PBS INTERNALS

PBS & TORQUE

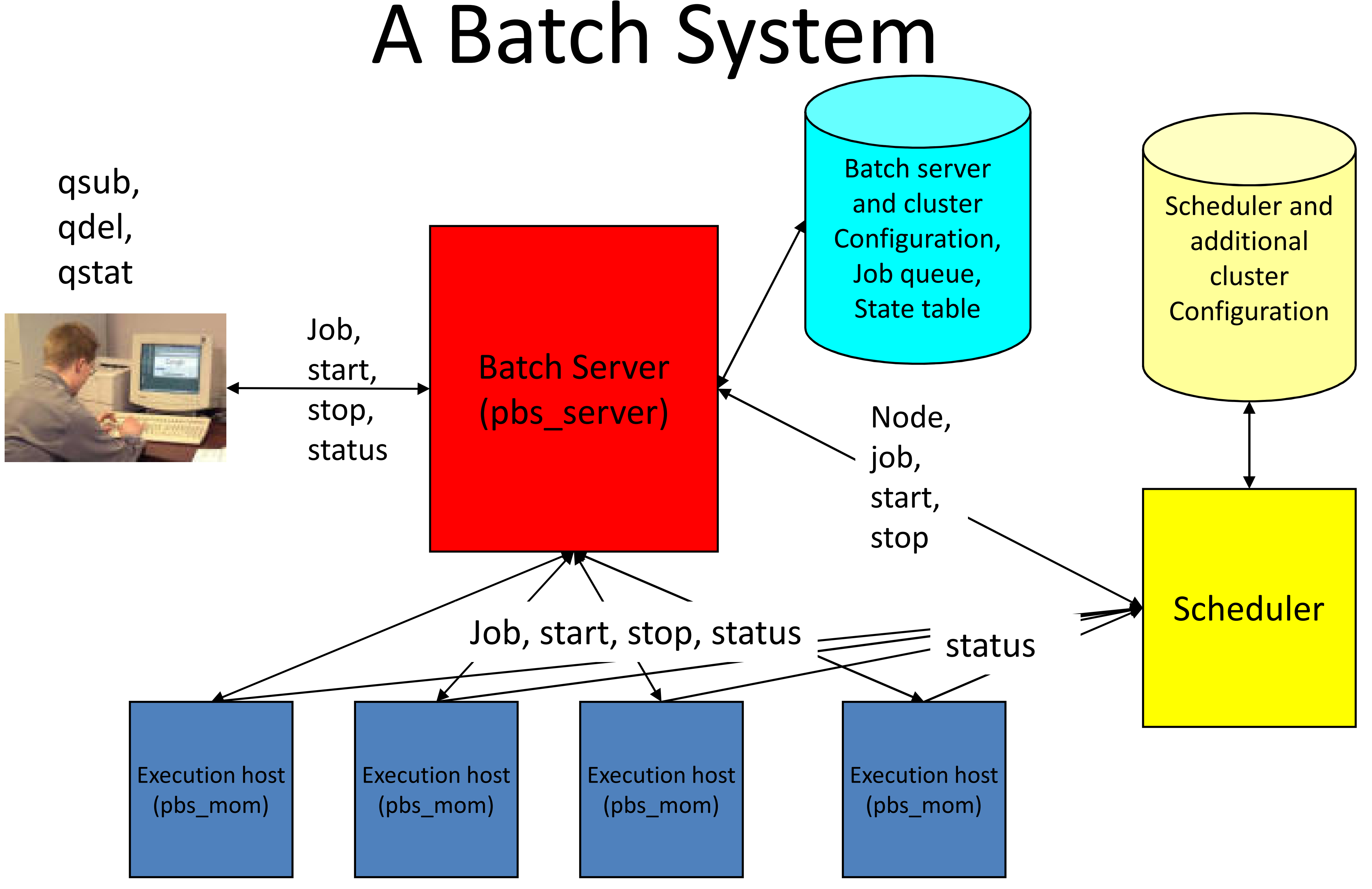
- **PBS (Portable Batch System)** - software system for managing system resources on workstations, SMP systems, MPPs and vector computers. It was based on Network Queuing System (NQS) 1986 NASA – first on Cray and then ported to other architectures

