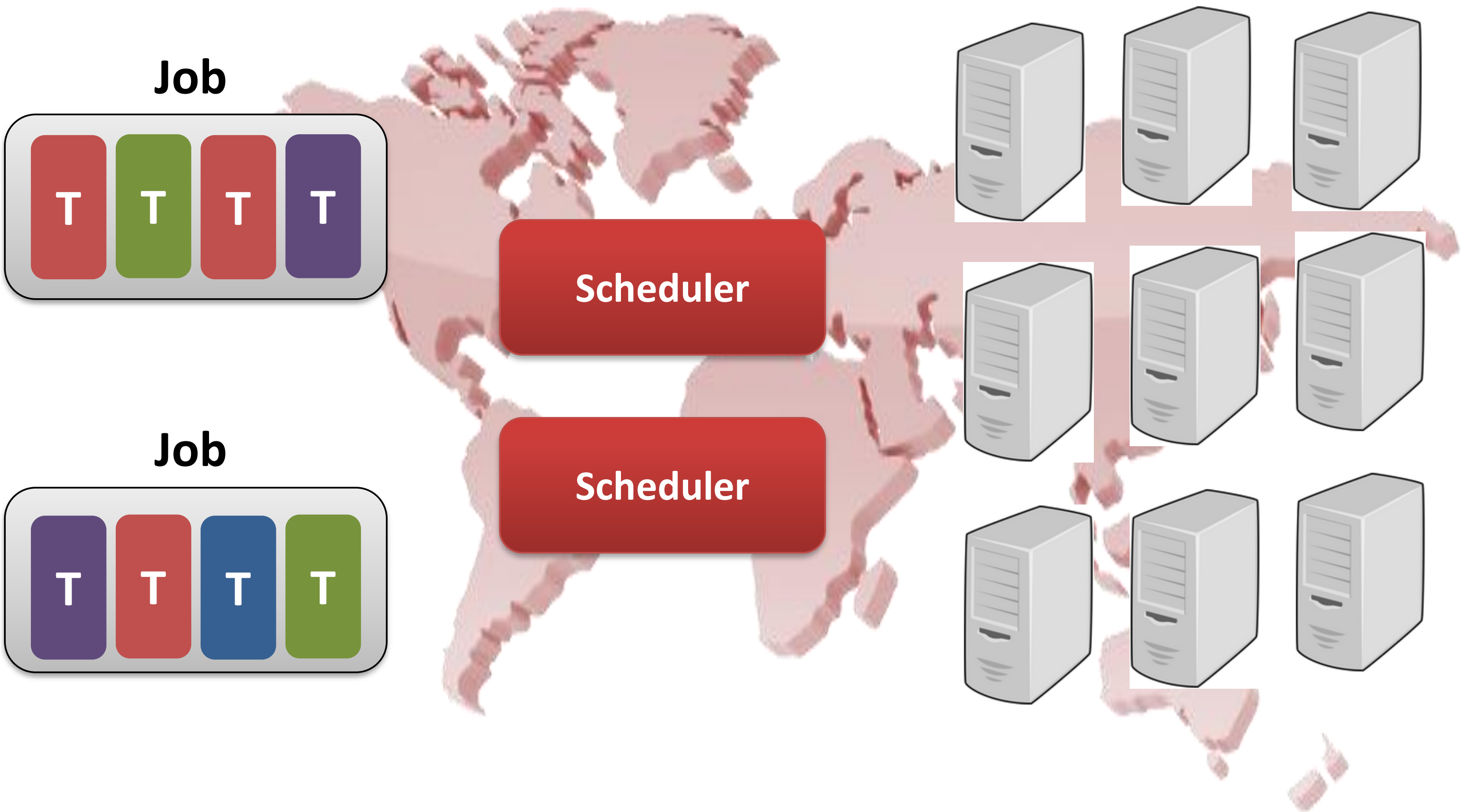
Leveraging Dependency in Scheduling and Preemption for High Throughput in Data-Parallel Clusters

Jinwei Liu*, Haiying Shen† and **Ankur Sarker**†

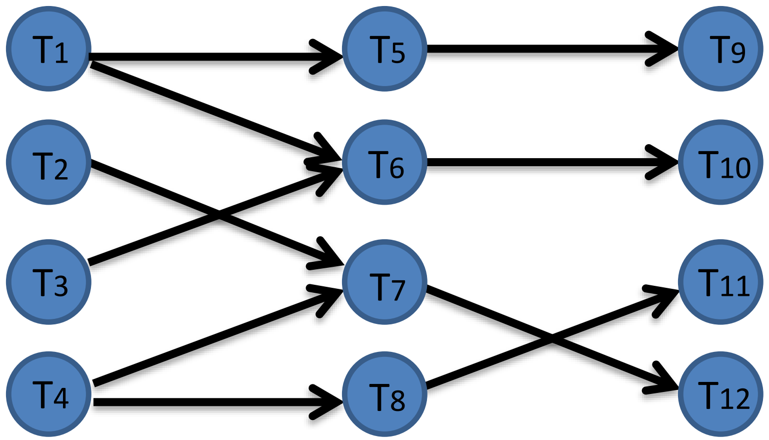
*Dept. of Electrical and Computer Engineering, Clemson University, Clemson, SC, USA
†Dept. of Computer Science, University of Virginia, Charlottesville, VA, USA

