Preemptive Parallel Job Scheduling for Heterogeneous Systems Supporting Urgent Computing

Presented by
Lizzy Shakman & Oscar Gardella

Urgent Computing

- Used for tasks that have real life time limitations (eg. earthquakes, tsunamis, etc.)
- In UC a computation operates under a strict deadline after which computation results are useless
- Typically UC uses dedicated resources
  - Pros: Always has resources available
  - Cons: Wastes processors most of the time, no backup plan
- Requires a job scheduling method that can handle normal and urgent jobs
  - Without this can either delay normal jobs significantly, or potentially make urgent jobs wait
- Oftentimes preemption is used to allow urgent jobs to “jump ahead”

What is Preemption?

- Preemption is defined as “the interruption or replacement of a scheduled program.”
  - Essentially preemption tells the machine that there is a job that needs to be done immediately, forcing it to pause or quite enough current jobs to make it run
- Preemption can delay urgent jobs due to having to save intermediate state of the preempted jobs (which is bad for such time constrained jobs)
- Becomes more challenging with integration between coprocessors devices and the host processor
  - Forces you to move memory of preempted jobs to host, which is called *Process Swapping*

Proposed Job Scheduling Method

- Partial Process Swapping (PPS): Preemption mechanism for heterogeneous systems using in-memory process swapping
- Urgent Job First with Backfilling (UJFB): Preemption-based job scheduling algorithm
- By swapping processes to memory lower preemption delays can be achieved
- Additionally it decreases the response time and slow down
  - Response Time: Total “Wall clock time” from job submission to its completion
  - Slowdown: Response time of the job normalized by the running time
    - When slowdown = 1, the job executes without any delays, with slowdown > 1 being worse

$$Response = T_w + T_r, \quad (1)$$

$$Slowdown = \frac{Response}{T_r}, \quad (2)$$

Preemptive Job Scheduling for UC

- Proposed job scheduling method that addresses in shared heterogenous UC systems
  - 1) A preemption mechanism using in-memory process swapping
  - 2) A preemption-based job scheduling algorithm for preventing significant delays of urgent jobs
- Kill/Restart: Preempting a job fully kills the job, and starts it again later
  - Dead jobs return to scheduler queue - Partial results are lost - Time is wasted
- Suspend/Resume: Preempting a job is paused, and started again later
  - Partial results are saved -Takes time to save results in different memory, called *Swapping Time*
- Preemption Delay (PD): Longest swapping time among the processes

$$PD = \max_{1 \le p \le N_p} S_p.$$

Partial Process Swapping (PPS)

- Has two system functions that save/restore VE memory of suspended VE processes.
  - Swap Out: Releases VE memory by saving part of it to VH memory
  - Swap In: Restores the memory saved in VH memory to VE memory so that execution can resume
- Preempted jobs need to resume on the same VE it was preempted from
  - PPS makes VE memory return to its VH, making it necessary to resume processes in the same cluster
  - PPS only partially swaps VE memory, meaning that processes must resume on their original VE

- *VE: Virtual engine*
- *VH: Virtual host*

Urgent Job First with Backfilling (UJFB)

- UJFB aims to prevent urgent jobs being delayed while also reducing response time and slowdown for regular jobs
- Executes an urgent job by performing one of the following three subtasks *(fig 5)*:
  - 1) Prioritize urgent jobs to be executed before regular jobs
  - 2) Backfill urgent jobs
  - 3) Preempt regular jobs to give resources to urgent jobs
- UJFB backfills like conservative where all urgent jobs backfill before regular jobs
- Urgent Queue: Only used if there are multiple urgent jobs can’t simultaneously run