- **TORQUE Resource Manager (Tera-scale Open-source Resource and QUEue manager)** - an open source version of PBS, providing control over:
  - batch jobs
  - distributed compute nodes

Batch Systems

- provide a mechanism for submitting, launching, and tracking jobs on a shared resource
- provide centralized access to distributed resources
- allow users a ‘single system image’ in terms of the management of their jobs and the aggregate compute resources available.

A Batch System



TORQUE Features

- TORQUE provides enhancements over standard OpenPBS in the following areas:
  - ✓scalability
  - ✓fault tolerance
  - ✓scheduling interface
  - ✓usability

TORQUE Benefits

- Initiate and manage serial and parallel batch jobs remotely (create, route, execute, modify and/or delete jobs)
- Define and implement resource policies that determine how much of each resource can be used by a job
- Apply jobs to resources across multiple servers to accelerate job completion time
- Collects information about the nodes within the cluster to determine which are in use and which are available.

PBS Structure

- General components (daemons)
  - A resource manager ***pbs_server***
  - A scheduler ***pbs_sched***
  - Many “executors” ***pbs_mom***
    moms (Machine Oriented Mini-servers)

- PBS provide an API to communicate with the server and another one to interface the moms

PBS Components

- Job server (*pbs_server*)
  - provides the basic batch services
    - receiving/creating a batch job
    - modifying the job
    - protecting the job against system crashes
    - running the job.
    - cancelling a job
  - logs information about jobs for accounting
  - keeps track of all nodes and jobs

PBS Components

- Job Executor (*pbs_mom*)
  - receives a copy of the job from the job server
  - sets the job into execution
  - creates a new session as identical user
  - returns the job's output to the user.
- Job Scheduler (*pbs_sched*)
  - runs site's policy controlling which job is run and where and when it is run

➢ PBS allows each site to use its own scheduler

➢ Currently Maui Scheduler is used in ***NAR***

PBS Job Session

- From a user's point of view
  - Determine resource requirements(CPU time, memory, number of CPUs/node) for a job and write a batch script.
  - Submit the script to the queuing system.
  - Wait for the job to be scheduled and ran.
  - Get the results.

PBS Job Session

- From the queuing system point of view
  - User submits the job with *qsub* command
  - PBS places the job into a queue based on its resource requests and runs the job when those resources become available
  - The job runs until it either completes or exceeds one of its resource request limits
  - PBS copies the job's output into the directory from which the job was submitted and optionally notifies the user via email that the job has ended

PBS Job Workflow

- When a scheduling interval starts
  - Scheduler asks *pbs_server* the state of the nodes and of any jobs.
  - Scheduler query moms for determining available resources (memory, cpu load, etc.)
  - Scheduler examines job queues attempts to schedule any eligible jobs
  - If there are enough resources free, scheduler allocates resources for job, returning job id and resource list to *pbs_server* for execution
  - *pbs_server* contacts the *pbs_mom* on the first node assigned to the job (That *pbs_mom* is called the *mother superior*).
  - The *mother superior* executes the jobs scripts submitted by the user (it also monitors resource usage of child processes and reports back to server.)
- When a PBS heartbeat happens
  - The *pbs_server* will contact each *pbs_mom* and ask the status of its node.

Scheduler Problem

- The default TORQUE scheduler (*pbs_sched*)
  - is very basic
  - will provide poor utilization
  - lacks some important features like
    - backfill scheduler
    - advance reservation

- Solution => MAUI Scheduler

MAUI

- MAUI is a scheduler (not a resource manager!) developed by the MAUI High Performance Computing Center (MHPCC) as an alternative to the default Loadleveler scheduler in their IBM SP (Scalable POWERparallel) environment.
- Ported to PBS by using the appropriate API provided by PBS.

