SCHEDULING JOBS ACROSS GEO-DISTRIBUTED DATACENTERS

CHIEN-CHUN HUNG, LEANA GOLUBCHIK, MINLAN YU

Presented by Rohan Seth

WHY DO WE NEED DISTRIBUTED JOB SCHEDULING?

- With growing data volumes, it is becoming increasingly inefficient to aggregate all data required for computation at a single datacenter

- Instead, a recent trend is to distribute computation to take advantage of data locality, thus reducing the resource (e.g., bandwidth) costs while improving performance.

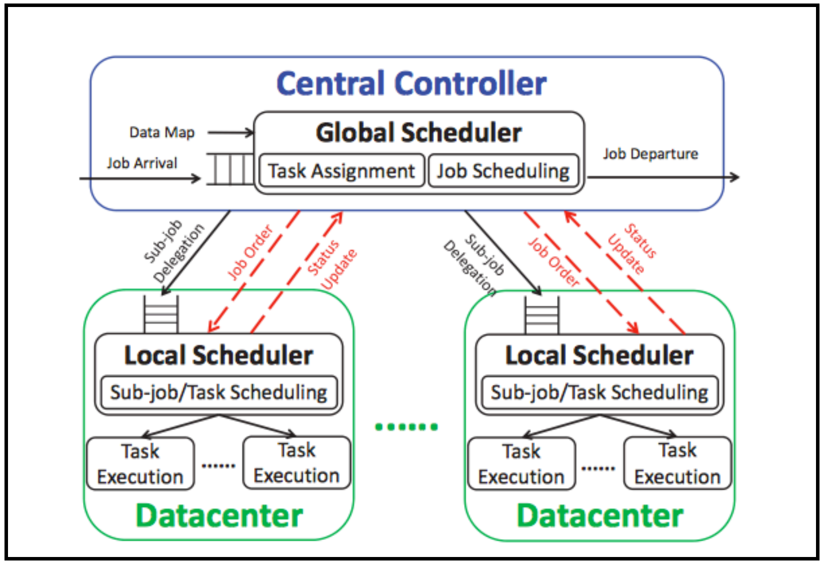
- However, this poses new challenges in job scheduling as it requires coordination among the datacenters

WHAT DOES THE PAPER ACHIEVE?

- The authors illustrated why natural SRPT based job scheduling algorithms provide room for improvement
- Proposed the technique of "Reordering" that can be easily added to any scheduling algorithm to improve it's performance
- Developed *Workload-Aware Greedy Scheduling (SWAG)*, which greedily serves the job that finishes the fastest by taking existing workload at the local queues into consideration
- *SWAG* and *Reordering* achieve 50% and 27% improvements, respectively in average job completion as compared to SRPT based algorithms

SYSTEM MODEL USED FOR DISTRIBUTED JOB EXECUTION

- Our system consists of a central controller and a set of datacenters D spanning geographical regions, while the system serves the jobs running with input data stored across the geo-distributed datacenters.
- The global scheduler residing in the central controller, makes job-level scheduling decisions for all jobs in the system , and assigns a job's tasks to the datacenters that host the input data.
- The local scheduler at each datacenter has a queue q_d that stores the tasks assigned by the global scheduler, and launches the tasks at the next available computing slot based on the job order determined by the global scheduler (or the local scheduler itself)