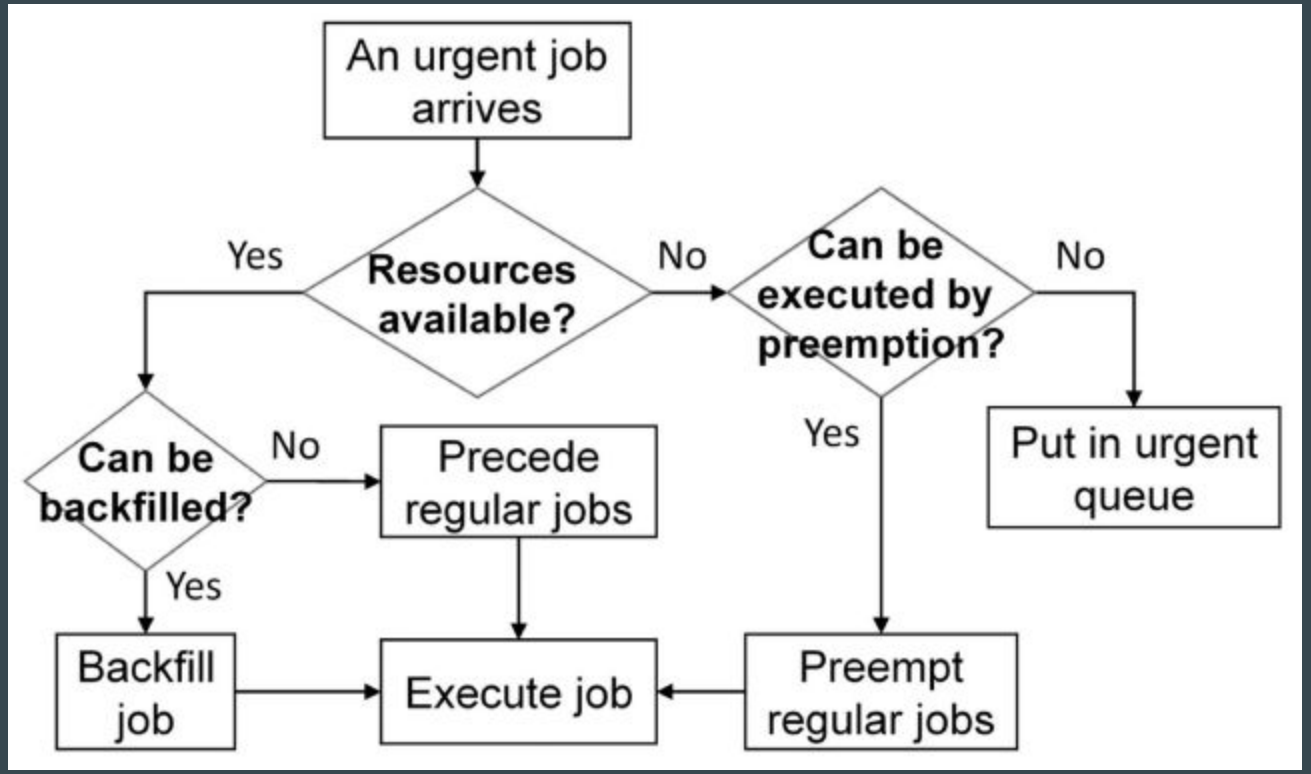


Figure 5. UJFB algorithm for executing urgent jobs.

Algorithm 1 The UJFB Algorithm

```
Input: urgentQ                                  ▷ Queue of urgent jobs
Input: regularQ                                 ▷ Queue of regular jobs
Input: runningJobs                              ▷ Currently running jobs
 1: procedure onArrival(job)
 2:     if job is urgent then                   ▷ An urgent job
 3:         if canExecute(job) then             ▷ Resources are
    available
 4:             if canBackfill(job) then        ▷ Backfilling
 5:                 backfill(job)
 6:             else                            ▷ Precede regular jobs
 7:                 execute(job)
 8:             end if
 9:         else if canRunWithPreempt(job) then
10:             pJobs ← selectJobs(runningJobs)
11:             preempt(pJobs)                  ▷ Preemption
12:             execute(job)
13:         else
14:             enqueue(urgentQ, job)
15:         end if
16:     else                                    ▷ A regular job
17:         enqueue(regularQ, job)
18:     end if
19: end procedure
```

Urgent Lateness (U_L)

- Basic wait time and response time metrics aren't useful for urgent jobs
- Used to evaluate the effects of a job scheduling method on urgent jobs

1. Calculate the slowdown of each urgent job (*Equation 2)*
2. Find the maximum slowdown among urgent jobs (*Equation 4)*

$$U_L = \max_{1 \le u \le N_U} Slowdown_u,$$

Tsunami Simulation

Number of CPUs	Job length (minutes)	Maximum slowdown
128	10	1
256	6.5	1.54
512	4	2.5

Table 1. The number of processors and lengths of urgent jobs for the tsunami simulation.