MAUI Features

- Scheduling behavior is constrained by way of throttling policies
  - Both soft and hard limits used
  - Applied to each iteration
- Three main algorithms used
  - Backfill
  - Priority
  - Fairshare
- Reservations for high priority jobs
- More control parameters on users
- Commands for querying the scheduler

MAUI Philosophy

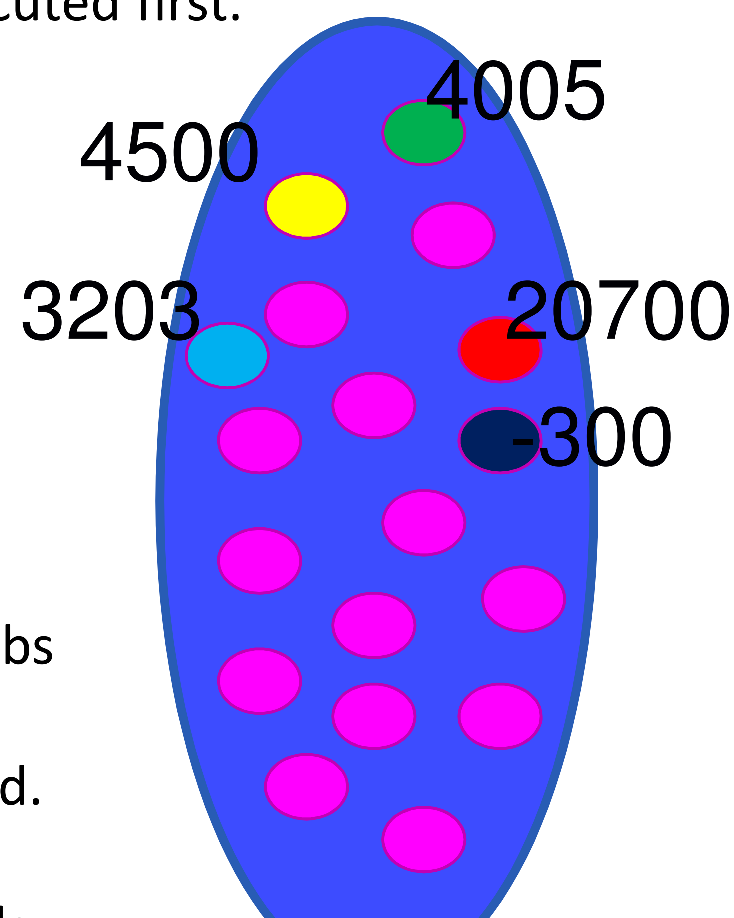
- Maui is particularly concerned about scheduling multiprocessor jobs
- How do you arrange a matching set of processors to be simultaneously available for a single job ?
- Maui tries to plan the execution of such jobs at a particular time when it expects sufficient processors to be available - on the basis of the job maximum **walltime** parameters.
- It establishes **reservations** on a set of processors for a job – ensuring all the processors are free at the planned time

MAUI Philosophy

- As the reservations take effect, more and more processors become idle as the planned job time approaches
- A scheme called **backfill** tries to exploit these idle processors by running short single/few processor jobs *out of priority order* in the gaps
- Maximum efficiency is achieved by **scheduling big jobs first** and running small jobs in the gaps
- Maui really cares about walltimes

Scheduling a Job in MAUI

- Jobs are submitted into a pool of jobs.
- Forget about queues, MAUI considers all jobs.
- Each job has a priority number calculated.
- The highest priority is executed first.



Maui scans through all the jobs and nodes:

- When a job is submitted.
- When a job completes.
- And at periodic intervals.