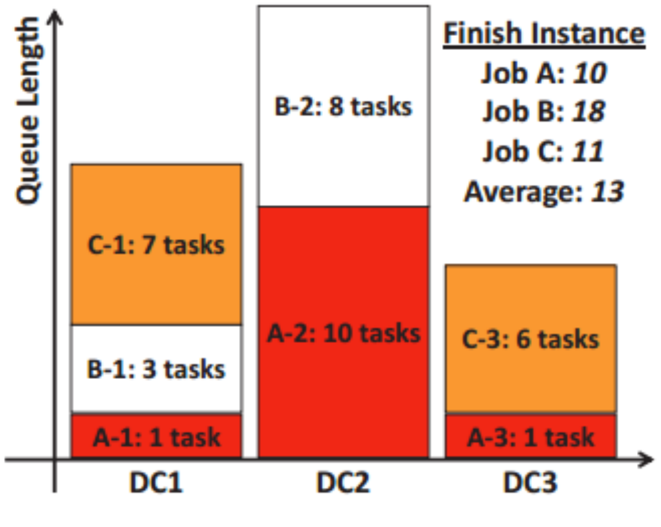


GLOBAL SRPT BASED SCHEDULING

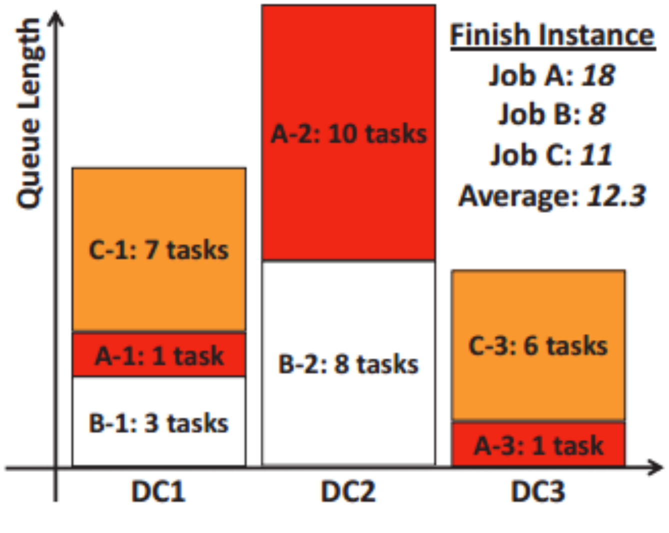
- Compute the jobs priority based on the jobs' total remaining size across all the datacenters
- Global-SRPT runs at the central controller, as it requires the global state of the current jobs' remaining tasks across all the datacenters.
- Central controller passes the job order computed by Global-SRPT to all the datacenters
- Each datacenter scheduler updates its sub-jobs order in the queue based on the new job order

GLOBAL SRPT

Job ID	Arrival Sequence	Remaining Tasks in DC1	Remaining Tasks in DC2	Remaining Tasks in DC3	Total Remaining tasks
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13



(a) FCFS



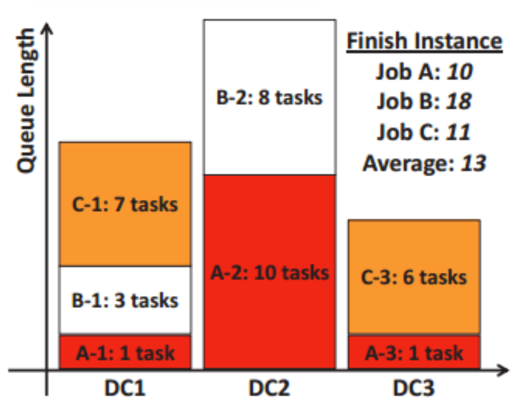
(b) Global-SRPT

INDEPENDENT SRPT BASED SCHEDULING

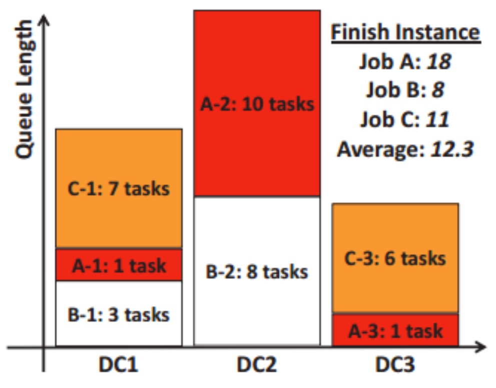
- Enable each datacenter scheduler to perform SRPT on its own
- The datacenter prioritizes its sub- jobs based on the their sizes and updates the queue order independently from the information of other datacenters

INDEPENDENT SRPT

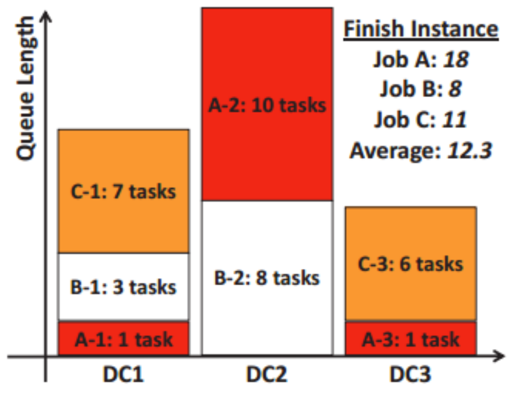
Job ID	Arrival Sequence	Remaining Tasks in DC1	Remaining Tasks in DC2	Remaining Tasks in DC3	Total Remaining tasks
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13