Tsunami Simulation

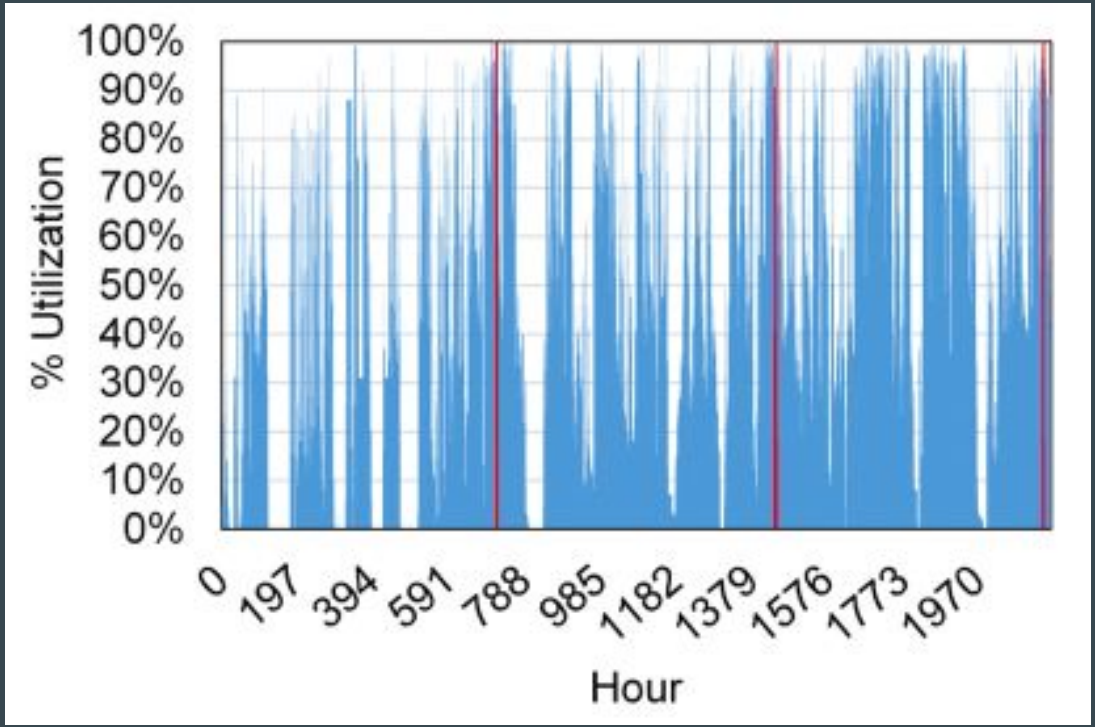


Figure 2. Arrival times of injected urgent jobs.

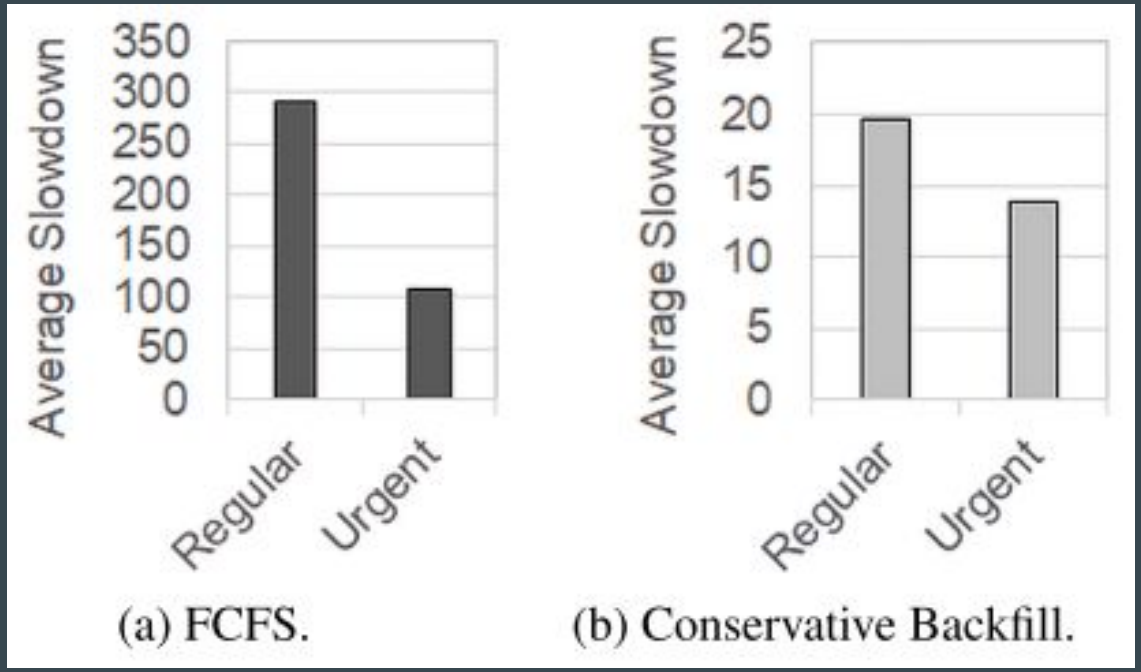
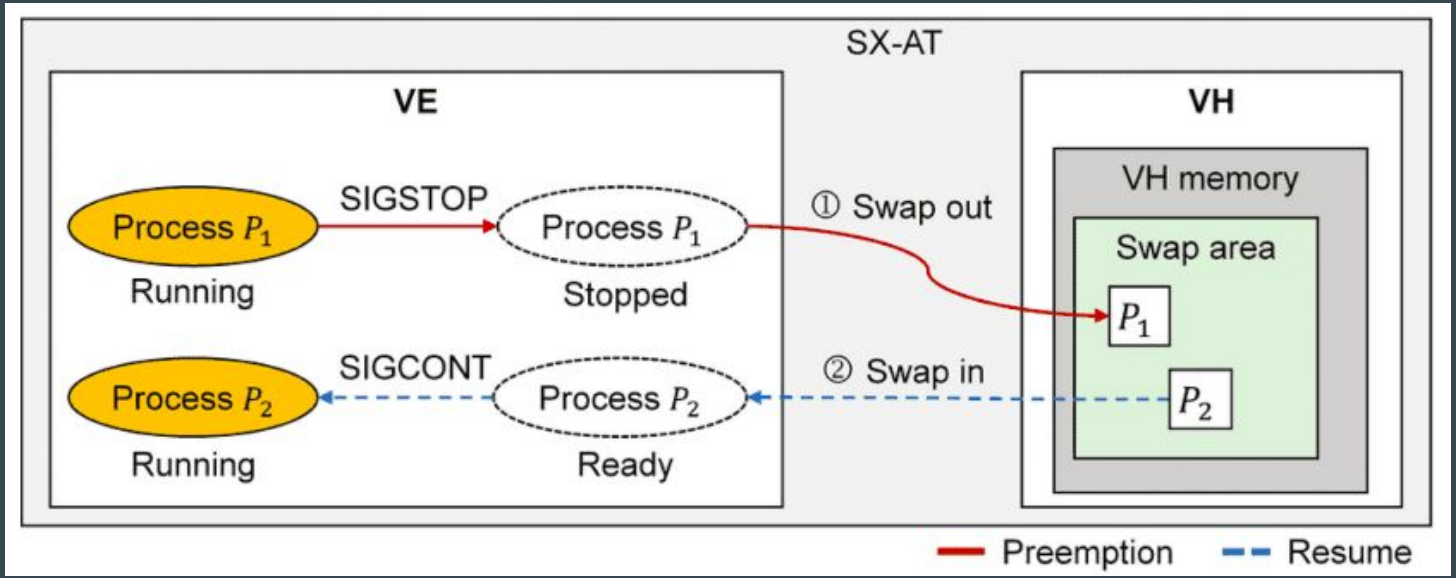


