SCALABLE COMPUTING LABORATORY

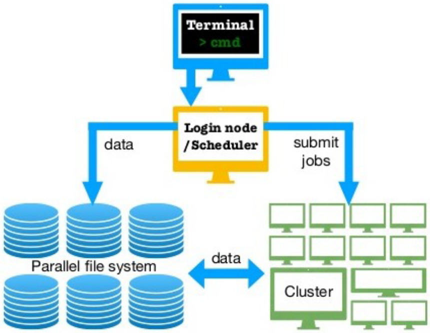
VANDERBILT UNIVERSITY

SC3260 / SC5260

Batch Scheduler

Lecture by: Ana Gainaru

Today



- HPC system middleware
  - Distributed operating system
    - Memory management, processes and communication management
  - Parallel file system
    - Access performance, resiliency, security
  - Scheduler
  - Daemons on compute nodes
    - Performance monitoring, fault tolerance

Today

Batch Scheduling

- From the **user's perspective**
  - Submission principles
  - Performance
- From the **system's perspective**
  - Principles
  - Brief theoretical resutls
  - Currently used schedulers
  - How good is a schedule?

Recap

Why are schedulers needed?

Recap

Why are schedulers needed?

- Performance
- Fairness (every user wants to be on a dedicated machine)

Performance / Fairness

From the system's perspective

Administrators want to keep the system utilized

- Utilization (max) : percentage of the CPU time that is spent computing
- Power consumption (min)
- User fairness : give space on the machine to all users

Performance / Fairness

From the system's perspective

Administrators want to keep the system utilized

- Utilization (max) : percentage of the CPU time that is spent computing
- Power consumption (min)
- User fairness : give space on the machine to all users

From the user's perspective

Users want their job to compute as fast as possible

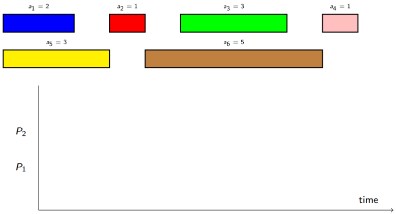
- Makespan (min) : time to complete the job from start to end
- Response time (min) : time to complete the job from submission to end
- Stretch (min) : ration between the response time and the ideal execution time

VANDERBILT UNIVERSITY

Scheduling policies

The scheduler can be used to balance all the metrics

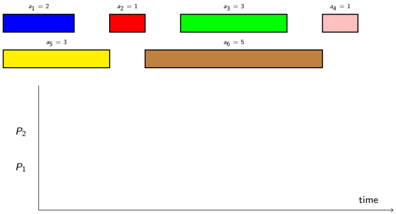
- User fairness
- System utilization
- Application response time



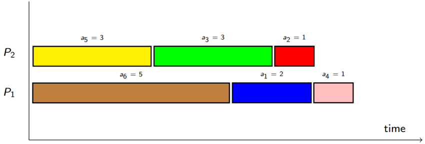
Scheduling policies

The scheduler can be used to balance all the metrics

- User fairness
- System utilization
- Application response time



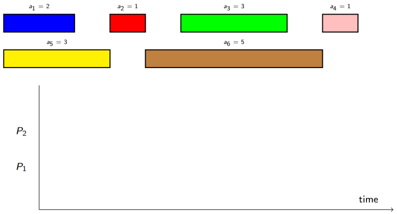
Longest job first



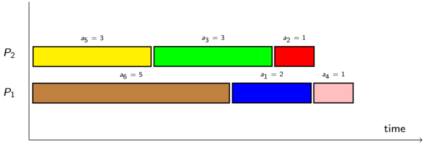
Scheduling policies

The scheduler can be used to balance all the metrics

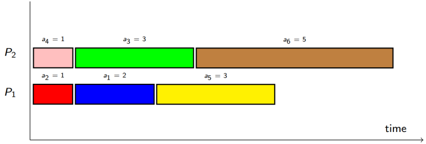
- User fairness
- System utilization
- Application response time