Scheduling Objects

- MAUI functions by manipulating these five elementary objects

  ✓Jobs
  ✓Nodes
  ✓ Reservations
  ✓ QOS (*Quality of service*) structures
  ✓ Policies

Scheduling Objects

- A job consists of one or more requirements, each of which requests a number of resources of a given type.
- A node is a collection of resources with a particular set of associated attributes.
- Policies are generally specified via a configuration file and serve to control how and when jobs start.
- The MAUI Scheduler allows administrators fine grain control over QOS levels on a per user, group and account basis.

Backfill

- Backfill is a scheduling optimization
- Allows some jobs to be run 'out of order' so long as they do not delay the highest priority jobs in the queue
- Offers significant scheduler performance improvement

Backfill Algorithm

- Uses wallclock limit = an estimation of the wall time (or elapsed time) from job start to job finish to find “holes”(amounts of time dedicated to a job that has already finished or is waiting); it fills these “holes” with execution of other jobs in the queue

- Better estimates of wallclock limit will increase the amount of improvement backfill scheduling can provide for your jobs

Priority Algorithm

- Default is trivial FIFO but is weighted and combined based on a range of job related components
- These components have subcomponents such as user, group, priority, QoS, etc.
- Weight values are configurable parameters and values for each component is calculated from subcomponents listed above

Components of a Job's Priority

- CRED = Credentials, e.g user or group name, submission queue, ...
- FS = Fairshare, e.g considers historical usage of user, group, ...
- RES = Resources, e.g. Number of nodes requested, length of job, ..
- SERV = Service, e.g Time job has been queued,
- TARGET = Target, e.g Jobs must run within two days.
- USAGE = Usage e.g Time consumed by jobs running now.

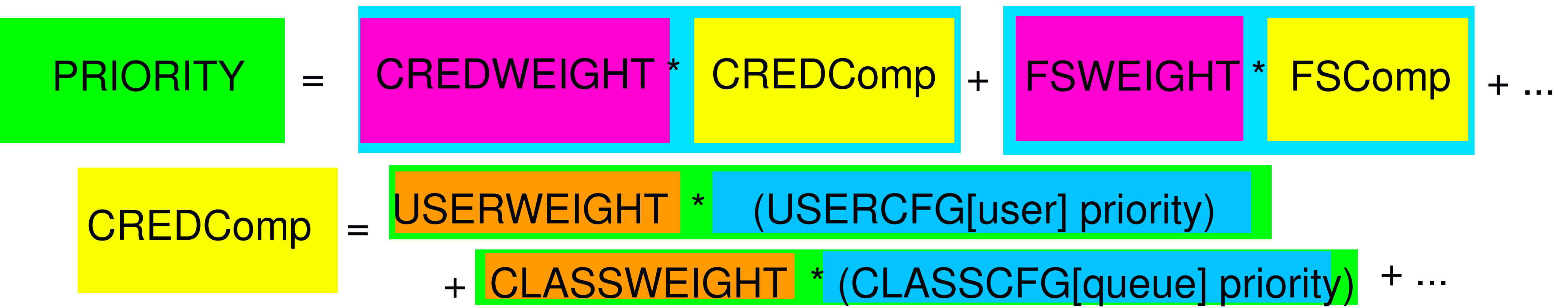
Each component is weighted and summed to form the priority,

- A common mistake is to leave say FSWEIGHT at 0 having configured FS.

Example Subcomponents

- CRED components are static contributions to the overall priority number. e.g username, groupname, submission queue.