Figure 3. Slowdowns of regular and urgent jobs.

Partial Process Swapping (PPS)



- Red lines: the flow of preempting a job process using PPS
- Blue lines: the flow of resuming a job after it is preempted

Figure 4. Preemption using PPS.

In UJFB, urgent jobs are executed under three cases:

The urgent job is executed by preceding regular jobs that are waiting in the queue.

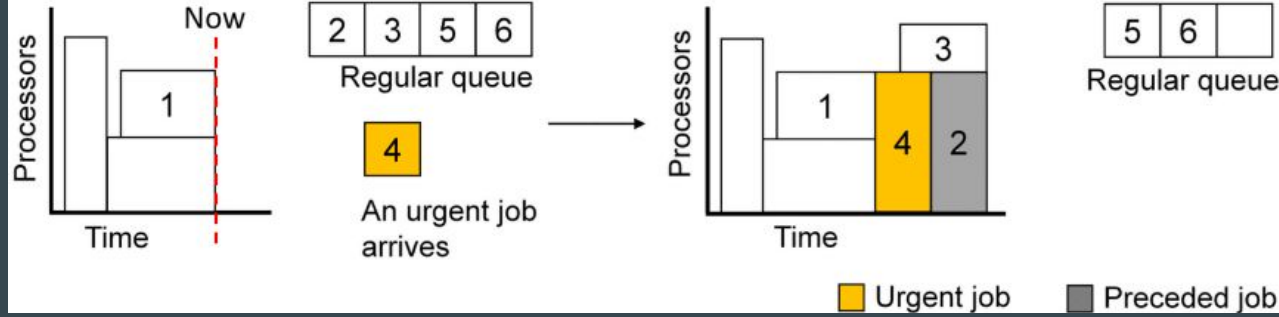


Figure 6. Urgent job execution by preceding regular jobs.

The urgent job is executed by backfilling.

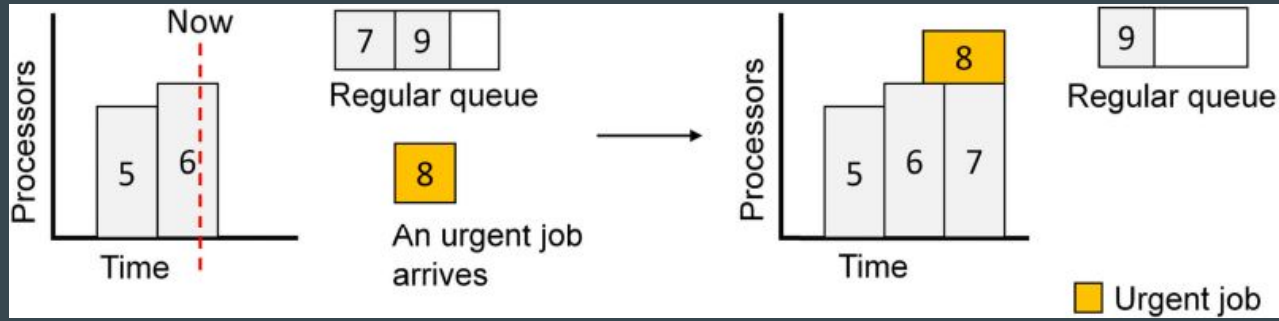


Figure 7. Urgent job execution by backfilling.

The urgent job is executed by preempting regular jobs.

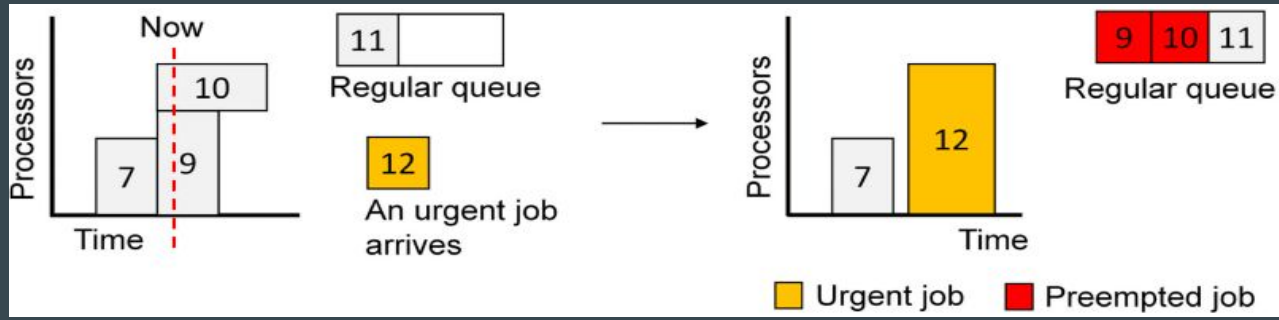


Figure 8. Urgent job execution by preemption.