Longest job first



Shortest job first



Batch Schedulers

- Applications in HPC systems are run as batch jobs, i.e. time-limited requests for resources to run the application binaries.
- Once an application is submitted on a cluster it becomes a job
- Each job is defined as a **Number of nodes** (p_i) and a **Time** (t_i)
  `I want 6 nodes for 1h`

Typically users are charged against an allocation: e.g. "You only get 100 CPU hours per week"

A batch scheduler is a central middleware to manage resources (e.g. processors)

- accept jobs (computing tasks) submitted by users
- decide when and where jobs are executed
- start jobs execution

Batch Schedulers

Schedulers take into account:

- unavailability of some nodes
- users jobs mutual exclusion
- specific needs for jobs (memory, network, ...)

While trying to :

- maximize resources usage
- be fair among users

To run multiple applications concurrently, **HPC schedulers order the execution of batch jobs** to achieve high utilization while controlling their turnaround times

Batch Schedulers

Typical wanted features:

- Interactive mode
- Batch mode
- Parallel jobs support
- Multi-queues with priorities
- Reservations
- Admission policies (limit on usage, notions of user groups)
- Resources matching
- File staging
- Jobs dependences
- Backfilling
- Environment reconfiguration

There are many existing batch schedulers: Slurm, LSF, Moab, PBS/Torque, EASY, OAR, ...

These are complex systems with many config options !

Batch Scheduler Components

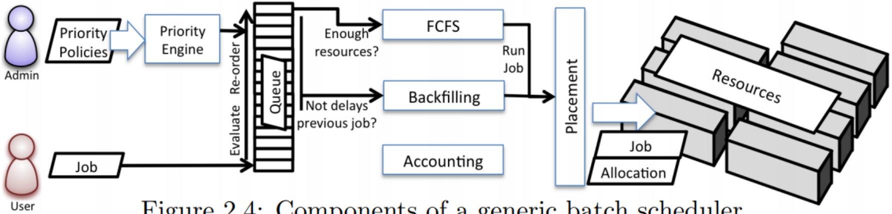


Figure 2.4: Components of a generic batch scheduler.

Life-cycle of a batch job

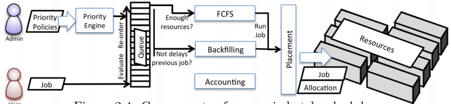


Figure 2.4: Components of a generic batch scheduler.

1) Job submission to the system

The job submission must provide

- Detailed specification of the requested resources (e.g. the number of cores, minimum RAM per core, or specific compute nodes to run on)
- An estimate of the job's runtime
- A priority request (expressing the job's importance)
- Optionally, a list of dependencies on other jobs (e.g. statements that the job should not start until some set of conditions is met)

Life-cycle of a batch job

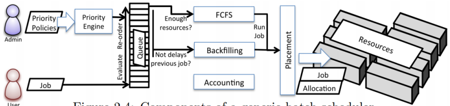


Figure 2.4: Components of a generic batch scheduler.

2) The scheduler contacts the resource manager

- If no other jobs are waiting and there are enough resources available, the scheduler runs the job immediately
- If there are holes in a schedule that would fit the current job, the job is ran immediately (backfilling)
- Otherwise, the job is appended to a job waiting queue

Life-cycle of a batch job

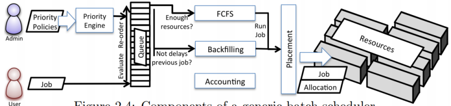


Figure 2.4: Components of a generic batch scheduler.

3) Jobs placed in the waiting queue

- Jobs in the waiting queue are initially ordered by arrival time
- Jobs are ranked and re-ordered by a priority engine
- Different ranking policies define priorities, based on
  - job size (e.g. smaller jobs should run sooner)
  - priority class (e.g. jobs in the real time class should run before any other job)
  - fairness (i.e. priorities dictated by system quotas)
  - wait time in the queue (jobs that have been waiting a long time to start)
  - other administrator-defined criteria

Life-cycle of a batch job

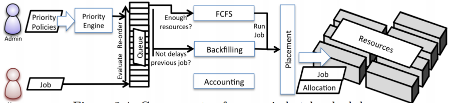


Figure 2.4: Components of a generic batch scheduler.

