SHARCNET™

computing tomorrow's solutions

SHARCNET
Job Scheduling Policy and Requirements

Strawman Document

2005/6

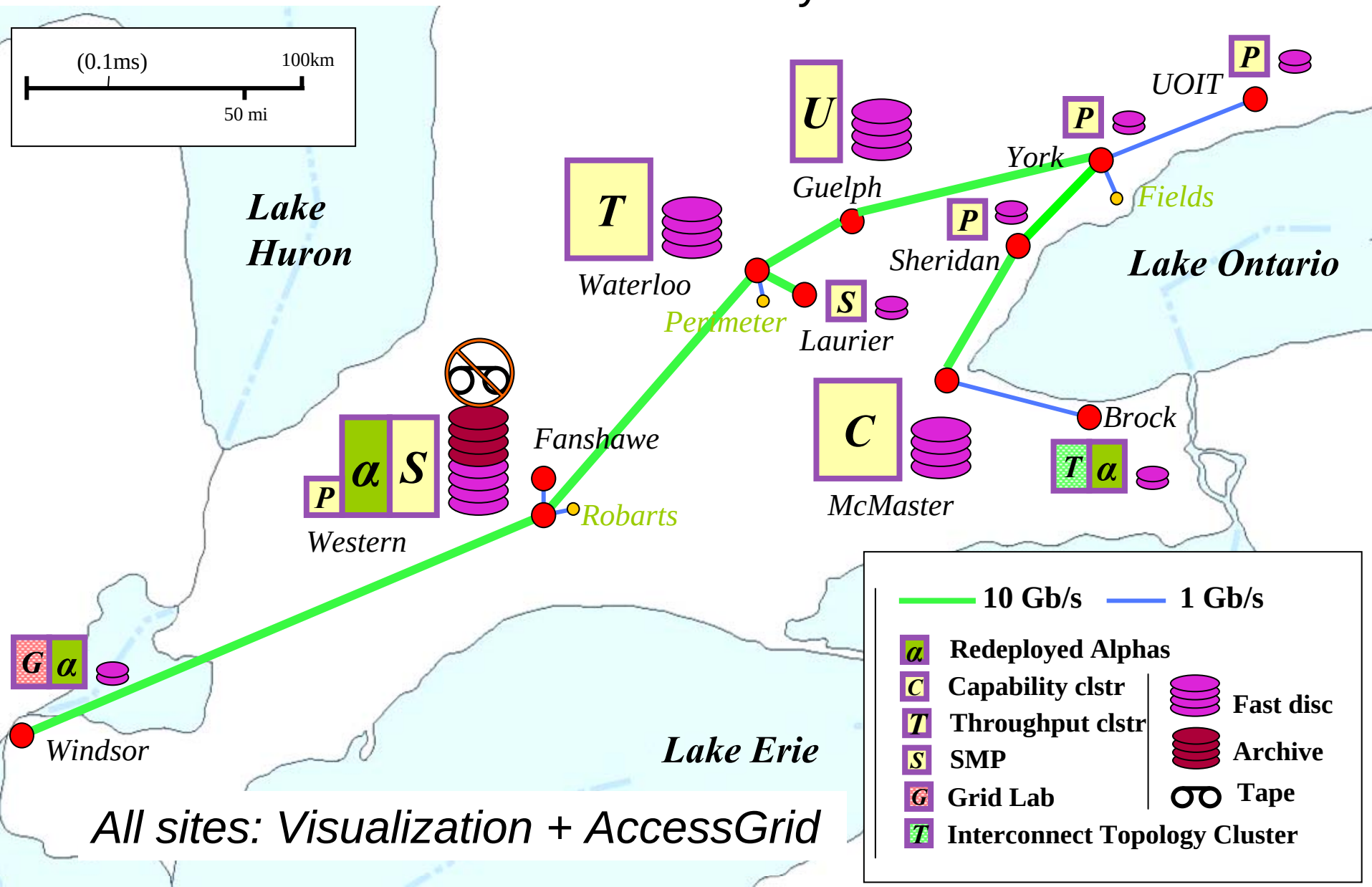
Overview

- High-level objectives
- Scheduling schematic
- Resource costing
- Fair access
- Job scheduling specifics
  - Test jobs
  - Serial
  - Parallel
- Technical requirements