Process swapping performance

- Process swapping: moving process memory from device to host memory and vice versa
- Performance depends on the process size
- Measured the swapping time by swapping out each application from the VE to the VH at random times during its execution
- Evaluated the effects of process swapping on different process sizes by executing the applications with different input sizes

Input Size (Class)	Application	Avg. Swap Size (Megabytes)	Avg. S_p (Seconds)
A	BT	118.531	0.022
	CG	123.727	0.023
	EP	76.020	0.014
	FT	396.707	0.073
	IS	133.559	0.024
	LU	110.238	0.073
	MG	506.758	0.093
	SP	120.523	0.022
B	BT	248.621	0.040
	CG	512.705	0.082
	EP	76.020	0.012
	FT	1630.167	0.259
	IS	331.559	0.053
	LU	215.742	0.034
	MG	608.109	0.097
	SP	256.770	0.041
C	BT	573.680	0.106
	CG	741.656	0.138
	EP	76.023	0.014
	FT	2602.986	0.676
	IS	842.672	0.156
	LU	633.914	0.118
	MG	1741.941	0.323
	SP	797.586	0.148
D	BT	1223.390	0.201
	CG	1071.495	0.176
	EP	76.023	0.013
	FT	5127.443	0.844
	IS	2116.223	0.348
	LU	1116.267	0.184
	MG	2469.829	0.406
	SP	1441.358	0.237

Table 3. Swapping time results of the NPB applications.