Config Attribute	Value	Summary
CREDWEIGHT	10	Component Weight
USERWEIGHT	20	Subcomponent's Weight
USERCFG[user]	PRIORITY=1000	Static Priority for user
CLASSWEIGHT	5	Subcomponent's Weight
CLASSCFG[queue]	PRIORITY=10000	Static Priority for queue



Fairshare

- A mechanism which allows historical resource utilization information to be incorporated into job feasibility and priority decisions
- Composed of several parts which handle historical information: fairshare windows, impact, usage and target
- All parts are configurable parameters
- Purpose of fairshare is to steer existing workload

Fairshare Components

- Fairshare Windows
  - Actually they are timeframes
  - Length and number of windows to be considered while evaluating historical information can be controlled by FSINTERVAL and FSDEPTH parameters
- Impact
  - Defines importance of each window for fairshare evaluation
  - Can be configured using FSDECAY parameter

Fairshare Components

- Usage
  - Defines metric to be used for fairshare evaluation
  - 2 types of usage metrics (controlled by FSPOLICY parameter)
    1. Dedicated usage tracks what the scheduler assigns to the job
    2. Consumed usage tracks what the job actually uses
- Target
  - Expected usage value for a user, group or class (queue) used for fairdhare evaluation
  - Determined and configured by administrator

Example Fairshare Calculation

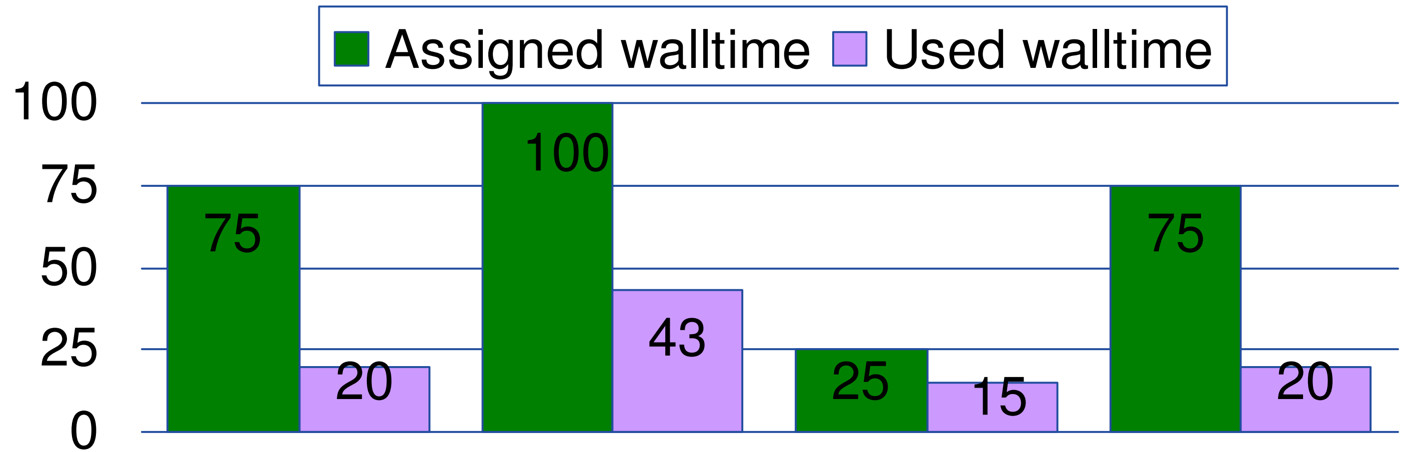
FSPOLICY = consumed usage

$$Usage = \frac{\sum_{i=0}^{DEPTH-1}(U_i * DECAY^i)}{\sum_{i=0}^{DEPTH-1}(T_i * DECAY^i)}$$