SHARCNET™

SHARCNET™

Proposed expansion; Fully funded 4/3/2004



General Policy Issues

Overall aim is to use all **SHARCNET** resources effectively & maximize user throughput

- **System-wide accounting**: user priorities determined from use of all SN facilities (by user, group or project)
- **Appropriate cluster use,** but avoid wasting cycles
  - *SN2 architectures tailored to specific applications but all can run, e.g., serial jobs*
- **Contributors:** pre-emptive access
- **Batch operation**: jobs normally run in batch mode – throughput timescale ideally ~ $f \times$ runtime, $f \lesssim 1$; allow testing & debugging
- **Fair use**: users should have similar probability of *starting* a job (at similar levels of use)
- **Transparent accounting & feedback**: procedures and information used to determine priority/fairshare, expected queue time etc. should be available to users

SHARCNET™

General Requirements

- "Queue-less" scheduling: treat systems as one flat resource, let scheduler handle job placement: improves efficiency and flexibility (really partition-less scheduling, *cf.* APAC document)
- Central database: records jobs across all SHARCNET
- SN-wide queuing:
  - Reference global user statistics to determine fairshare
  - Ability to manage global and local job submission including moving jobs, fault tolerance…
  - work towards "property" queues: serial, small parallel/threaded, licenced software etc.

SHARCNET™

Cost

- "Charge" for CPU time: could measure use of many other resources: memory, network, storage
- Normalised SHARCNET Cost/CPU-Hour (NSC): priorities/fair use determined by the NSC, function of real cpu-time and other factors:
  - User-adjusted priorities: users can choose to run/start at higher or lower priority and corresponding higher or lower NSC; ideally a user could dynamically adjust this priority (perhaps designate a single urgent job)
  - Dynamic cost: a job started as "urgent" at 3 × rate would drop to 1 × rate if machine became empty
  - Varying CPU power: charge less for less powerful CPUs – primarily to encourage use of older systems
  - Incentives: discount NSC/actual to encourage appropriate/beneficial use:
    - **Checkpointing** (especially parallel jobs)
    - **Demonstrated efficiency** (scaling, fraction of node peak, effort etc.)
    - **Certification**
    - **Research reporting**
  - Dedicated time: runs at NSC/actual = 0 (perhaps accumulated at night, weekends etc. within 6-month window)

SHARCNET™

Fair Access

- Generally, once a job is started, it should run to completion: the priority assigned to a user or job affects its probability of starting
- Probability of a job starting should not depend on submission time. Starting probability:
  - Is decreased if a large number of that user's jobs is running
  - Is decreased if that user has accumulated a large NSC over some period*
  - Depends on user-assigned priorities
  - May depend on *dynamic "fairshare" averages: priority decreased for a day, week etc. if heavy use over a day, week etc.
- A user's fairshare state(s), starting probability and queue wait time should be available to them via the portal together with average wait times etc.
- Ability to modify user's priority depending upon group or project use

SHARCNET™

Test Jobs

Users should have ability to run test jobs (even on production clusters). Such jobs/queues should:

- Run quickly
- Be established to avoid misuse: higher NSC or allocation of test time
  - One test job at a time per user
- Need to ensure that test jobs do not orphan preempted jobs (a test job should pre-empt a job of the same size, perhaps preferentially a user's own job; any job that is pre-empted should ideally resume as before once the test job has completed)
- Ideally, such jobs should be flexibly and dynamically scheduled without the need for a special test queue of reserved processors

SHARCNET™

Job Scheduling

- **Serial jobs**: jobs normally run to completion; starting probability is determined as above (do not normally share resources with other equal priority jobs)
- **Parallel jobs**: challenge is to reserve sufficient processors to start job without idling those processors; simple pre-emption can leave stranded jobs
  - Reserve a second slot on a cpu for a pre-empting job and when this occurs, coarsely time slice between the two.
  - Allows pre-empted job to checkpoint – resubmit/migrate
  - Duty cycle perhaps 50/50, but must ensure pre-empted job stops – 80/20? In principle priority could affect duty cycle
  - Efficient scheduling probably requires power-of-2 no. of processors; scheduler should enforce correct no. of processors/node etc.
  - Cost should encourage scaling not merely many procs, suggest:

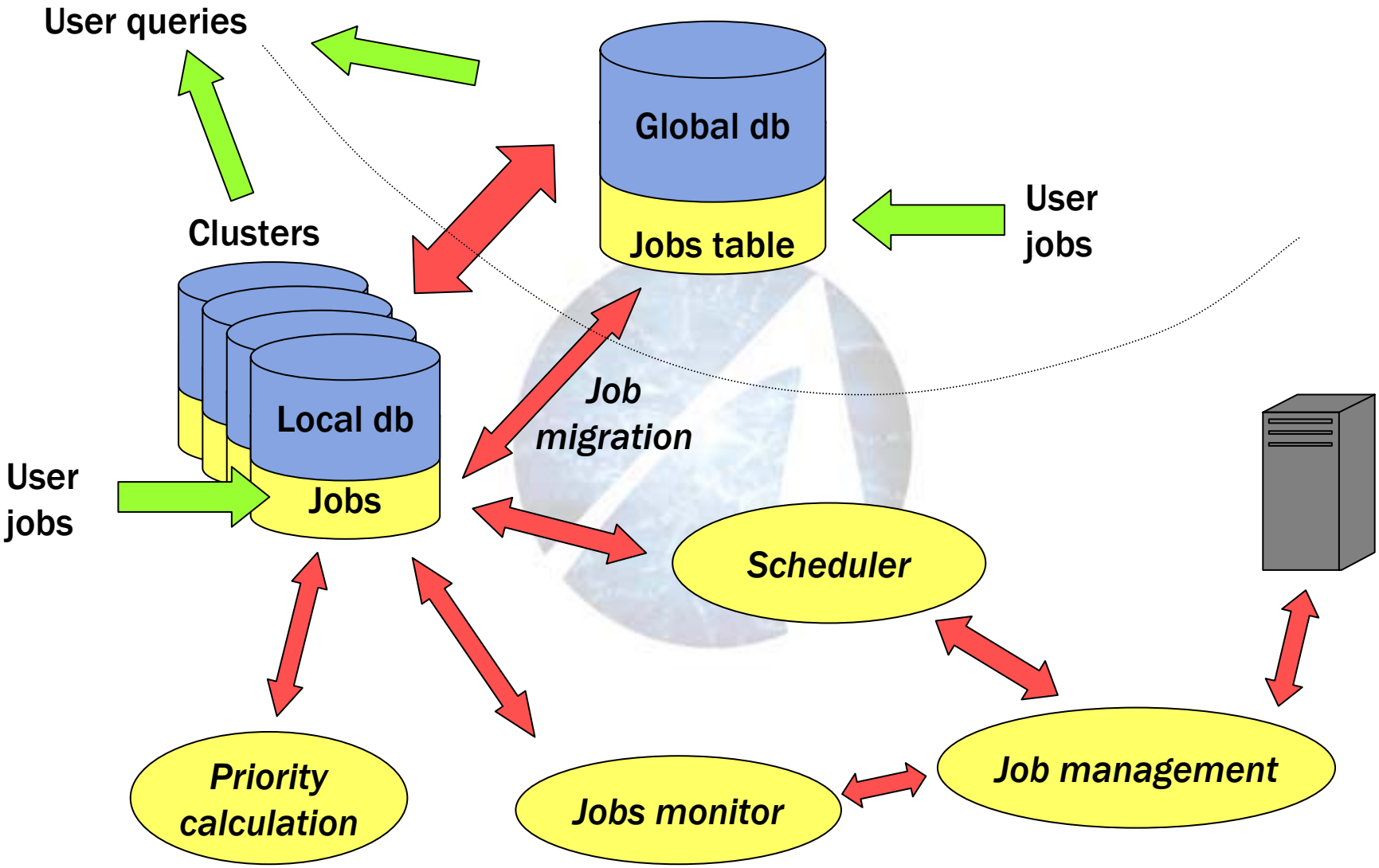
$$NSC := NSC \times T_{cpu} / \sqrt{N_{cpu}}$$

SHARCNET™

Technical Requirements

- Direct access to and control of job starting and placement mechanisms
- Ability to dynamically adjust priority of job (for users and system – cf. nice)
- Ability to do coarse-grained time-slicing (suspend/resume)
- Detailed control of priority/fairshare calculation
- Ability to access system accounting tables and generate SHARCNET-wide data
- …

SHARCNET™


User queries
Global db
Jobs table
User
jobs
Clusters
Local db
Jobs
Job
migration
User
jobs
Scheduler
Priority
calculation
Jobs monitor
Job management

SHARCNET™