(a) FCFS



(b) Global-SRPT



(c) Independent-SRPT

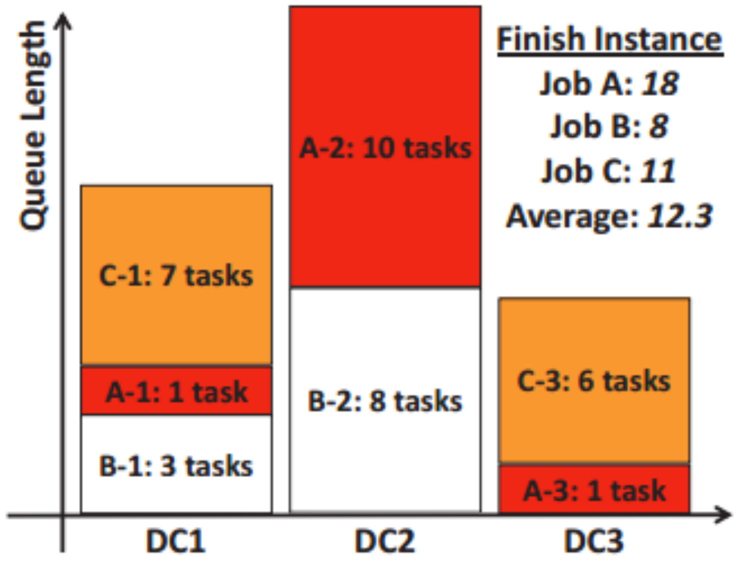
SHORTCOMINGS OF SRPT BASED EXTENSIONS

- Both Global-SRPT and Independent-SRPT improve the average job completion time by favoring small jobs.

- However, since each job may have multiple sub-jobs across all the datacenters, the imbalance of the sizes among the sub-jobs causes the problems for SRPT-based scheduling**.**

GLOBAL - SRPT

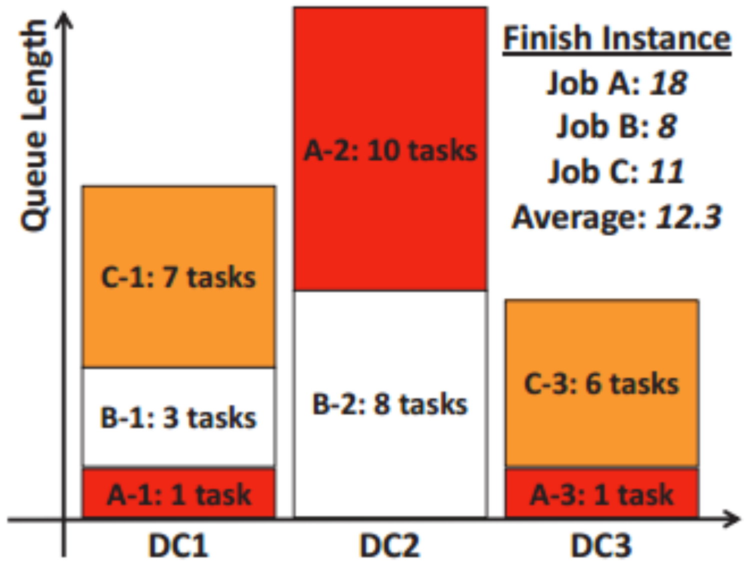
We see that job A's sub-jobs in datacenter 1 and 3 finish even before its sub-job at datacenter 2 starts. Since the job's completion time is determined by the last completed sub-job across all datacenters, we can actually defer $v_{A,1}$ and $v_{A,3}$ bit without hurting job A's finish instant, while it can yield the compute resources to the tasks of other sub-jobs.