Introduction



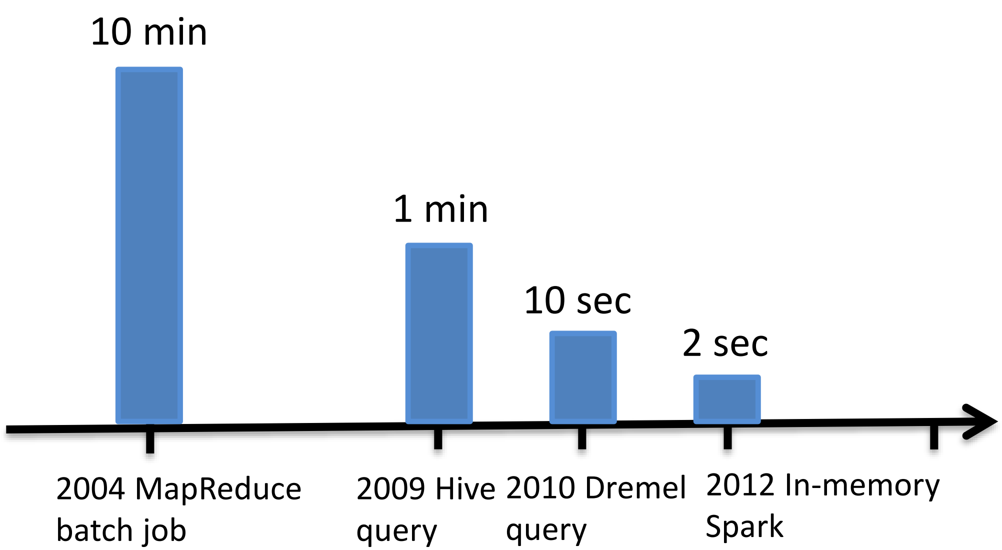
Motivation

- **Diverse task dependency**



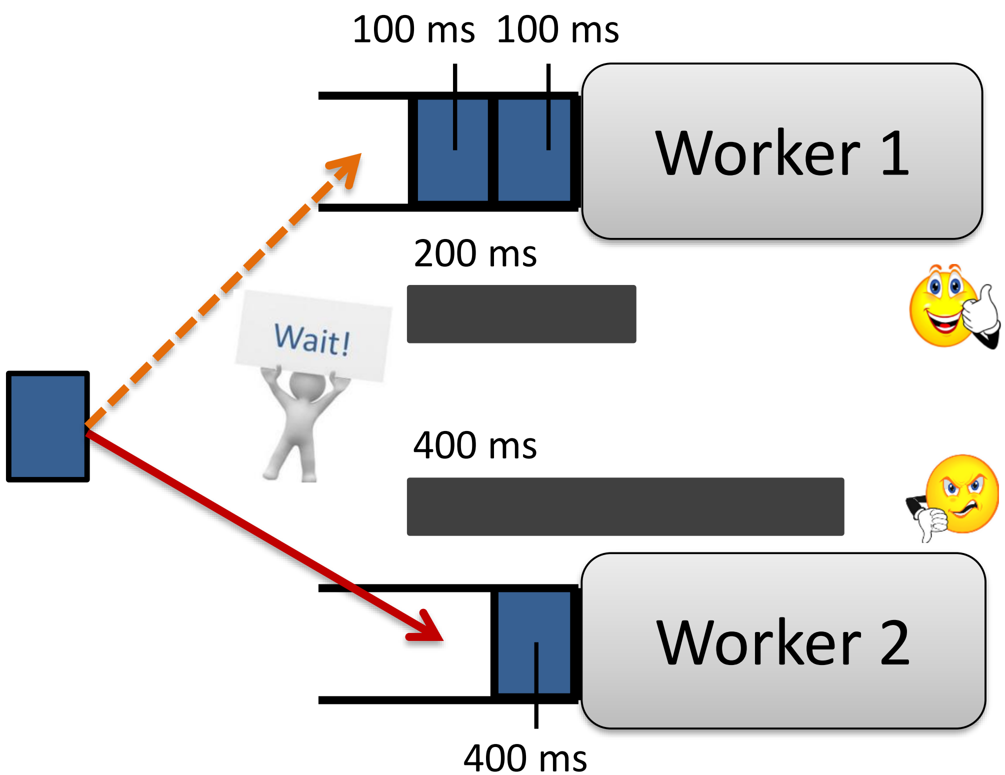
Motivation (cont.)

- **High requirements on completion time**



Motivation (cont.)

- **Queue length poor predictor of waiting time**

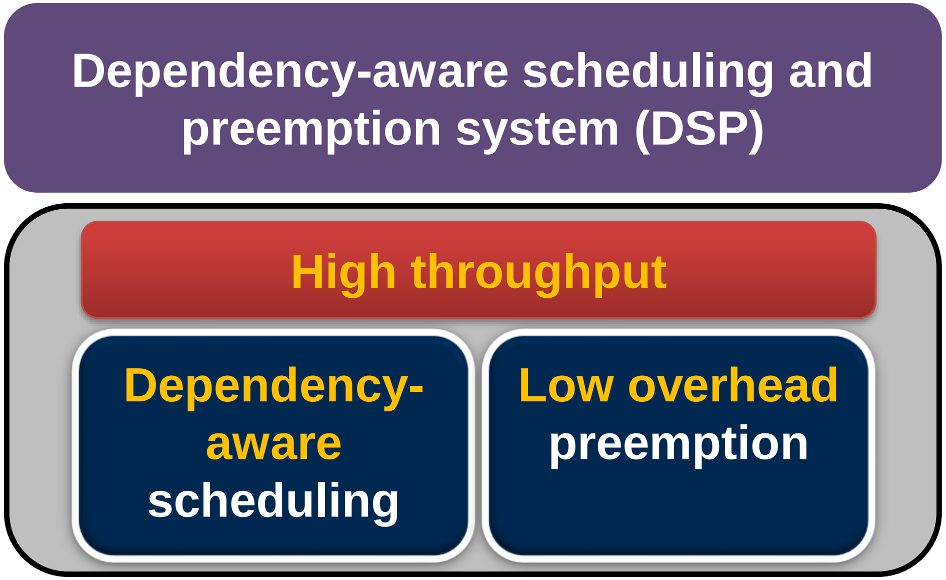


Outline

- Introduction
- Overview of Dependency-aware Scheduling and Preemption system (DSP)
- Design of DSP
- Performance Evaluation
- Conclusion

Proposed Solution

- **DSP:** Dependency-aware scheduling and preemption system
  - **Features of DSP**
    - Dependency awareness
    - High throughput
    - Low overhead
    - Satisfy jobs' demands on completion time



Framework of DSP

Design of DSP

- **Dependency-aware scheduling**
  - **Mathematical model for offline scheduling**

$$Min\{\mathcal{L}_{\mathcal{MS}}\}$$

$$s.t.\ \ \max\left(t_{ij}^s + t_{ij,k}\cdot x_{ij,k} + N_{ij,k}^p(t_{ij}^r+\sigma)x_{ij,k}\right) - \min t_{ij}^s \le \mathcal{L}_{\mathcal{MS}}$$

$$(t_{ij}^s + t_{ij,k})\cdot x_{ij,k} \le (t_{uv}^s + (1-y_{ij,uv,k})\cdot t_{uv,k})\cdot x_{uv,k}$$

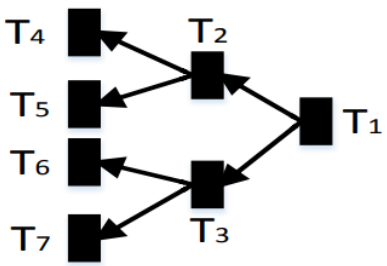
$$\max\left(t_{ij}^s + t_{ij,k}\cdot x_{ij,k} + N_{ij,k}^p(t_{ij}^r+\sigma)x_{ij,k}\right) \le t_i^a$$

$$\max\left(t_{ij}^s + t_{ij,l}\cdot x_{ij,l} + N_{ij,l}^p\cdot(t_{ij}^r+\sigma)\cdot x_{ij,l}\right) \le t_{iq}^s$$

Derive the target worker and starting time for each task

Design of DSP (cont.)