4) Jobs are allocated on the compute nodes

- Jobs progress towards the top area of the waiting queue until they are extracted by the scheduling algorithms and then executed
- Online or reservation-based placement decision
- Scheduler informs the resource manager about the new placement

Most HPC batch schedulers include the FCFS (First Come First Served) and backfilling scheduling algorithms with different priority re-ordering

Online / Reservation-based

There are usually many jobs in the queue waiting for resources to become available

Online Scheduler

- When a job finishes, the scheduler chooses the first job in the queue to execute that fits the available resources
- To make sure that large jobs do not starve, the scheduler divided all jobs in the queue in batches
- **Advantage** Easy to implement, fast, the resource requests of jobs don't need to be accurate
- **Disadvantage** Local optimal execution, not the best utilization nor makespan

Online / Reservation-based

There are usually many jobs in the queue waiting for resources to become available

Reservation-based Scheduler

- On job arrival and when a job finishes, the scheduler computes tentative start times for all (most of) the jobs in the queue in order to maximize utilization. **These start times are called reservations**
- Jobs start within their assigned reservations
- **Advantage** Gives the best job placements, fair and starvation free algorithm
- **Disadvantage** More complex and slower (cut of in the waiting queue), resource requests must reflect resource usage

Online / Reservation-based

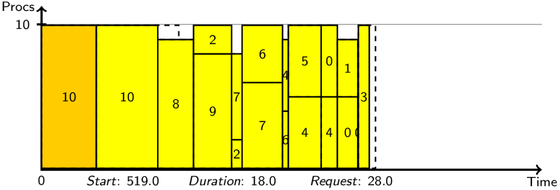
There are usually many jobs in the queue waiting for resources to become available

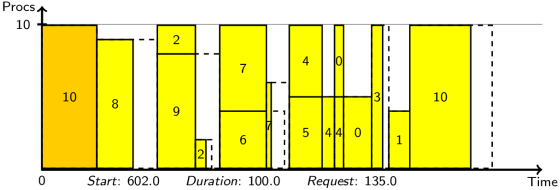
Reservation-based Scheduler

- On job arrival and when a job finishes, the scheduler computes tentative start times for all (most of) the jobs in the queue in order to maximize utilization. **These start times are called reservations**
- Jobs start within their assigned reservations
- **Advantage** Gives the best job placements, fair and starvation free algorithm
- **Disadvantage** More complex and slower (cut of in the waiting queue), resource requests must reflect resource usage

Most schedulers are reservation-based using priority queues and backfilling

Online / Reservation-based





Placement of 11 jobs using online or reservation-based strategies.