(b) Global-SRPT

INDEPENDENT SRPT

The same observation is also valid for Independent-SRPT in the example, in which $V_{A,1}$ can yield to $V_{B,1}$ and $V_{C,1}$ in datacenter 1, and $V_{A,3}$ can yield to $V_{C,3}$ in datacenter 3, without delaying job A's finish time



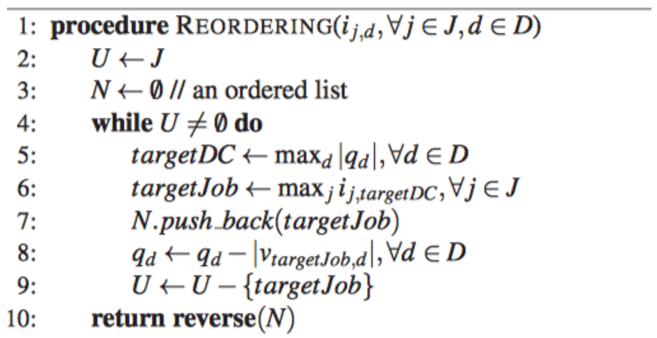
(c) Independent-SRPT

REORDERING

- SRPT-based heuristics do not result in better performance is that they fail to consider the competition for resources faced by each of its component sub-jobs

- Reordering, is used as an auxiliary mechanism to reduce the “imbalance” of a job’s sub-jobs.

- The basic idea behind Reordering is to continue moving sub-jobs later in a local queue, as long as delaying them does not increase the overall completion time of the job to which they belong.



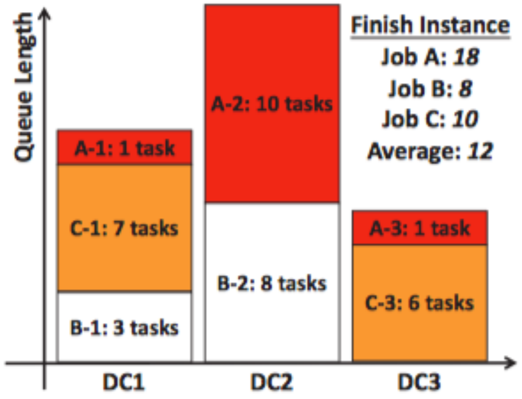
```
procedure REORDERING($i_{j,d}, \forall j \in J, d \in D$)
    $U \leftarrow J$
    $N \leftarrow \emptyset$ // an ordered list
    while $U \neq \emptyset$ do
        $targetDC \leftarrow \max_d |q_d|, \forall d \in D$
        $targetJob \leftarrow \max_j i_{j,targetDC}, \forall j \in J$
        $N.push\_back(targetJob)$
        $q_d \leftarrow q_d - |v_{targetJob,d}|, \forall d \in D$
        $U \leftarrow U - \{targetJob\}$
    return reverse($N$)
```

REORDERING

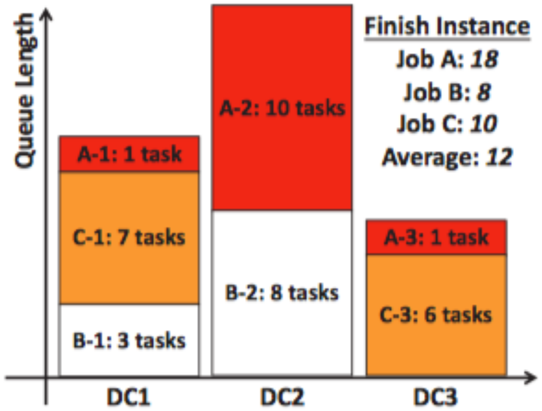
- Reordering improves both Global-SRPT and Independent-SRPT by delaying $V_{A,1}$ and $V_{A,3}$ until the end of their associated queues.

- The delay of $V_{A,1}$ and $V_{A,3}$ does not degrade Job A's finish instant as it is determined by $V_{A,2}$.

- This procedure continues by selecting Job C, and finally Job B, which results in N = A → C → B. Thus, Reordering returns B → C → A.



(e) Global-SRPT w/*Reordering*



(f) Independent-SRPT w/*Reordering*

SWAG DESIGN PRINCIPLES

- Jobs that can finish quickly should be scheduled before the other jobs.
- Should consider scheduling jobs more as a function of sub-job sizes rather than the size of the overall job.
- Should also consider the local queue sizes in assessing the finish times of sub-jobs.