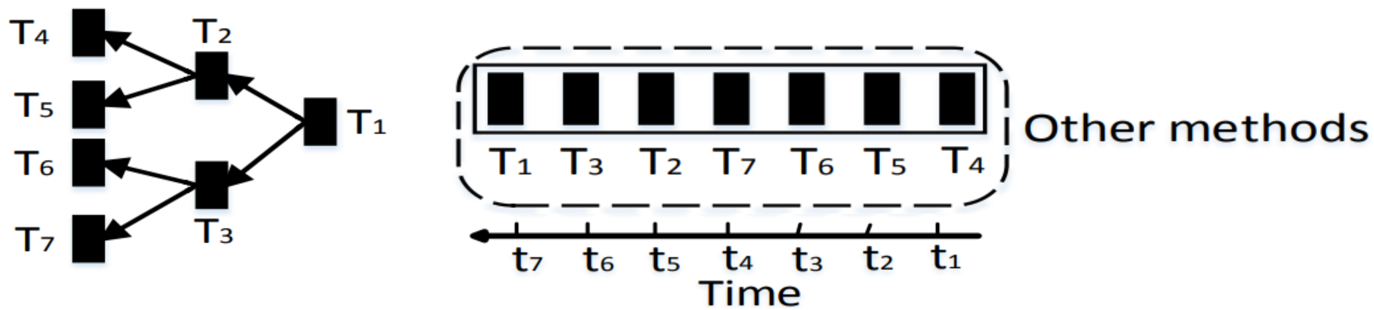
- **Dependency-aware task preemption**
  - **Dependency-aware task priority determination**



  - Task dependency: T_2 and T_3 depend on T_1, T_4 and T_5 depend on T_2, and T_6 and T_7 depend on T_3

Design of DSP (cont.)

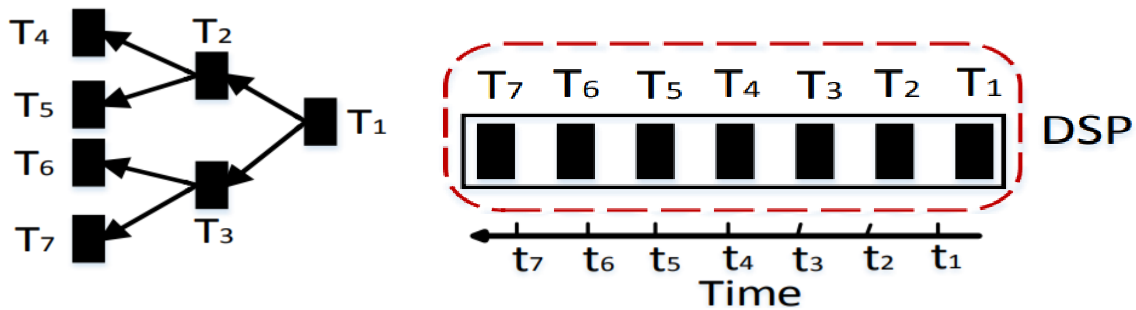
- **Dependency-aware task preemption**
  - **Dependency-aware task priority determination**



  - Task dependency: T_2 and T_3 depend on T_1, T_4 and T_5 depend on T_2, and T_6 and T_7 depend on T_3
  - Priorities assigned by other methods W/o considering dependency
    - $T_1 < T_3 < T_2 < T_7 < T_6 < T_5 < T_4$

Design of DSP (cont.)

- **Dependency-aware task preemption**
  - ➢ **Dependency-aware task priority determination**



– Task dependency: T_2 and T_3 depend on T_1, T_4 and T_5 depend on T_2, and T_6 and T_7 depend on T_3

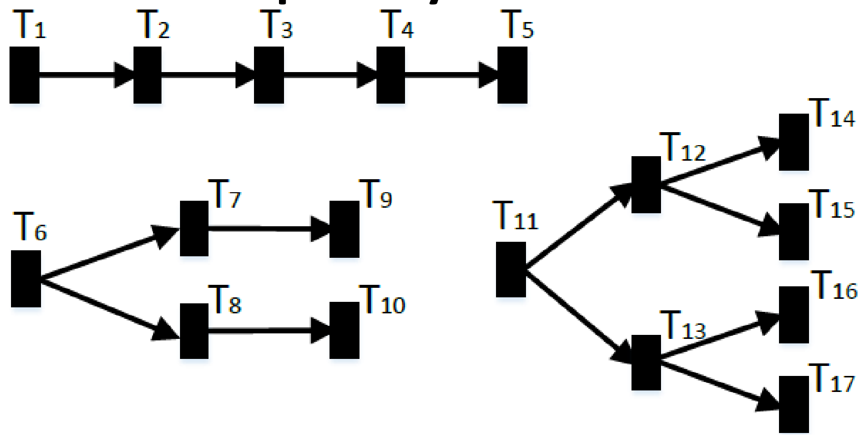
– Priorities assigned by DSP

- $T_7 < T_6 < T_5 < T_4 < T_3 < T_2 < T_1$ or $T_6 < T_7 < T_5 < T_4 < T_3 < T_2 < T_1$

Rationale: Choosing tasks with more dependent tasks to run enables more runnable tasks; more runnable task options enable to select a better task that can more increase the throughput

Design of DSP (cont.)