- The reservations are computed only during job arrival and not job ending

Stochastic jobs will get better results from online schedulers

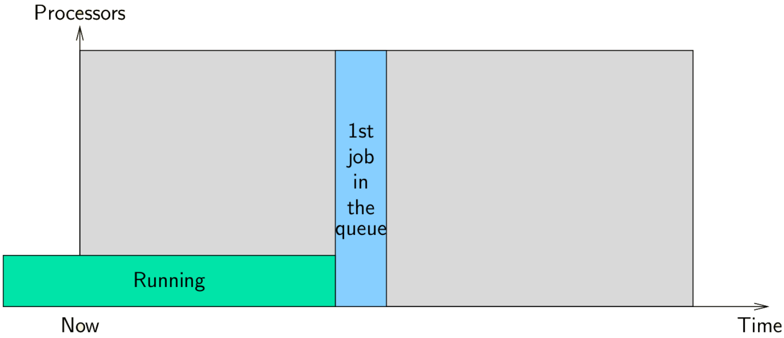
VANDERBILT UNIVERSITY

Online / Reservation-based

- FCFS = simplest scheduling option
- Fragmentation = need for backfilling

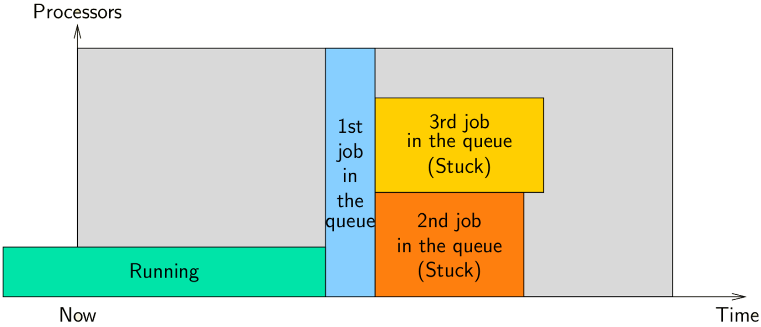
Online / Reservation-based

- FCFS = simplest scheduling option
- Fragmentation = need for backfilling



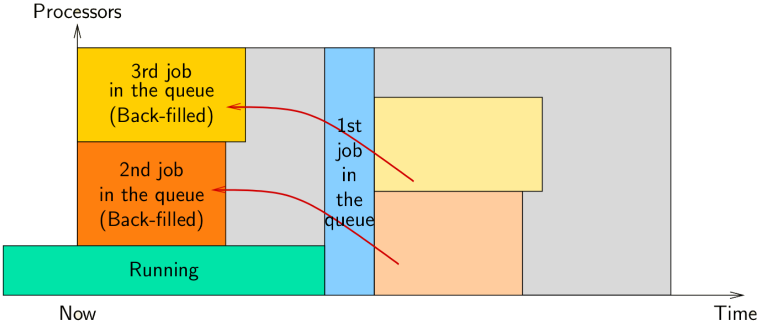
Online / Reservation-based

- FCFS = simplest scheduling option
- **Fragmentation** = need for backfilling



Online / Reservation-based

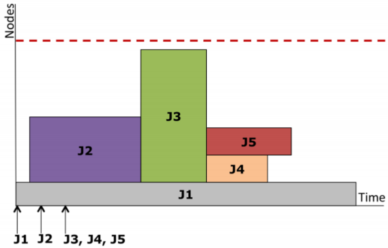
- FCFS = simplest scheduling option
- Fragmentation = need for backfilling

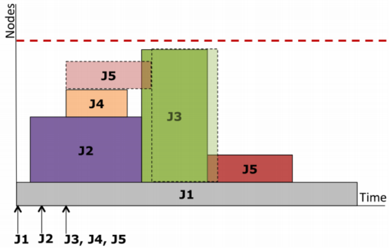


Processors
3rd job in the queue (Back-filled)
2nd job in the queue (Back-filled)
1st job in the queue
Running
Now
Time

Online / Reservation-based

- FCFS = simplest scheduling option
- Fragmentation = need for backfilling





Additional functions

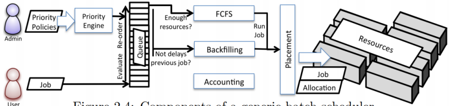


Figure 2.4: Components of a generic batch scheduler.

Mechanisms that are needed to manage an HPC system

- Placement systems that calculate which resources should be used for specific jobs.
  - These decisions take into account the network topology or special job requirements
  - Example: in a system with a fat-tree interconnect topology, a tightly coupled application will run faster if all its assigned nodes are leaves pending from same network switch
- Workload managers include functions to handle the basic operations to run an HPC system, such as managing the compute resources, staging-in jobs, controlling their execution, and staging-out resources

Additional functions

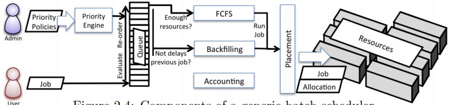


Figure 2.4: Components of a generic batch scheduler.

HPC Accounting for registering the use of compute hours and resources by user jobs

- Prevent users from utilizing the system beyond their assigned quota (e.g. by de-prioritizing their jobs)
- Encourage those who have not used it (e.g. by elevating the priority of users with little quota usage)

Backfilling policies

Which job(s) should be picked for promotion through the queue?

Backfilling policies

Which job(s) should be picked for promotion through the queue?