SWAG ALGORITHM

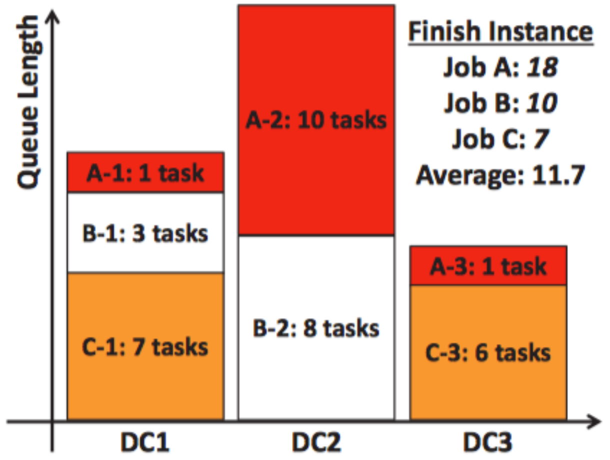
- The central controller runs SWAG whenever a new job arrives or departs. The new order of all jobs is computed from scratch based on the estimated job finish times

- SWAG greedily prioritizes jobs by computing their estimated finish times based on the current queue length as well as the job’s remaining size

SWAG ALGORITHM

- SWAG first selects Job C as it has the smallest makespan of 7, as compared to 10 for Job A and 8 for Job B.
- The queue length for datacenter 1 and datacenter 3 would be updated to 7 and 6, respectively, according to Job C's sub-job size
- Jobs A and B result in the same makespan of 10, with respect to the new queue lengths. Since Job B has a smaller remaining size than Job A, it is added after Job C, followed by Job A
- Final job order as computed by SWAG is C → B → A,

```
procedure SWAG($J, v_{j,d}, \forall j \in J, d \in D$)
    $N \leftarrow \emptyset$ // an ordered list
    $q_d \leftarrow 0, \forall d \in D$
    while $|N| \neq |J|$ do
        $m_j \leftarrow \max_d (q_d + |v_{j,d}|), \forall j \in J, d \in D$
        $targetJob \leftarrow \min_j m_j, \forall j \in J$
        $N.push\_back(targetJob)$
        $q_d \leftarrow q_d + |v_{targetJob,d}|, \forall d \in D$
    return $N$
```



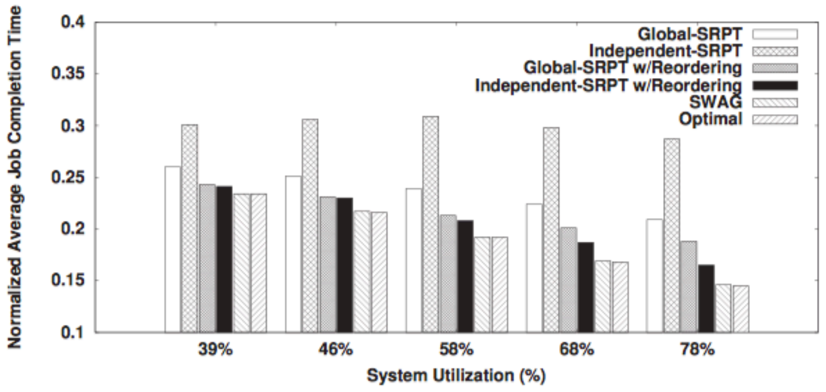
(d) *SWAG*

PERFORMANCE EVALUATION

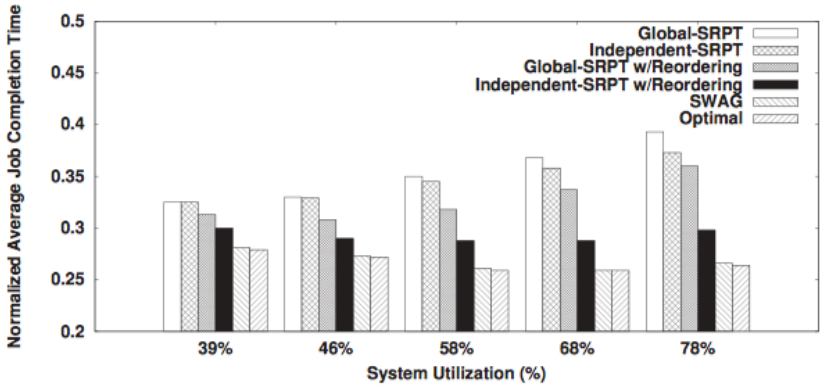
- The main performance metric the paper focused on was average job completion time, which is defined as the average elapsed duration from the job’s arrival time to the time instant at which the job has all its tasks completed and can depart from the system
- Compared the performance of: FCFS, Global-SRPT, Independent-SRPT, Global-SRPT followed by Reordering, Independent-SRPT followed by Reordering, and SWAG
- Workloads used in the experiments were Facebook’s production Hadoop cluster and Google cluster work- load trace, as well as the Exponential Distributions