- **Dependency-aware task preemption**
  - ➢ **Dependency-aware task priority determination**



– Priority of task T_{ij} at time t

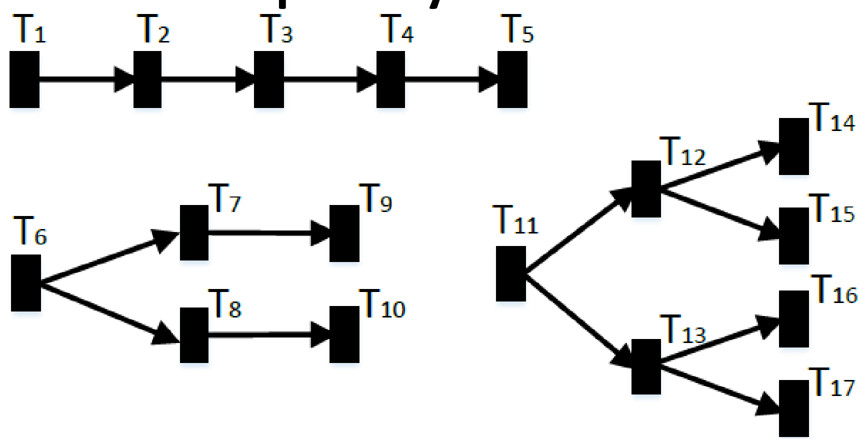
$$P_{ij}^{t} = \sum_{T_{ik} \in s_{ij}} (\gamma + 1) P_{ik}^{t}$$

Recursive computation of task priority (1)

s_{ij} is a set consisting of T_{ij}'s children, $\gamma \in (0,1)$ is a coefficient, t_{ij}^{a} is the allowable waiting time of task T_{ij}, ω_1, ω_2, ω_3 are the weights for task's remaining time, waiting time and allowable time

Design of DSP (cont.)

- **Dependency-aware task preemption**
  - **Dependency-aware task priority determination**



– Priority of task T_{ij} at time t

Recursive computation of task priority

$$P_{ij}^{t} = \sum\nolimits_{T_{ik} \in s_{ij}} (\gamma + 1) P_{ik}^{t} \qquad (1)$$

– Priority of task T_{ij} without dependent tasks at time t

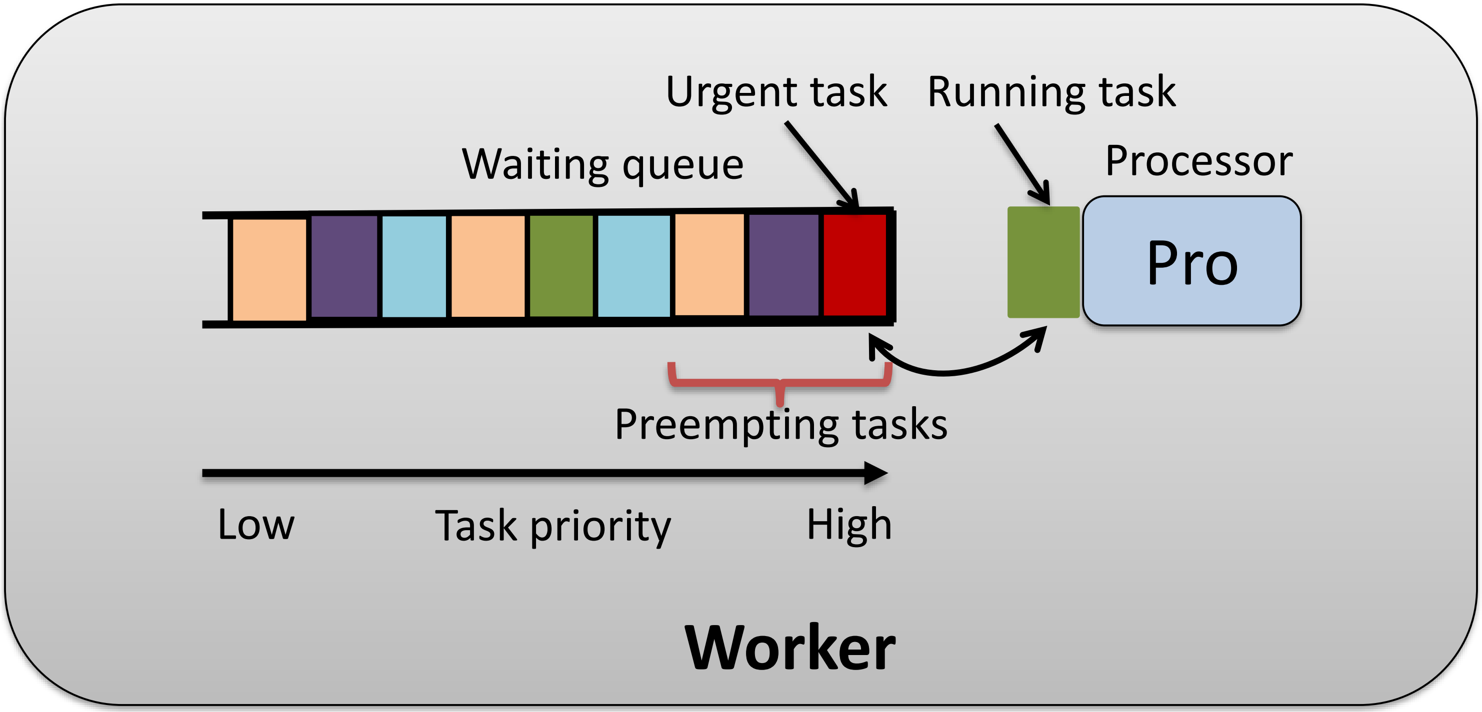
Leaf task

$$P_{ij}^{t} = \omega_1 \cdot \frac{1}{t_{ij}^{rem}} + \omega_2 \cdot t_{ij}^{w} + \omega_3 \cdot t_{ij}^{a} \qquad (2)$$

s_{ij} is a set consisting of T_{ij}'s children, $\gamma \in (0,1)$ is a coefficient, t_{ij}^{a} is the allowable waiting time of task T_{ij}, ω_1, ω_2, ω_3 are the weights for task's remaining time, waiting time and allowable time

Design of DSP (cont.)

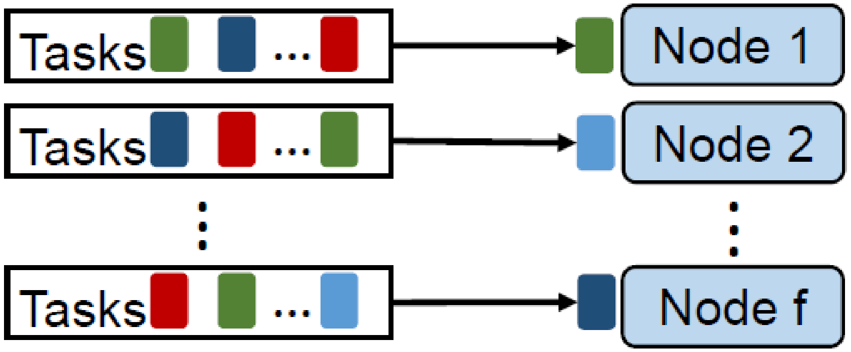
- **Priority based preemption**
  - **Selective preemption:** δ portion of tasks could be preempted



Advantage: Significantly reduce overhead caused by preemption

Design of DSP (cont.)