- Many heuristics are possible
- Two have been studied in detail
  - EASY
  - Conservative Back Filling (CBF)
- In practice EASY is used in almost all current schedulers
- The OAR scheduler (used by french clusters) uses CBF

EASY Backfilling

Extensible Argonne Scheduling System

Maintain only one reservation, for the first job in the queue.

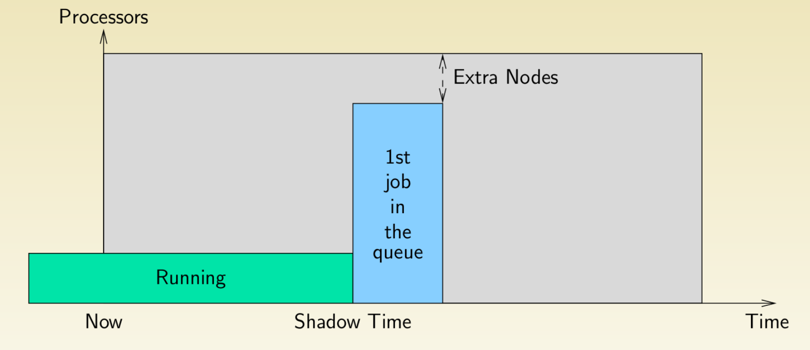
Definitions:

- **Shadow time** time at which the first job in the queue starts execution
- **Extra nodes** number of nodes idle when the first job in the queue starts execution

Policy

1. Go through the queue in order starting with the 2nd job.
2. Backfill a job if it will terminate by the shadow time, or it needs less than the extra nodes.

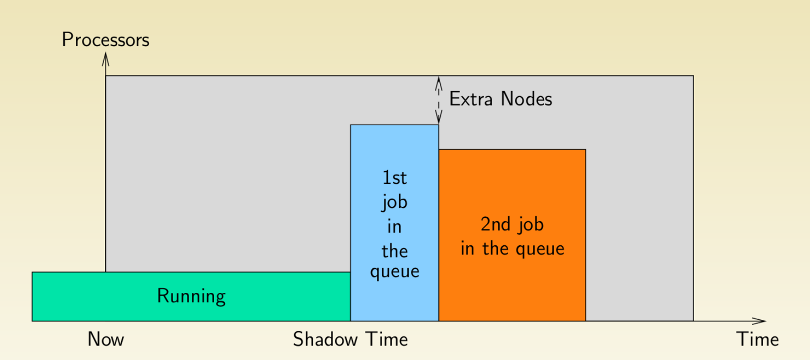
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs

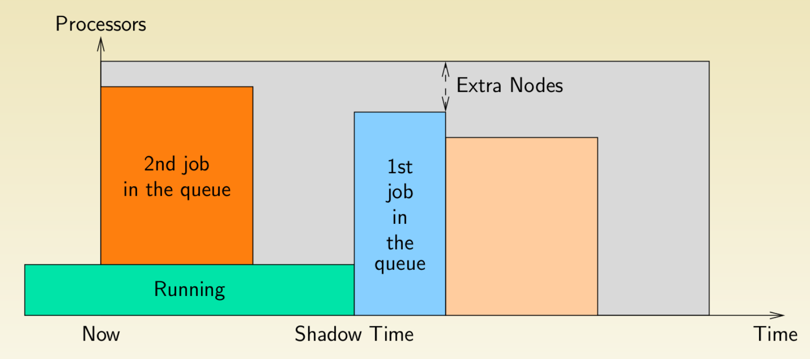
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs

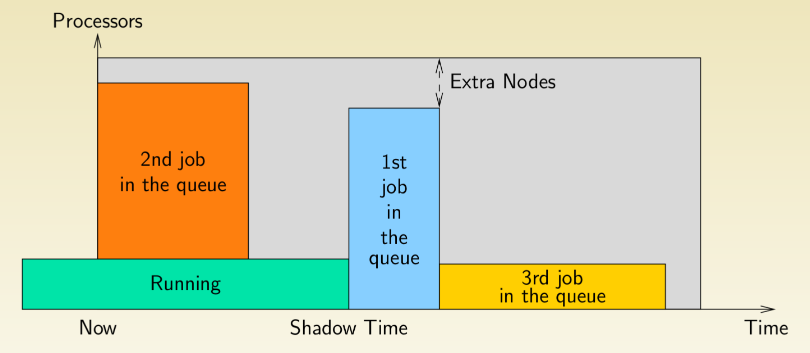
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs

