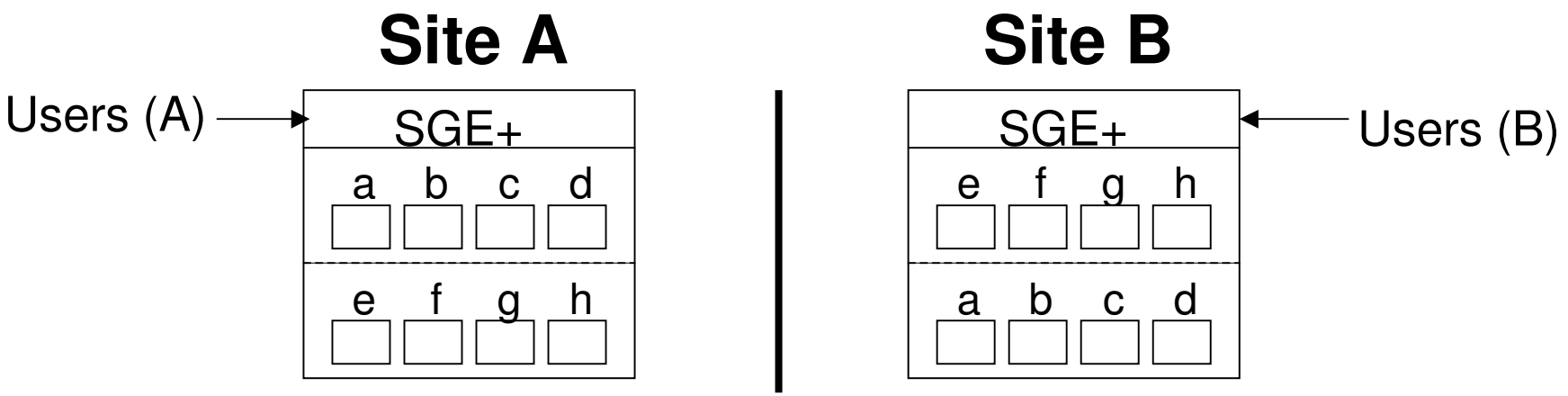
The Project

- Project scenario
  - Two collaborating sites A and B both have some machines
  - Both sites run Sun Grid Engine (SGE) to schedule jobs
  - Local demand for machines is variable
    - Sometimes it exceeds supply
    - Other times machines lie idle



The Project

- Ideal Situation
  - If sites A and B could expose their machines to each other across the internet through SGE…
    - Both sites could enjoy through-put efficiency improvements
    - Large gains when one site is busy while the other is idle



The Project

- Final goal
  - Develop a job-scheduler based on SGE which can schedule jobs across a combination of local and remote machines
  - Globus will provide a secure means of running jobs on remote sites

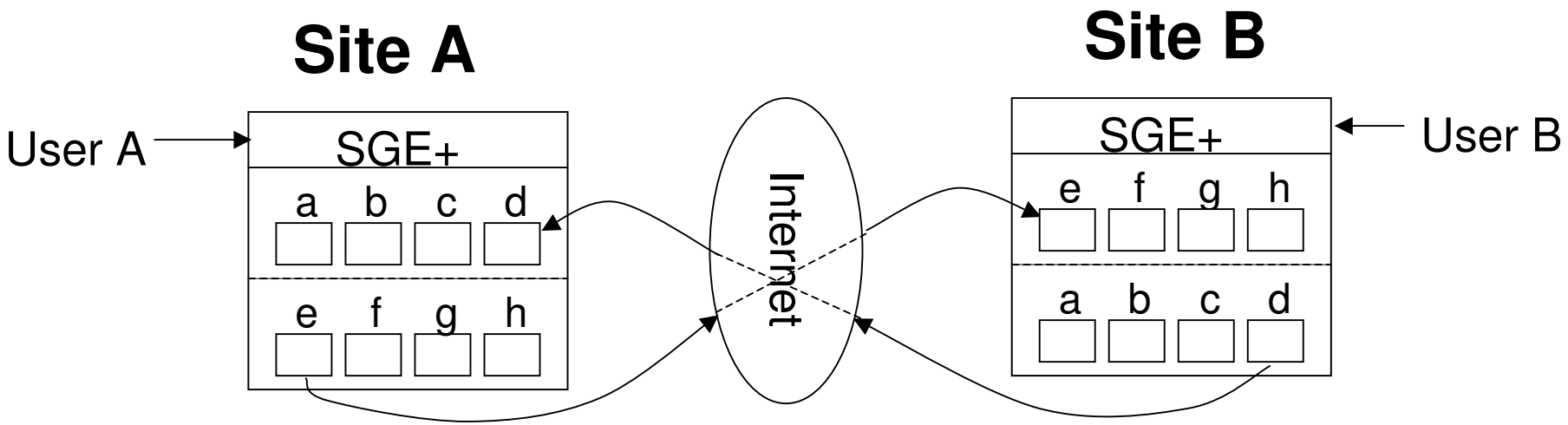
Project Status

- Self-education and code analysis
  - Current phase
  - Learning about SGE internals and Globus
  - Documenting relevant sections using UML
- Requirements
- Design
- Implementation
- Final test