- **Priority based preemption**
  - ➢ **Preemption for multiple tasks running on multiple processors**



  - – Each node has a queue containing tasks that will run on the node
  - – Tasks with the same color belong to the same job
  - – Tasks are in the ascending order of their starting times

Design of DSP (cont.)

- **Priority based preemption**
  - **Pseudocode for the dependency-aware task preemption algorithm**

Algorithm 1: Pseudocode for DSP task preemption

Input: A is the set of waiting tasks and B is the set of running tasks

Compute the priorities of all waiting and running tasks
Sort B in ascending order based on their priority values
for *each* $i = 1, ..., |A|$ **do**
    if $t^a[i] \leq \epsilon \; OR \; t^w[i] \geq \tau$ **then**
        for *each* $j = 1, ..., |B|$ **do**
            if $A[i]$ *does not depend on* $B[j]$ **then**
                Suspend task $B[j]$ and run task $A[i]$
                break
            else
                $j \leftarrow j + 1$
                continue
for *each* $i = 1, ..., |\delta|A||$ **do**
    for *each* $j = 1, ..., |B|$ **do**
        if $A[i]$ *depends on* $B[j]$ **then**
            $j \leftarrow j + 1$
            continue
        Compute priority difference $\hat{P}_i$ and normalized priority $\tilde{P}_i$ of $A[i]$
        if $\hat{P}_i > 0 \; \&\& \; \tilde{P}_i > \rho\hat{P}/\bar{P}$ **then**
            Suspend task $B[j]$ and run task $A[i]$
            break

Step 1: Task preemption based on two conditions

Step 2: Reduce excessive preemptions based on the normalized priority

Outline

- Introduction
- Overview of Dependency-aware Scheduling and Preemption system (DSP)
- Design of DSP
- Performance Evaluation
- Conclusion

Performance Evaluation

- **Methods for comparison**
  - **Tetris [1]**: Maximize to task throughput and speed up job completion time by packing tasks to machines
  - **Aalo [2]**: Minimize the average coflow's completion time
  - **Amoeba [3]**: Checkpointing mechanism in task preemption
  - **Natjam [4]**: Priority based preemption for achieving low completion time for high priority jobs
  - **SRPT [5]**: Priority based preemption based on waiting time and remaining time for a task

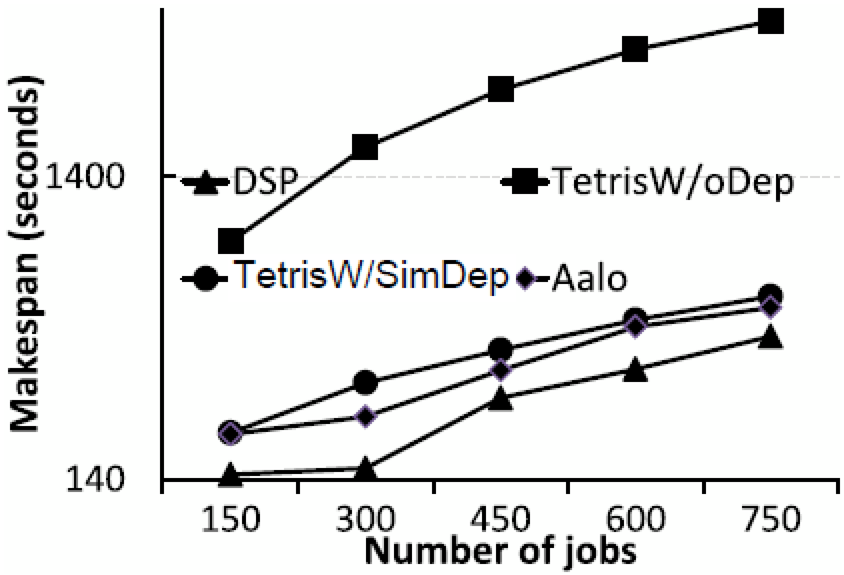
[1] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *Proc. of SIGCOMM*, 2014.
[2] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *SIGCOMM*, 2015.
[3] G. Ananthanarayanan, C. Douglas, R. Ramakrishnan, S. Rao, and I. Stoica. True elasticity in multi-tenant data-intensive compute clusters. In *Proc. of SoCC*, 2012.
[4] B. Cho, M. Rahman, T. Chajed, I. Gupta, C. Abad, N. Roberts, and P. Lin. Natjam: Design and evaluation of eviction policies for supporting priorities and deadlines in mapreduce clusters. In *Proc. of SoCC*, 2013.
[5] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Trans. on Computer Systems*, 21(2):207--233, 2003.

Experiment Setup

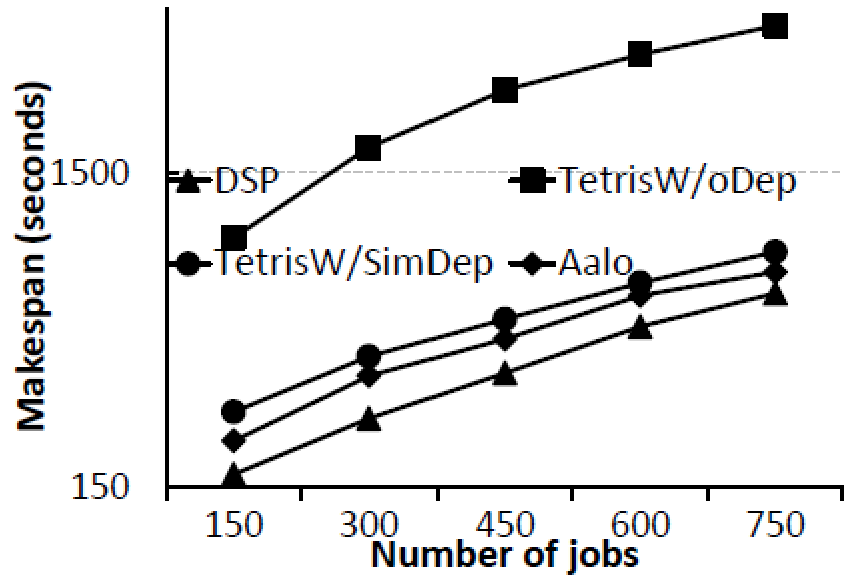
Parameter	Meaning	Setting
N	# of servers	30-50
h	# of jobs	150-2500
m	# of tasks of a job	100-2000
δ	Minimum required ratio	0.35
τ	Threshold of tasks' waiting time for execution	0.05
θ_1	Weight for CPU size	0.5
θ_2	Weight for Mem size	0.5
α	Weight for waiting time for SRPT	0.5
β	Weight for remaining time for SRPT	1
γ	Weight for waiting time	0.5
ω_1	Weight for task's remaining time	0.5
ω_2	Weight for task's waiting time	0.3
ω_3	Weight for task's allowable waiting time	0.2