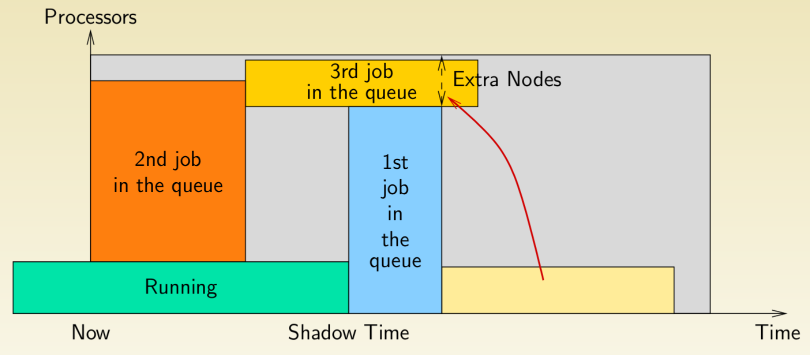
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs

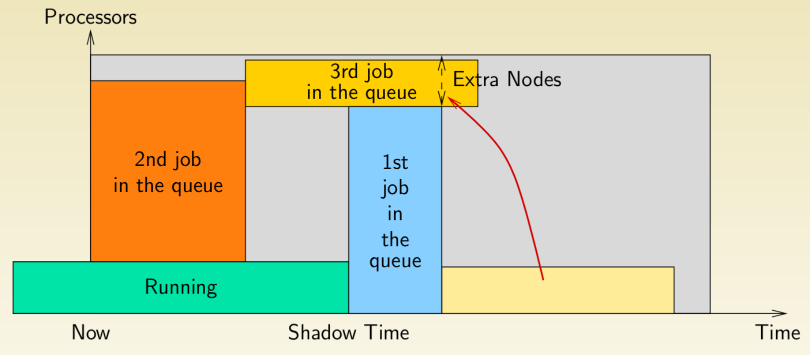
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs

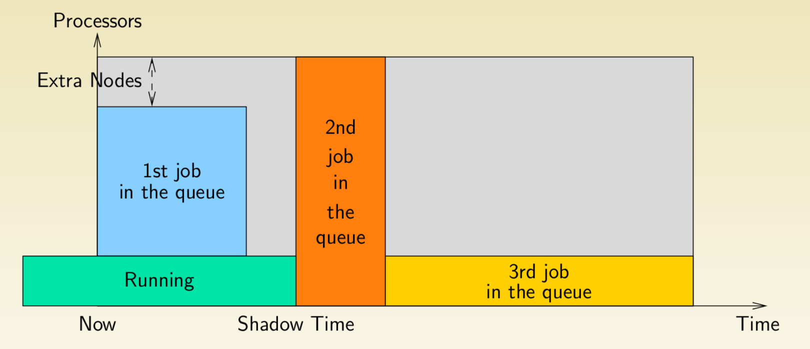
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

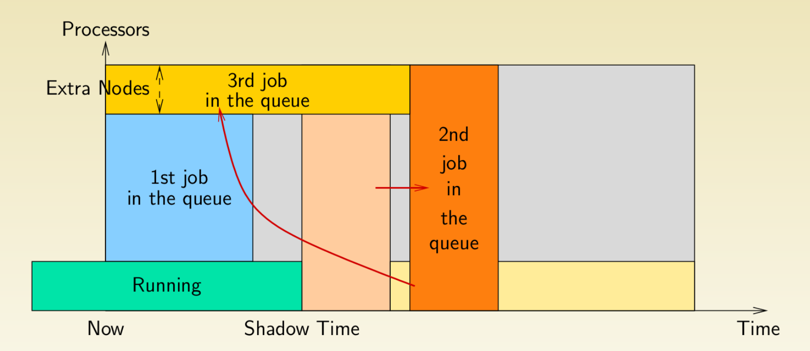
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