Workshop Goals

- Meet new colleagues
- Learn more about the internals of SGE
- Find out about any other attempts to extend SGE functionality over multiple sites
- Solicit feedback on the solution approaches we might adopt
  - Three approaches outlined in the remaining slides

1. Transparent Scheduling

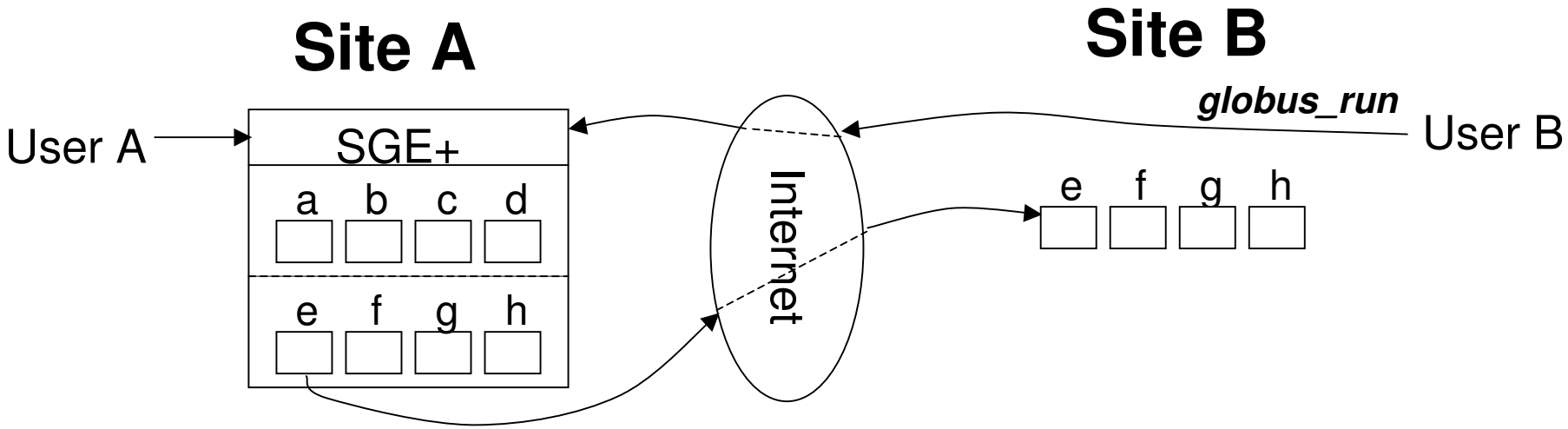


- Outline
  - SGE+ is SGE enhanced to work with remote machines
  - Both sites can run SGE+
  - Change scheduler to take local/remote issues into account
  - Modify daemons (execd, commd etc.) to work across internet?
  - Or use globus calls to fork jobs and query machine and job status?

1. Transparent Scheduling

- Pros
  - Simple approach (?)
  - Usability – nothing new for users to learn
- Cons
  - Low quality scheduling decisions (?)
    - Neither SGE+ system knows exactly what jobs are running on a machine
    - May be a time-lag in getting query results back from remote site
  - Duplicated scheduling effort
    - Both SGE+ systems are trying to do the same thing - wasteful
  - Poor scalability
    - With N sites, might need N execd daemons per machine
  - Queue control – awkward to turn off remote access temporarily
    - Have to explicitly ask other sites to stop forking jobs
  - Remote load management tool is by-passed
    - Remote administrator might not want lots of forked jobs