Evaluation of DSP

- **Makespan**



(a) On the real cluster

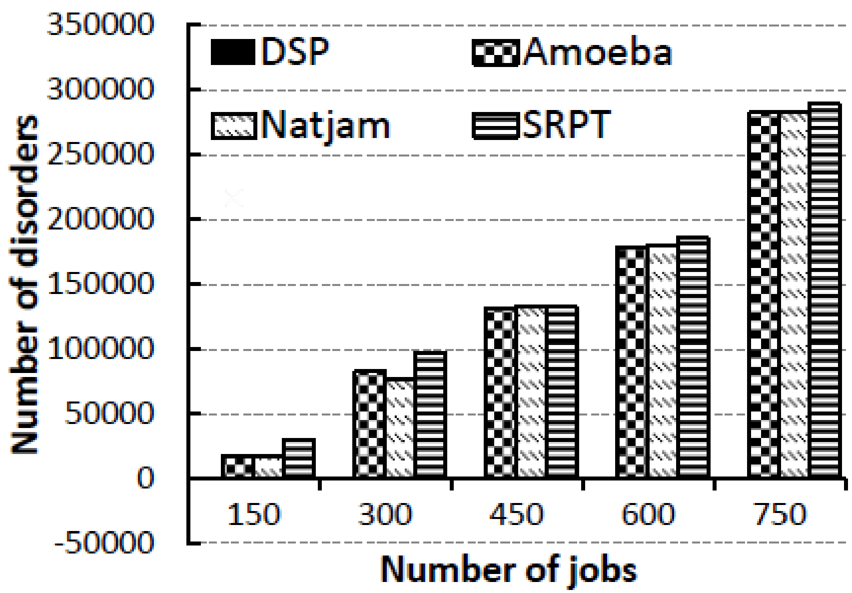


(b) On Amazon EC2

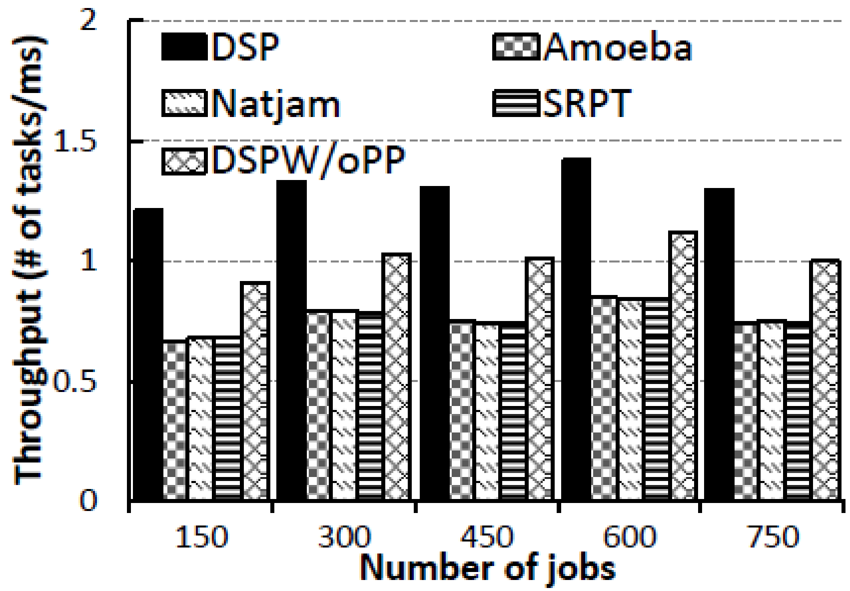
Result: Makespan increases as the number of nodes increases; makespans follow DSP < Aalo < TetrisW/SimDep < TetrisW/oDep

Evaluation of DSP (cont.)

- **Number of disorders and throughput**



(a) The number of disorders

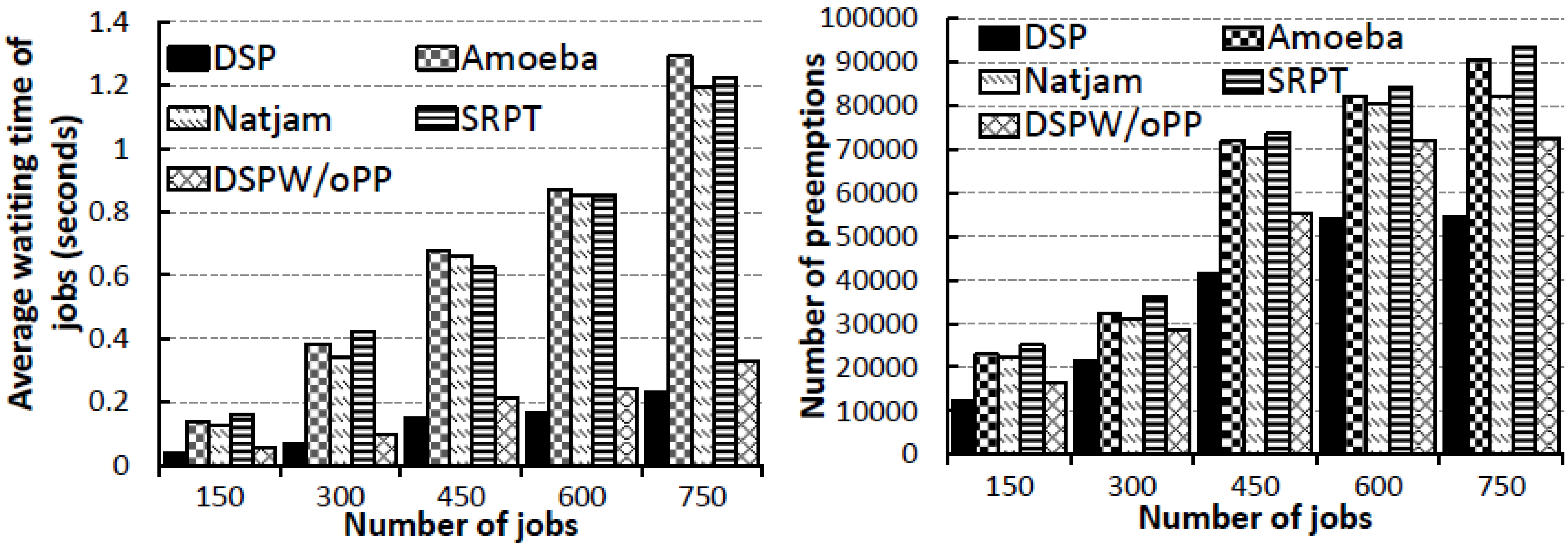


(b) Throughput

Result: # of disorders follows DSP < Natjam ≈ Amoeba < SRPT; throughput follows SRPT < Amoeba ≈ Natjam < DSPW/oPP < DSP

Evaluation of DSP (cont.)