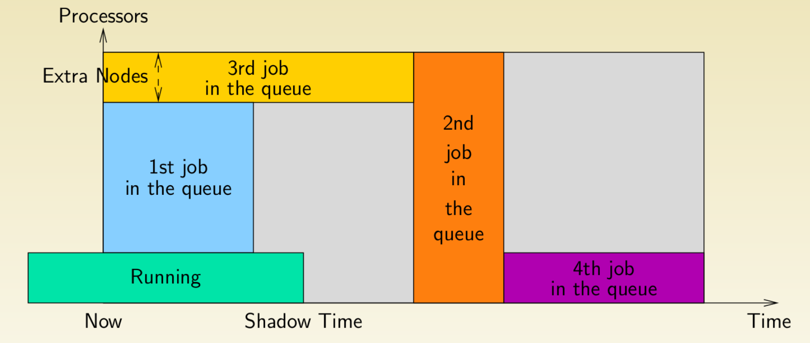
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

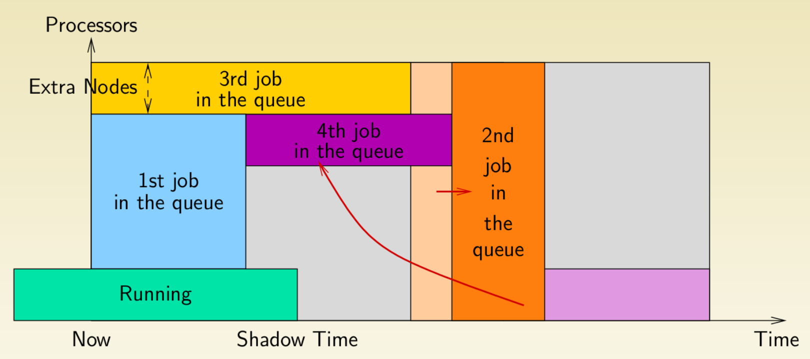
EASY



Property

- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

EASY



Property

- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

EASY Properties

Unbounded Delay

- The first job in the queue will never be de- layed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

No starvation

- Delay of first job is bounded by runtime of current jobs
- When the first job finishes, the second job becomes the first job in the queue
- Once it is the first job, it cannot be delayed further

Other backfilling approach

Conservative Backfilling

- EVERY job has a reservation. A job may be backfilled only if it does not delay any other job ahead of it in the queue
- Fixes the unbounded delay problem that EASY has. More complicated to implement (The algorithm must find holes in the schedule) though.
- EASY favors small long jobs and harms large short jobs.

Performance metrics

How Good is the Schedule?

When does backfilling happen?

Possibly when

- A new job arrives
- The first job in the queue starts
- When a job finishes early

When does backfilling happen?

Possibly when

- A new job arrives
- The first job in the queue starts
- When a job finishes early

Users provide job **runtime estimates** (Jobs are killed if they go over). Trade-off:

- provide a **conservative estimate**: goes through the queue faster (may be backfilled)
- provide a **loose estimate**: your job will not be killed

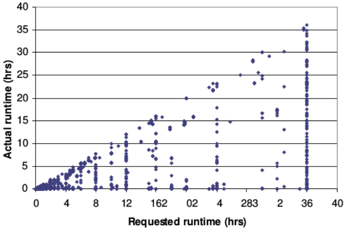
When does backfilling happen?

Possibly when

- A new job arrives
- The first job in the queue starts
- When a job finishes early

Users provide job **runtime estimates** (Jobs are killed if they go over). Trade-off:

- provide a **conservative estimate**: goes through the queue faster (may be backfilled)
- provide a **loose estimate**: your job will not be killed



Measure performance

... but how do we know what a "good" schedule is? FCFS, EASY, CFB, Random?

What we need are metrics to quantify how good a schedule is. It has to be an aggregate metric over all jobs

Measure performance

... but how do we know what a "good" schedule is? FCFS, EASY, CFB, Random?