Process swapping performance

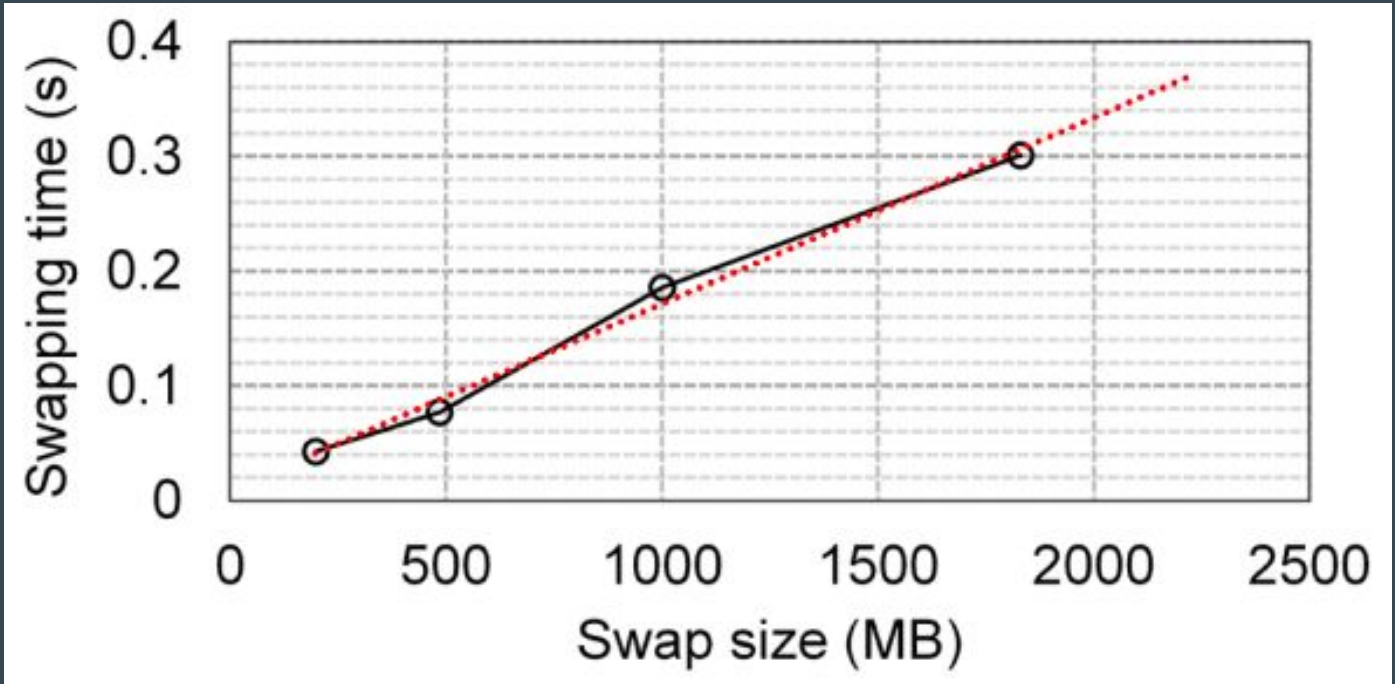
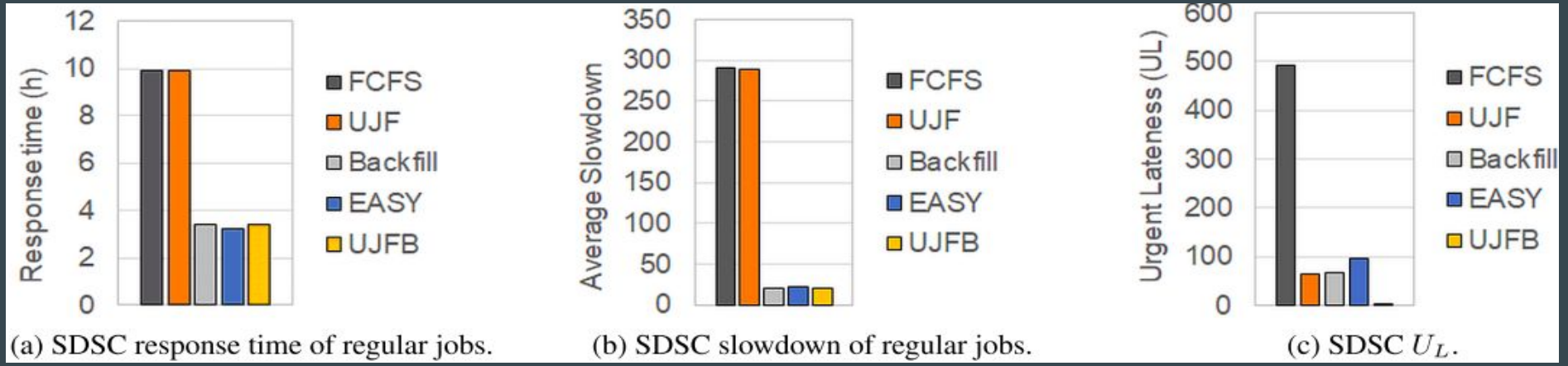
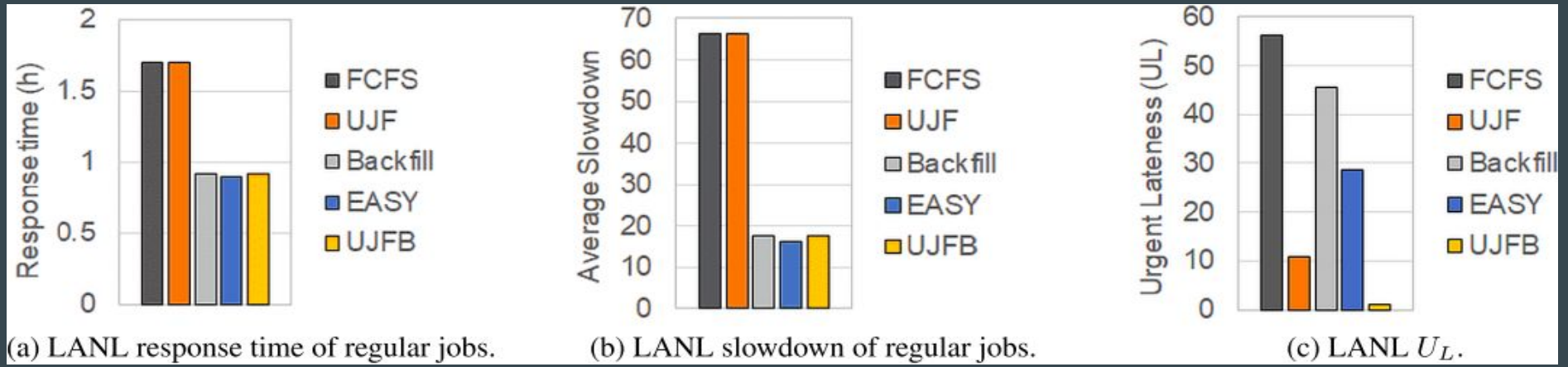


Figure 9. Performance of PPS.