- **Waiting time and overhead**

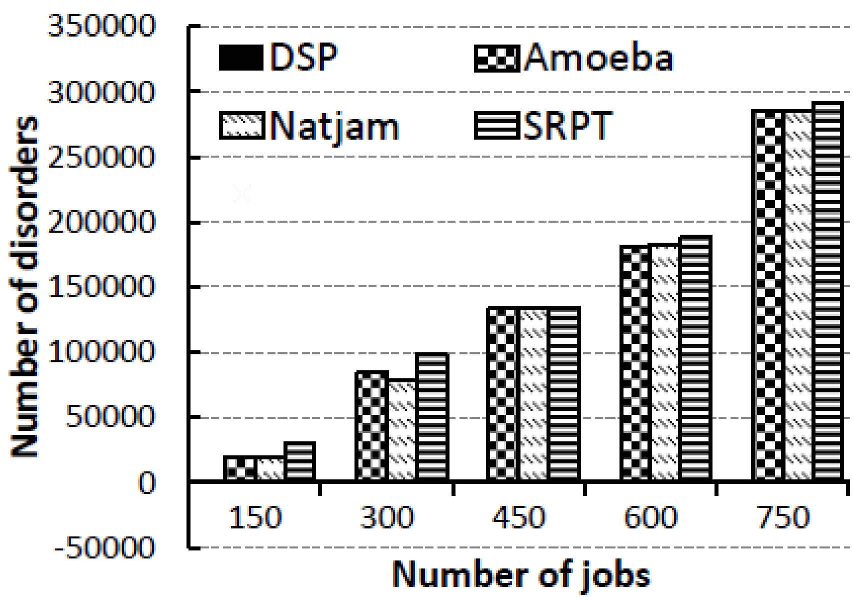


(a) Jobs' average waiting time **(b) Overhead**

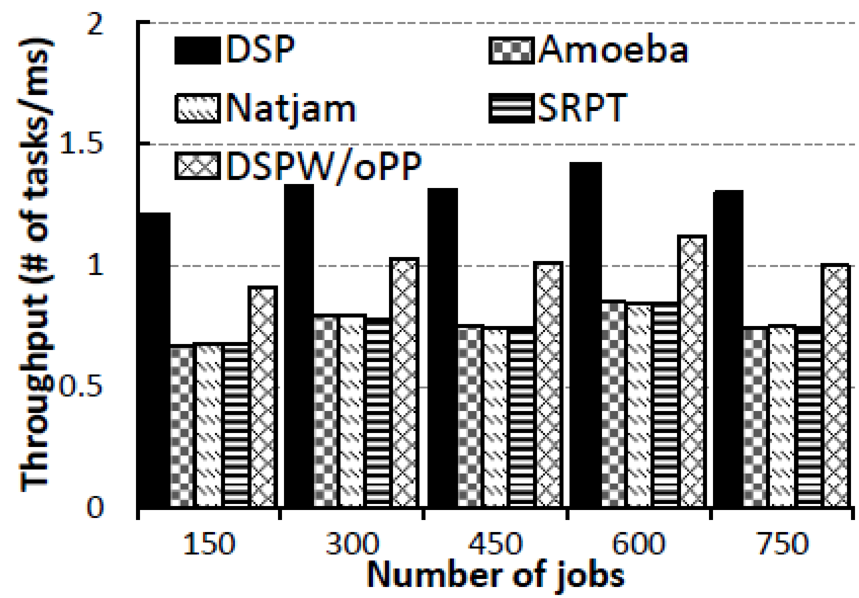
Result: Ave. waiting time of jobs approximately follows DSP < DSPW/oPP < Natjam ≈ SRPT < Amoeba; overhead follows DSP < DSPW/oPP < Natjam < Amoeba < SRPT

Evaluation of DSP (cont.)

