Scheduling in an HPC environment

Andrea Lorenz

Aachen University of Technology,
Center for Computing and Communication (CCC)

History

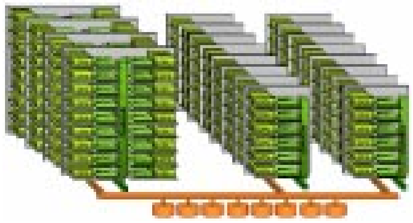
- NQS 1994 - 2001
- Heterogeneous cluster (by order of appearance):
  - IBM (AIX)
  - SGI (IRIX)
  - HP (HP-UX)
  - Fujitsu (UXP/V) (Vector computer)
  - Sun (Solaris)
  - PC (LINUX)
- Home-grown scheduler across all these (designed with NQS in mind)

Current state

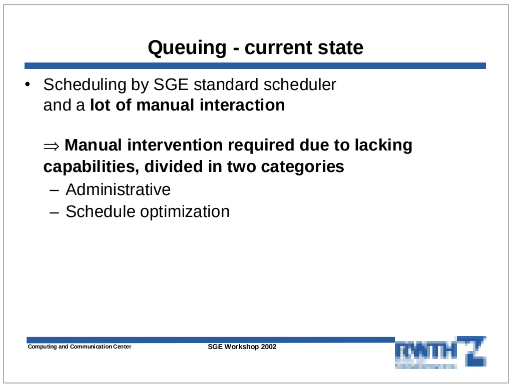
- Sun Grid Engine since 2001 in production
- Still (less) heterogeneous cluster (Fujitsu and IBM withdrawn)
- SunFire-Cluster as an main compute resource

SunFireCluster at Aachen University

4 x Sunfire 15k **16 SunFire 6800**



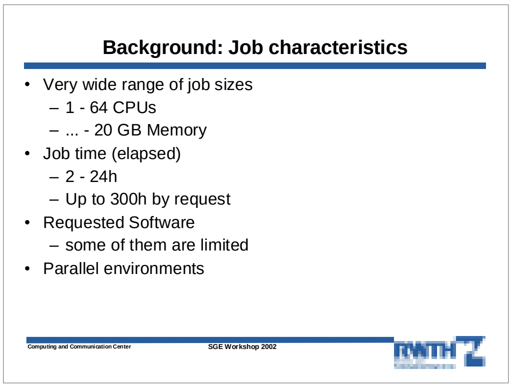
Storage Area Network (SAN)



Queuing - current state

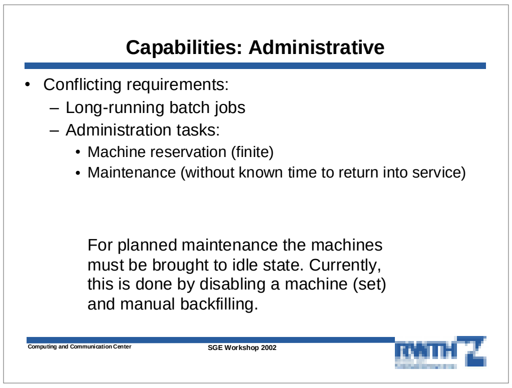
- Scheduling by SGE standard scheduler
  and a **lot of manual interaction**

  ⇒ **Manual intervention required due to lacking capabilities, divided in two categories**
  - Administrative
  - Schedule optimization



Background: Job characteristics

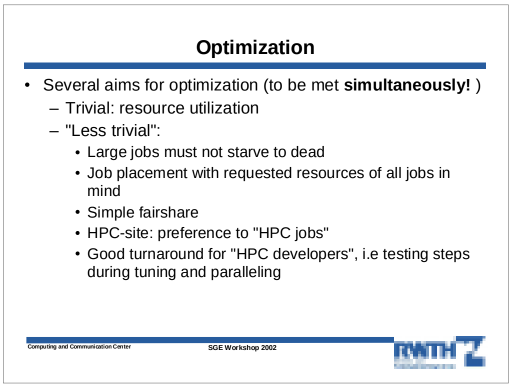
- Very wide range of job sizes
  - 1 - 64 CPUs
  - ... - 20 GB Memory
- Job time (elapsed)
  - 2 - 24h
  - Up to 300h by request
- Requested Software
  - some of them are limited
- Parallel environments