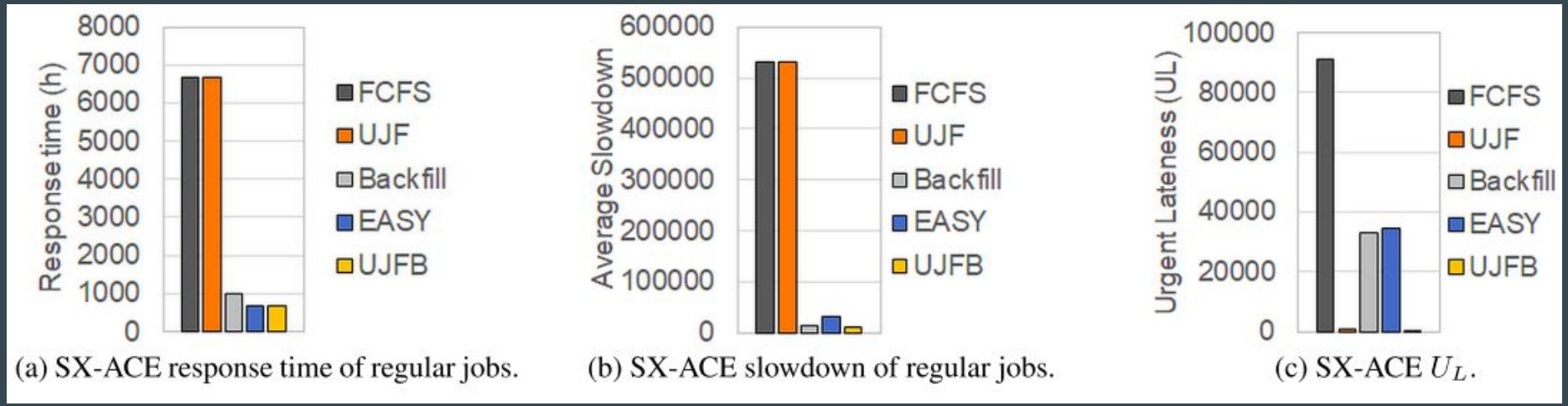
(a) SDSC response time of regular jobs. (b) SDSC slowdown of regular jobs. (c) SDSC U_L.

Figure 10. The results of the SDSC workload.



(a) LANL response time of regular jobs. (b) LANL slowdown of regular jobs. (c) LANL U_L.

Figure 11. The results of the LNLL workload.



(a) SX-ACE response time of regular jobs. (b) SX-ACE slowdown of regular jobs. (c) SX-ACE U_L.

Figure 12. The results of the SX-ACE workload.

Algorithm	U_L	Num. preemption
FCFS	90996.73	0
UJF	641.61	0
Backfill	32853.86	0
EASY	34688.48	0
UJFB	1.05	28

Table 2. SX-ACE results for urgent jobs.

Evaluation with Longer Swapping Times

- Found that longer swapping times do not affect regular jobs' performance
- Longer swapping times can significantly delay urgent jobs (another reason for PPS)
  - This is due to increased preemption delays which consequently increase lateness of urgent jobs

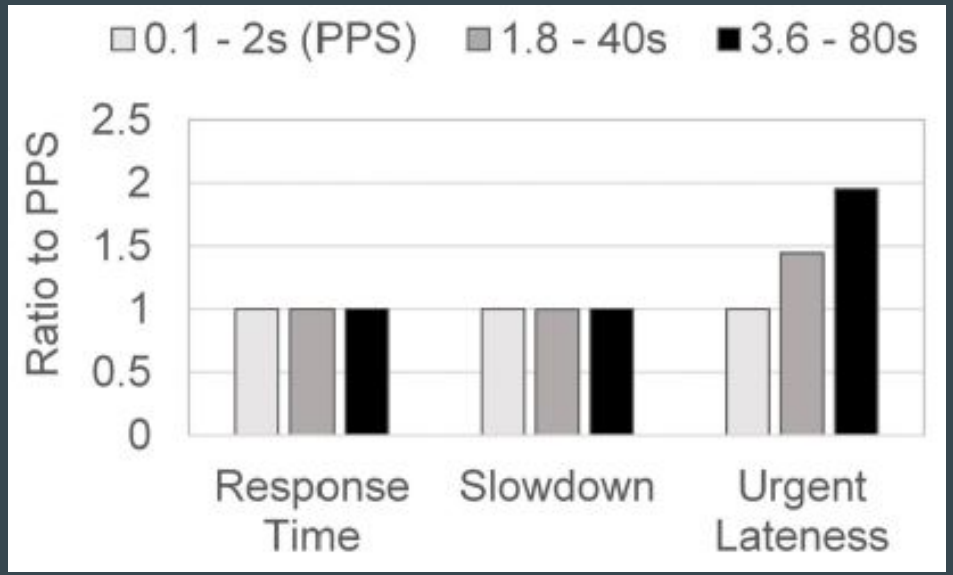


Figure 13. The LANL results with longer swapping times.

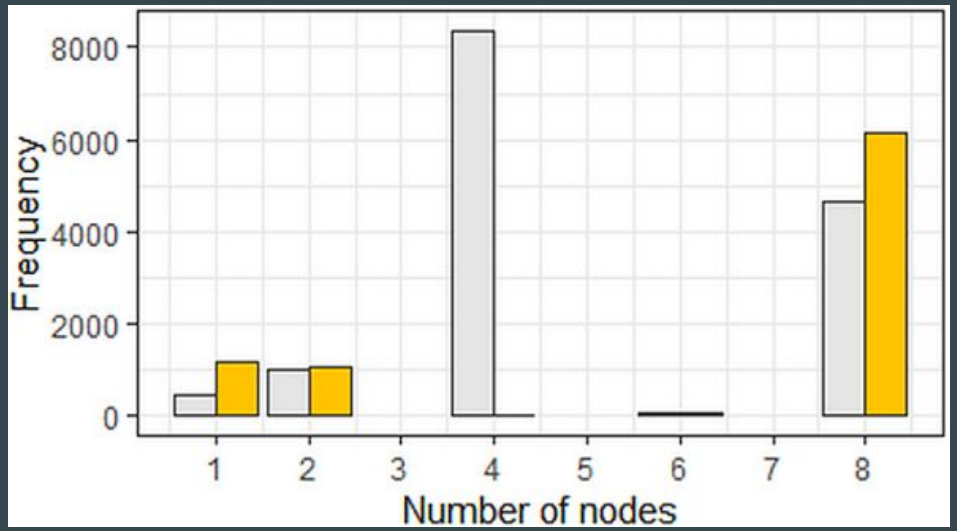


Figure 14. Job arrival times.