- **Number of disorders and throughput on EC2**



(a) The number of disorders

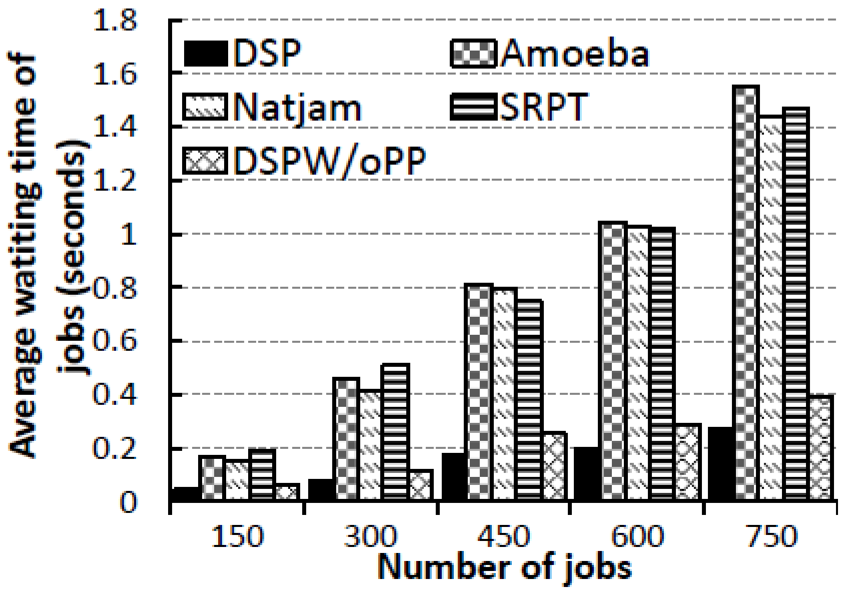


(b) Throughput

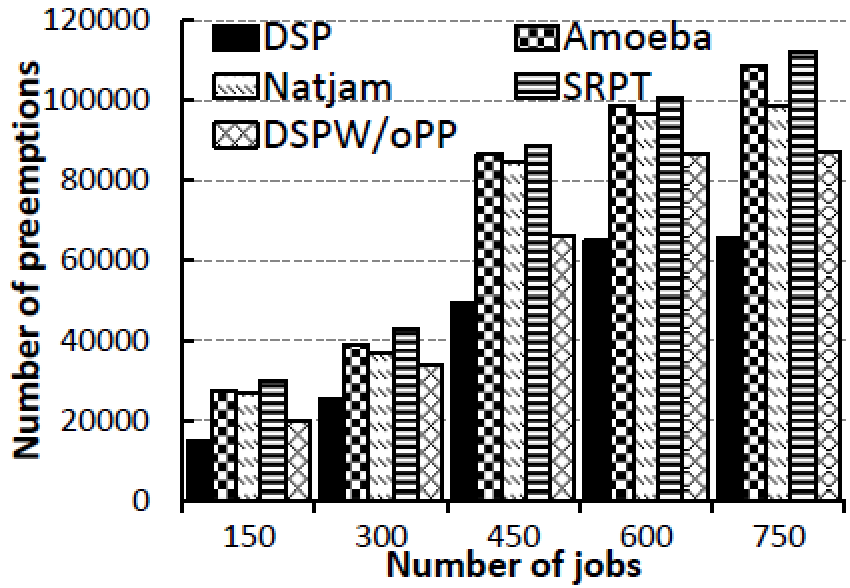
Result: # of disorders follows DSP < Natjam ≈ Amoeba < SRPT; throughput follows SRPT < Amoeba ≈ Natjam < DSPW/oPP < DSP

Evaluation of DSP (cont.)

- **Waiting time and overhead on EC2**



(a) Jobs' average waiting time

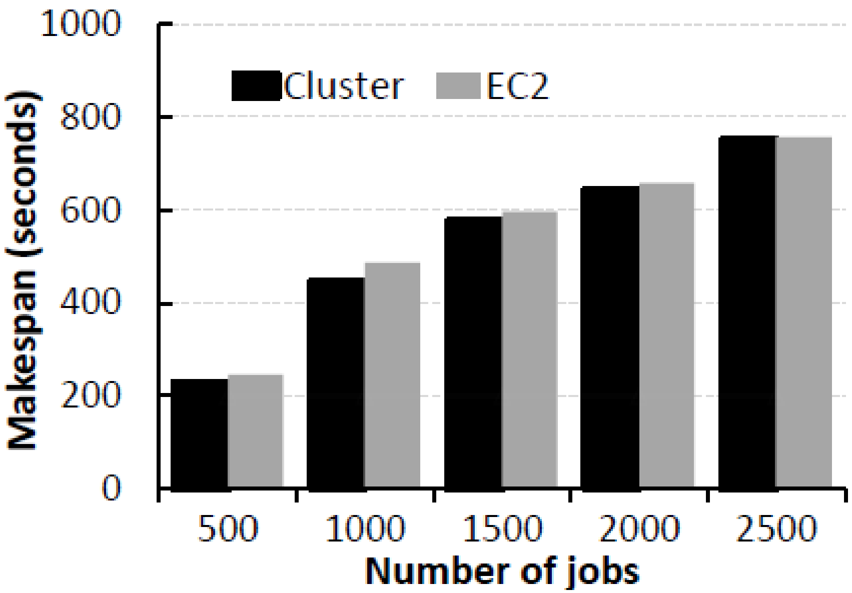


(b) Overhead

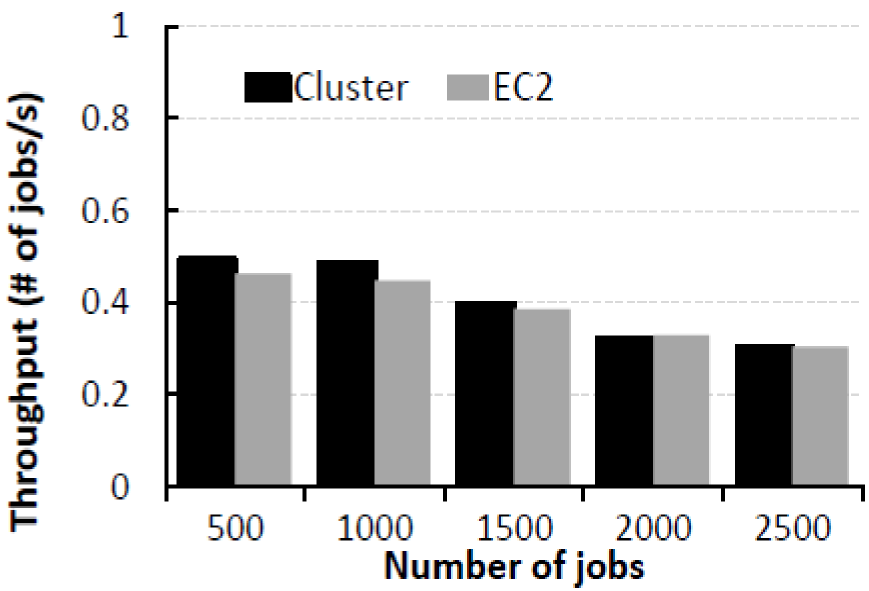
Result: Ave. waiting time of jobs approximately follows DSP < DSPW/oPP < Natjam ≈ SRPT < Amoeba; overhead follows DSP < DSPW/oPP < Natjam < Amoeba < SRPT

Evaluation of DSP (cont.)

- **Scalability**



(a) Makespan



(b) Throughput

Result: Makespan increases as the number of nodes increases; throughput decreases as the number of jobs increases

Outline

- Introduction
- Overview of Dependency-aware Scheduling and Preemption system (DSP)
- Design of DSP
- Performance Evaluation
- Conclusion

Conclusion

- **Our contributions**
  - Propose a dependency-aware scheduling and preemption system
  - Build a mathematical model to minimize makespan and derive target server for each task with the consideration of task dependency
  - Utilize task dependency to determine task priority
  - Propose a priority based preemption to reduce the overhead
- **Future work**
  - Study the sensitivity of the parameters
  - Consider data locality, fairness and cross-job dependency
  - Consider fault tolerance in designing a dependency-aware scheduling and preemption system

Thank you!
Questions & Comments?

Jinwei Liu (jinweil@clemson.edu)

Haiying Shen (hs6ms@virginia.edu)

Ankur Sarker (as4mz@virginia.edu)