FSDEPTH=4

FSDECAY=0.5

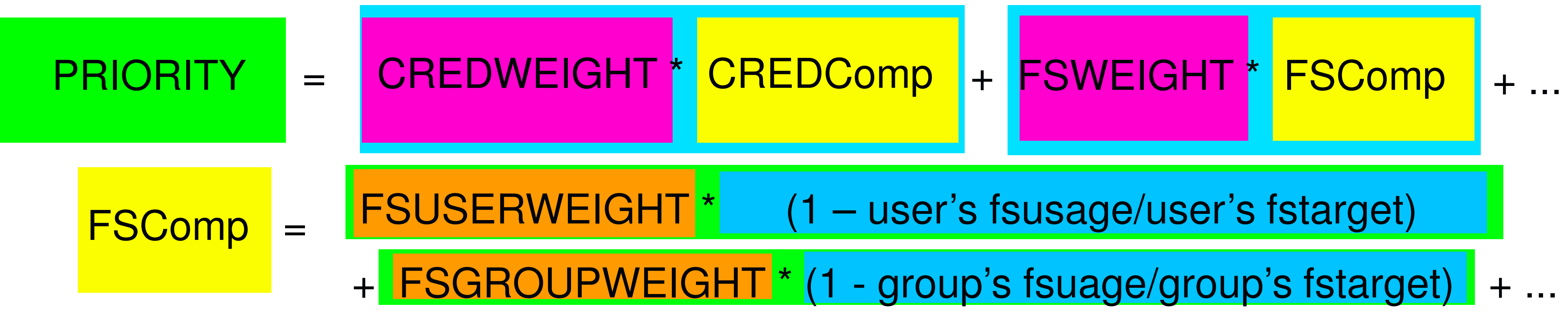
FSINTERVAL=24h



$$USAGE = \frac{20 * 0.5^0 + 43 * 0.5^1 + 15 * 0.5^2 + 20 * 0.5^3}{75 * 0.5^0 + 100 * 0.5^1 + 25 * 0.5^2 + 20 * 0.5^3}$$

Example Fairshare Calculation

- Then, a comparison between target and usage for the user, group or class gives the contribution to the job's overall priority.
- Difference or ratio can be used in this comparison (here ratio is used)



Soft and Hard Limits

- Hard and soft limit specification allow a site to balance both fairness and utilization on a given system
- They provide scheduling flexibility
- Soft limits
  - more constraining limits
  - in effect in heavily loaded situations and reflect tight fairness constraints
- Hard limits
  - more flexible limits
  - specify how flexible the scheduler can be in selecting jobs when there are idle resources available after all jobs meeting the tighter soft limits have been started

Example of Soft and Hard Limits

cenga jobs MAXJOB=2,4

cengb jobs MAXJOB=4,5

Job Slot

BLOCKED	IDLE	RUNNING
	6 5 4 3 2 1	
12 10 9 5	11 8 7 6	4 3 2 1
	12 11 10 9 8 7 5	6 4 3 2
11	12	9 8 7 5

Advance Reservations

- An advance reservation is the mechanism by which MAUI guarantees the availability of a set of resources at a particular time
- Every reservation consists of 3 major components
  - a list of resources
  - a timeframe
  - an access control list
- Additionally, a reservation may also have a number of optional attributes controlling its behavior and interaction with other aspects of scheduling

Advance Reservations

- Access control list (ACL): determines who or what can use the reserved resources.
- While reservation ACL's *allow* particular jobs to utilize reserved resources, they do not *force* the job to utilize these resources. Maui will attempt to locate the best possible combination of available resources whether these are reserved or unreserved.

Preemption

- Basically preemption is pausing a low priority (or time consuming) job when a higher priority (or quickly finishing) job requests some resources
- Can be controlled manually or by using a preemptive backfill policy
- Type of preemption determines how the *PREEMPTEE* (paused/low priority) job continues when the *PREEMPTOR* (high priority) job is finished

Types of preemption

- Job Requeue
  - Preemptee jobs are terminated and returned to the job queue.
- Job Suspend
  - Preemptee jobs stop executing but remain in memory.
  - While a suspended job frees up processor resources, it may continue to consume swap and/or other resources.
- Job Checkpoint
  - Preemptee jobs save off their current states (checkpointing) and terminates.
  - When resources become available again, the checkpointed job is restarted and it resumes execution from its checkpoint.

➢ Resource manager should support it for job checkpointing to work

➢ Torque supports checkpointing by using a kernel level package called BLCR (Berkeley Lab Checkpoint/Restart)

Thank you…

Any questions?