
Lecture 12: Multiprocessor Scheduling II

CSC 469H1F

Fall 2006

Angela Demke Brown



Parallel Job Scheduling

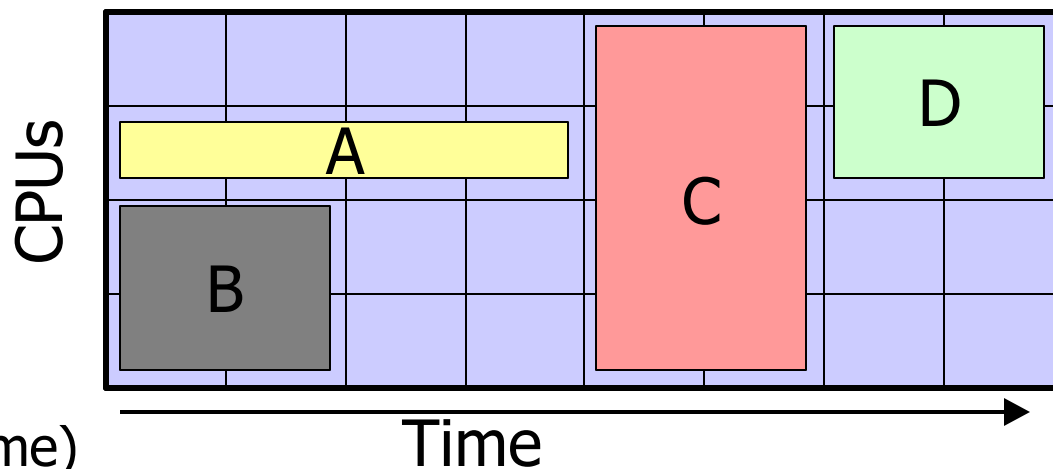
- Recall threads in a parallel job are not independent
 - Scheduling them as if they were leads to performance problems
 - Want scheduler to be aware of dependences
- Forms of scheduler-awareness
 - Know threads are related, schedule all at same time
 - Know when threads hold spinlocks and don't deschedule lock holder
 - Know about general dependences

Using thread relations

- Space sharing (typically supercomputers)
 - At job creation, specify number of threads
 - Scheduler finds set of CPUs
 - May negotiate with application
 - "I can't get you 512 CPUs right now, would you like to wait or run with only 8?"
 - Many parallel applications can choose the # of threads
- How should scheduler choose jobs to assign to CPUs?
What is optimal?
 - Uniprocessor scheduling → shortest job first (shortest expected next CPU burst)
 - MP version → smallest expected number of CPU cycles (cycles == num_cpus * runtime)
 - In practice, this info is rarely available
 - FCFS is hard to beat

Limits of FCFS (Space Sharing)

- Scheduling convoy effect
 - Long average wait times due to large job
 - Exists with FCFS uniprocessor batch systems
 - Much worse in parallel systems
 - Fragmentation of CPU space

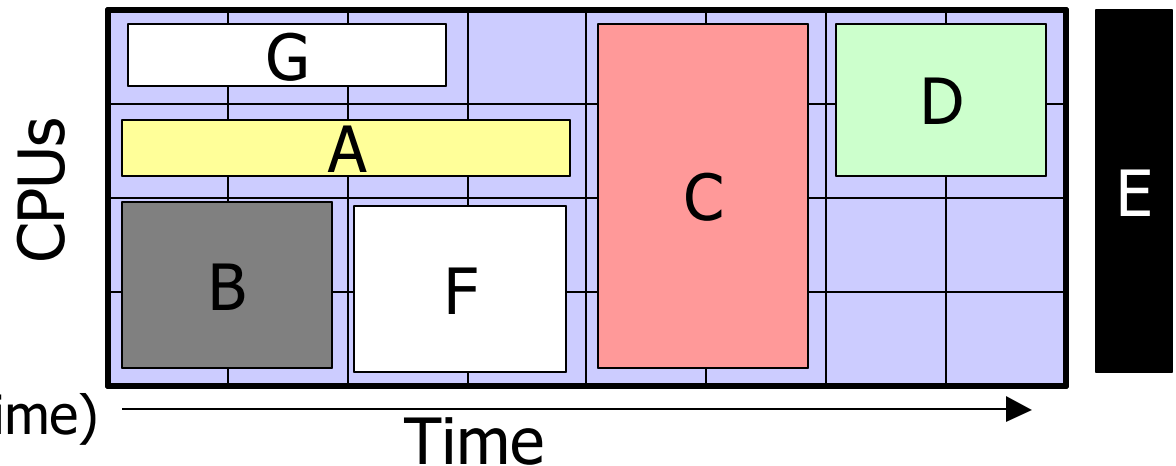


Scheduler queue (CPUs, time)

1,4	1,3	2,2	4,1
-----	-----	-----	-----

Solution: Backfilling

- Fill holes from queue in FCFS order
- Not FCFS anymore
- Want to prevent "fill" from delaying threads that were in queue earlier
 - EASY (argonne national lab scheduler)
 - Make reservation for next job in queue



Scheduler queue (CPUs, time)

		1,4	4,1
--	--	-----	-----

Variations on Backfilling

- **EASY**
 - Used FCFS to order jobs in queue
 - made reservation for first blocked job in queue
 - Backfilled jobs by looking at queue one at a time
- **Ordering alternative: include priority in queue**
 - administrative to distinguish between users
 - user to distinguish between own jobs
 - Scheduler to prevent starvation
- **Reservation alternatives**
 - All queued jobs get a reservation (too much can go wrong)
 - Queued job gets a reservation if it has been waiting more than a threshold
- **Queue lookahead**
 - Use dynamic programming to determine optimal packing

Using Thread Relations II

- Co-scheduling (Ousterhout, 1982)
 - Identify "working set" of processes (analogous to working set of memory pages) that need to run together
- Gang scheduling
 - All-or-nothing → co-scheduled working set is all threads in the job
 - Scheduling benefits of dedicated machine
 - Allows all jobs to get service

Gang Scheduling Issues

- All CPUs must context switch together
 - To avoid fragmentation, construct groups of jobs that fill a slot on each CPU
 - E.g., 8-CPU system, group one 4-thread job with two 2-thread jobs
 - Inflexible
 - If 4-thread job blocks, should be block entire group, or schedule group and leave 4 CPUs idle?
- Alternative 1: Paired gang scheduling
 - Identify groupings with complementary characteristics and pair them. When one blocks, the other runs
- Alternative 2: Only use gang scheduling for thread groups that benefit
 - Fill holes in schedule with any single runnable thread from those remaining

Knowing about Spinlocks

- thread acquiring spinlock sets kernel-visible flag
- Clears flag on release
- Scheduler will not immediately deschedule a thread with the flag set
 - Gives thread a chance to complete critical section and release lock
 - Spinlock-protected critical sections are (supposed to be) short
 - Does not defer scheduling indefinitely

Knowing General Dependences

- Implicit Co-scheduling (Arpaci-Dusseau et al.)
- Designed for workstation cluster environment
 - Explicit messages for all communication/synchronization
 - MUCH more expensive if remote process is not running when local process needs to synchronize
- Communicating processes decide when it is beneficial to run
 - Infer remote state by observing local events
 - Message round-trip time
 - Message arrival
- Local scheduler uses communication info in calculating priority