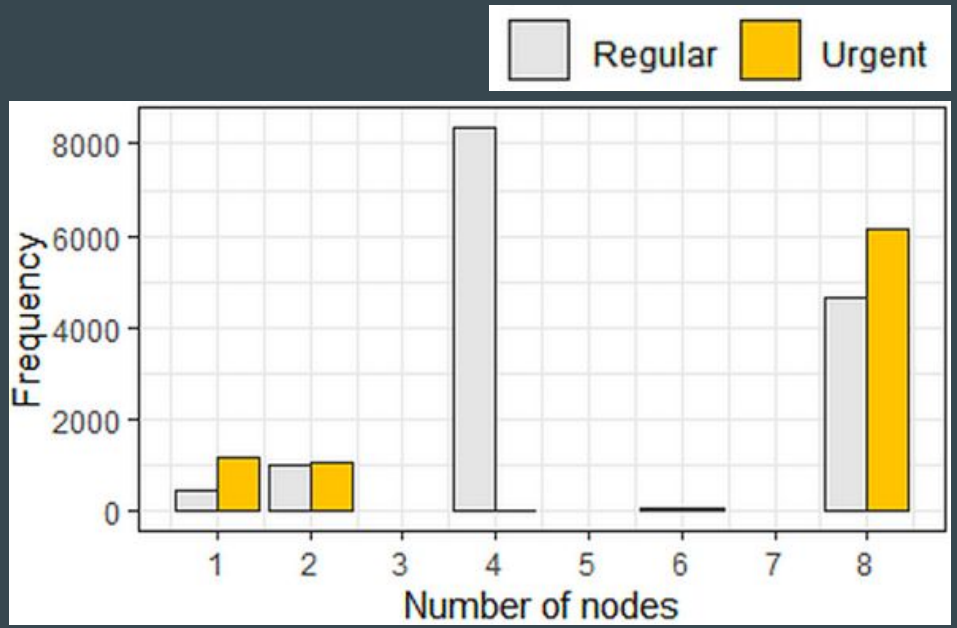
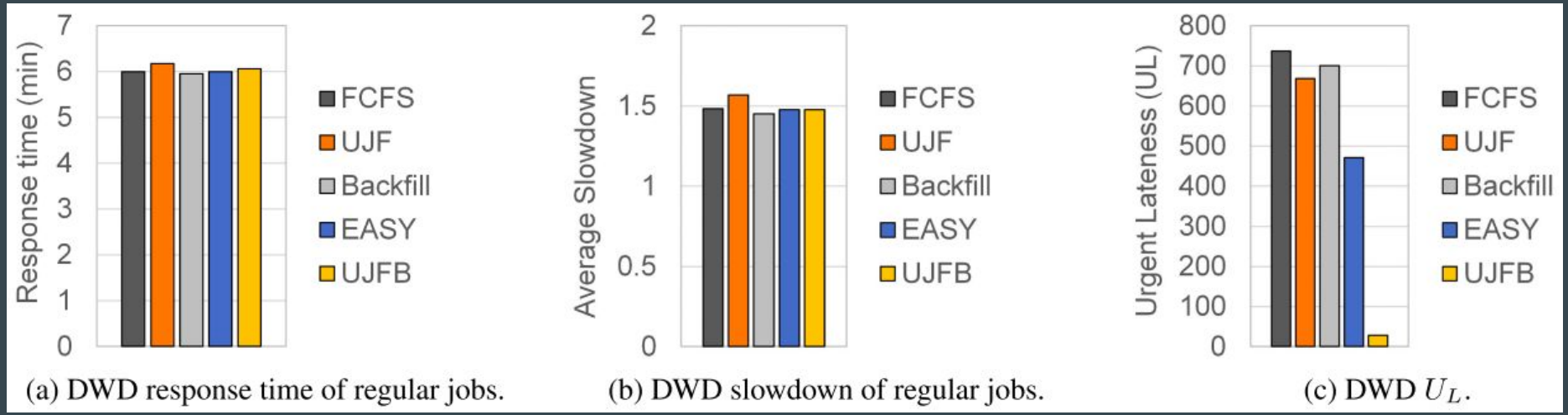


Figure 15. Number of nodes required by jobs.



(a) DWD response time of regular jobs. (b) DWD slowdown of regular jobs. (c) DWD U_L.

Figure 16. The results of DWD workload.

Conclusion

- Hard to test this out in real world situations (yay ethics)
- Potential for different protocols based on how urgent the job is (kill vs. pause)
- Use their method of measuring ability to handle UC in future research
- Look at how computer architecture can improve running time of PPS