2. Centralised Scheduling

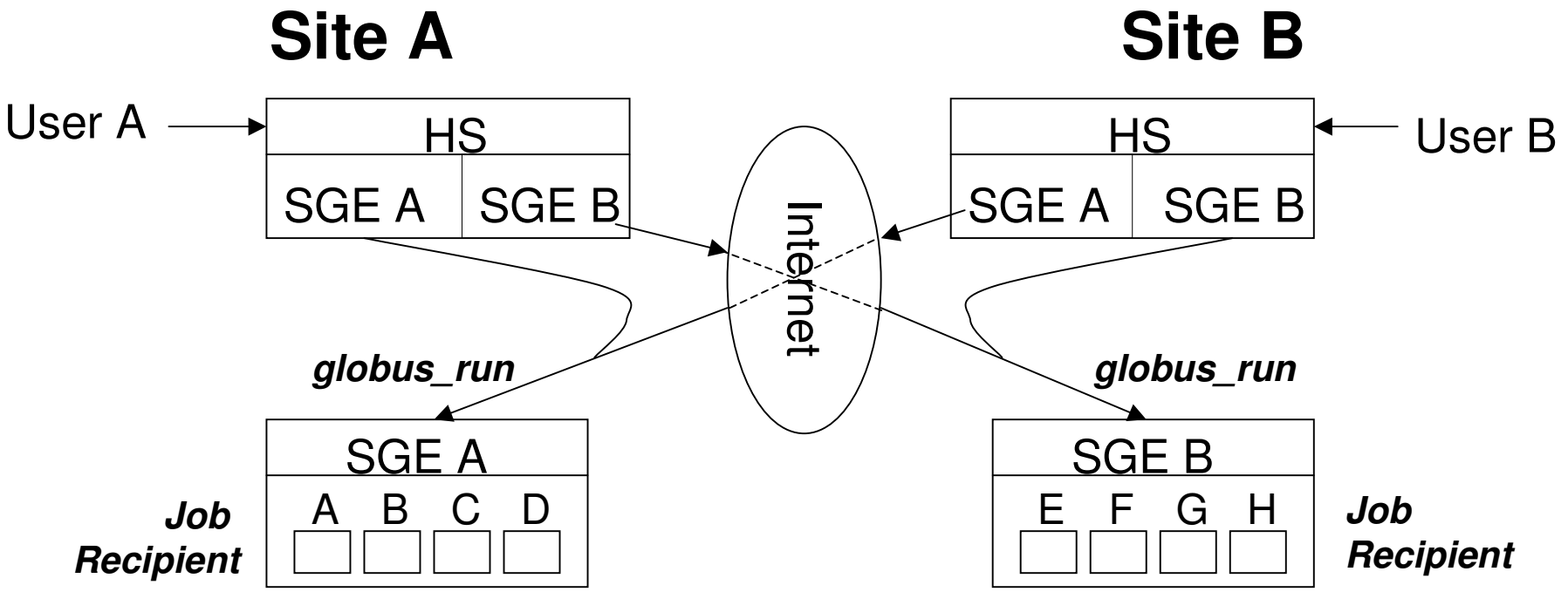


- Outline
  - Similar to previous approach but one site (A) nominated as master
    - All jobs from all sites are submitted through the master
  - Site A runs SGE+ as described in previous approach
  - Users at site B call globus_run to submit jobs to the enhanced SGE+ running at site A

2. Centralised Scheduling

- Pros
  - Good scheduling decisions due to accurate information
  - Under the remote daemons scheme:
    - Site B users might still be able to enjoy SGE job monitoring
    - Site B administrator might still have convenient queue control
- Cons
  - Need back-up server/site
    - Users at B will not want to wait for A to come back up
  - Political issues
    - Site B administrator loses control – may not be acceptable
  - Need very high degree of co-operation
    - Correct software applications installed and maintained
    - User-support
    - Accounting issues

3. Hierarchical Scheduling



- Outline
  - Build a new higher-level scheduler that schedules across globus job-recipients rather than individual machines or queues

3. Hierarchical Scheduling

- Outline continued
  - Define a ‘globus job-recipient’ as a machine or load-management tool that can:
    - Receive jobs via globus and execute them
    - Respond to queries (sent via globus) about machine and job status
  - Modify SGE to provide such a job-recipient interface
    - If it doesn’t already… (qstat may be enough?)
  - Introduce a new higher-level Hierarchical Scheduler (HS) to schedule across job-recipients
    - Each site has its own HS
    - Only users at site A can submit jobs to the HS at A
    - Only users at site B can submit jobs to the HS at B
    - When a job arrives at a HS, it queries the job-recipients then decides which one to send the job to

3. Hierarchical Scheduling

- Pros
  - Problem decomposition
    - Divide and conquer
    - Usually a good idea
  - Should give reasonable efficiency improvements
  - Local scheduling decisions made locally, as today
  - Queue access controlled locally
    - Administrator can turn queues on/off easily through SGE, as today
  - Extensible
    - Just make each HS aware of new job-recipients
  - No ‘behind-the-scenes’ job-forking
    - Jobs arrive at a site through the intended entry point so SGE has full knowledge about what jobs are running, as today

3. Hierarchical Scheduling

- Cons
  - Large effort to build new HS tool and front-end
  - Scheduling decision duplication
    - HS decision algorithm may be very similar to what SGE does already
  - HS scheduling is coarser-grained
    - So decisions might not be as good as centralised scheduling
  - HS user-interface uncertain
    - What information on job progress would the HS user see?
  - Political issues
    - This may not be what our project proposers want/expect

Any Comments?

- If you have any comments on these approaches please come and talk to us!
- We are very interested to hear your opinions
- Danke schön!