What we need are metrics to quantify how good a schedule is. It has to be an aggregate metric over all jobs

1. **Turn-around time or flow (Wait time + Run time)**
   Job 1 needs 1h of compute time and waits 1s
   Job 2 needs 1s of compute time and waits 1h
   Clearly Job 1 is really happy, and Job 2 is not happy at all

Measure performance

... but how do we know what a "good" schedule is? FCFS, EASY, CFB, Random?

What we need are metrics to quantify how good a schedule is. It has to be an aggregate metric over all jobs

1. **Turn-around time or flow (Wait time + Run time)**
   Job 1 needs 1h of compute time and waits 1s
   Job 2 needs 1s of compute time and waits 1h
   Clearly Job 1 is really happy, and Job 2 is not happy at all

2. **Wait time** (equivalent to "user happiness")
   Job 1 asks for 1 nodes and waits 1 h
   Job 2 asks for 512 nodes and waits 1h
   Again, Job 1 is unhappy while Job 2 is probably sort of happy.

We need a metric that represents happiness for small, large, short, long jobs

Measure performance

- Slowdown or Stretch (turn-around time divided by turn-around time if alone in the system)
  Doesn't really take care of the small/large problem.
  Could think of some scaling, but unclear !

For now this is all we have

Measure performance

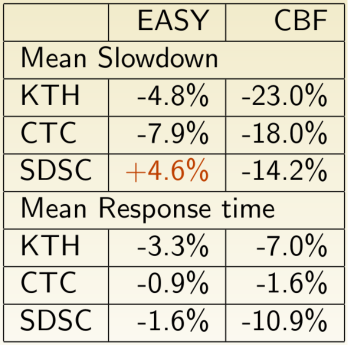
- Slowdown or Stretch (turn-around time divided by turn-around time if alone in the system)
  Doesn't really take care of the small/large problem.
  Could think of some scaling, but unclear !

For now this is all we have We can run simulations of the scheduling algorithms, and see how they fare. We need to test these algorithms in representative scenarios Supercomputer/cluster traces. Collect the following for long periods of time:

- Time of submission
- How many nodes asked
- How much time asked
- How much time was actually used
- How much time spent in the queue

Measure performance

Example experiment: replace user estimate by f times the actual run time Possible to improve performance by multiplying user estimates by 2!



	EASY	CBF
Mean Slowdown		
KTH	-4.8%	-23.0%
CTC	-7.9%	-18.0%
SDSC	+4.6%	-14.2%
Mean Response time		
KTH	-3.3%	-7.0%
CTC	-0.9%	-1.6%
SDSC	-1.6%	-10.9%

Performance of Schedulers

- All the schedulers presented are all heuristics
  - They are not specifically designed to optimize the metrics we have designed
- It is difficult to truly understand the reasons for the results.
- But one can derive some empirical wisdom.

- One of the reasons why one is stuck with possibly obscure heuristics is that we're dealing with an on-line problem
- We cannot wait for all jobs to be submitted to make a decision
- But we can wait for a while, accumulate jobs, and schedule them together.

Summary

Batch Schedulers are what we're stuck with at the moment

They are often hated by users

- I submit to the queue asking for 10 nodes for 1 hour.
- I wait for two days.
- My code finally starts, but doesn't finish within 1 hour and gets killed!!

Summary

Batch Schedulers are what we're stuck with at the moment

They are often hated by users

- I submit to the queue asking for 10 nodes for 1 hour.
- I wait for two days.
- My code finally starts, but doesn't finish within 1 hour and gets killed!!

A lot of research (and theoretical results), a few things happening "in the field".

**When you go to a company that has clusters (like most of them),
they typically have a job scheduler, so it's good to have some idea of what it is.**

Next

1. SLURM and how to use it
2. A few promising directions for the future
   - Gang scheduling
   - Task based scheduler (work stealing)

Further Reading

Book on the theory of scheduling

D.B. Shmoys, J. Wein, and D.P. Williamson. *Scheduling parallel machines on-line* Symposium on Foundations of Computer Science, 0:131-140, 1991.

Figures from today's slides courtesy of Arnaud Legrand and Guillaume Pallez
`http://people.bordeaux.inria.fr/gaupy/ressources/teachings/2019/algo_hpc/`