Capabilities: Administrative

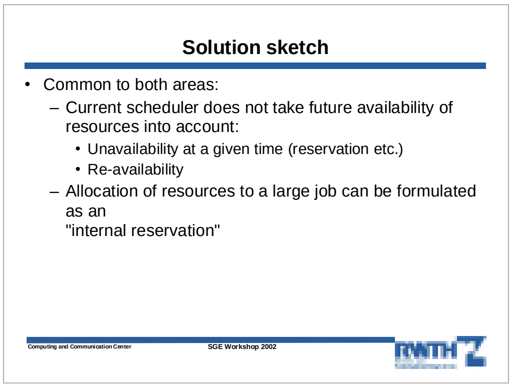
- Conflicting requirements:
  - Long-running batch jobs
  - Administration tasks:
    - Machine reservation (finite)
    - Maintenance (without known time to return into service)

For planned maintenance the machines
must be brought to idle state. Currently,
this is done by disabling a machine (set)
and manual backfilling.



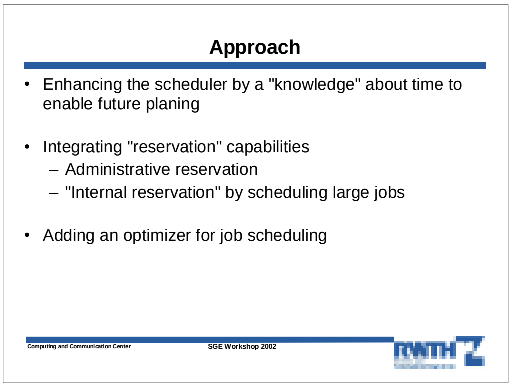
Optimization

- Several aims for optimization (to be met **simultaneously!**)
  - Trivial: resource utilization
  - "Less trivial":
    - Large jobs must not starve to dead
    - Job placement with requested resources of all jobs in mind
    - Simple fairshare
    - HPC-site: preference to "HPC jobs"
    - Good turnaround for "HPC developers", i.e testing steps during tuning and paralleling



Solution sketch

- Common to both areas:
  - – Current scheduler does not take future availability of resources into account:
    - Unavailability at a given time (reservation etc.)
    - Re-availability
  - – Allocation of resources to a large job can be formulated as an
    "internal reservation"

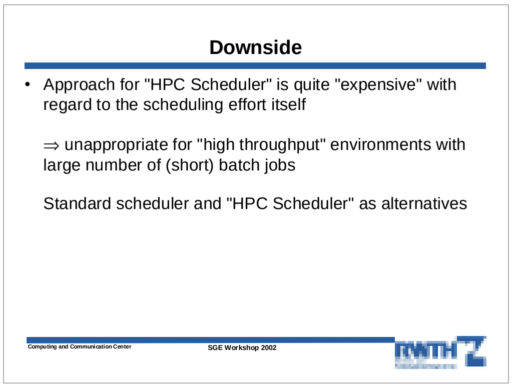


Approach

- Enhancing the scheduler by a "knowledge" about time to enable future planing

- Integrating "reservation" capabilities
  - – Administrative reservation
  - – "Internal reservation" by scheduling large jobs

- Adding an optimizer for job scheduling

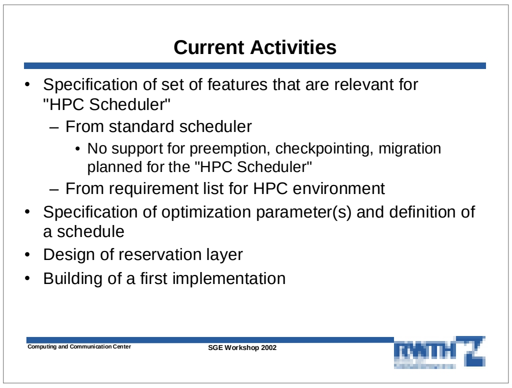


Downside

- Approach for "HPC Scheduler" is quite "expensive" with regard to the scheduling effort itself

  ⇒ unappropriate for "high throughput" environments with large number of (short) batch jobs

  Standard scheduler and "HPC Scheduler" as alternatives



Current Activities

- Specification of set of features that are relevant for "HPC Scheduler"
  - – From standard scheduler
    - No support for preemption, checkpointing, migration planned for the "HPC Scheduler"
  - – From requirement list for HPC environment
- Specification of optimization parameter(s) and definition of a schedule
- Design of reservation layer
- Building of a first implementation