PERFORMANCE EVALUATION

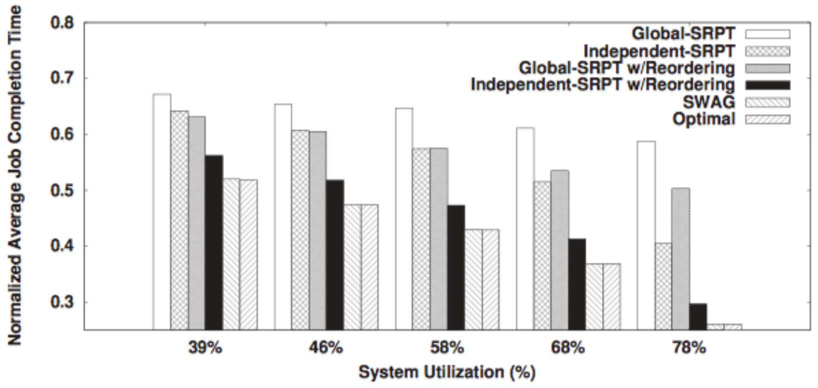
Trace Type	Avg. Job Size	Small Jobs (1 – 150 tasks)	Medium Jobs (151 – 500 tasks)	Large Jobs (501+ tasks)	Trace Characteristic
Facebook[5–8]	364.6 tasks	89%	8%	3%	high variance with a few extremely large jobs
Google[2, 28, 33]	86.9 tasks	96%	2%	2%	small variance with a few large jobs
Exponential	800 tasks	18%	29%	53%	moderate variance in job sizes



(a) Performance with Facebook Trace



(c) Performance with Google Trace



(e) Performance with Exponential Trace

PERFORMANCE EVALUATION - REORDERING

- Reordering results in reduction of average completion time for SRPT-based heuristics by 27% under highly utilized settings and is upto 17% under lower utilization.
- With Reordering, Independent-SRPT performs better than Global-SRPT in all the scenarios

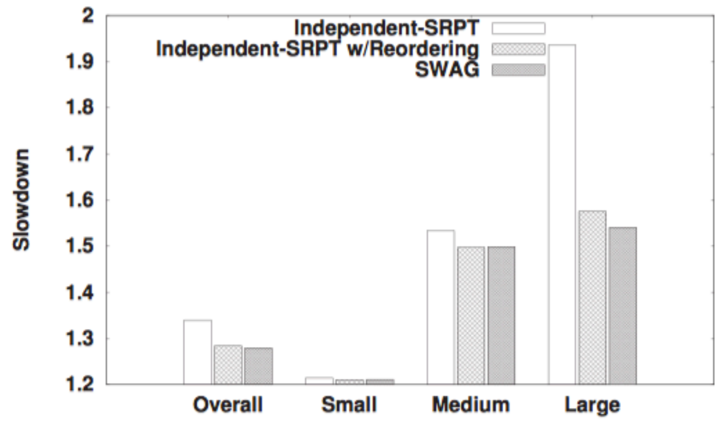
PERFORMANCE EVALUATION - SWAG

- Compared to SRPT-based heuristics, SWAG's performance improvements under higher utilization are up to 50%, 29% and 35% in the Facebook, Google and Exponential trace respectively
- The differences in performance improvements attribute to the fact that job traces with higher variance in job sizes tend to have more large jobs, which potentially results in more severe skews among the sub-jobs
- High-variance job trace like Facebook trace displays more opportunities that allow SWAG to achieve higher improvement by selecting jobs that can finish quickly according to its design principles

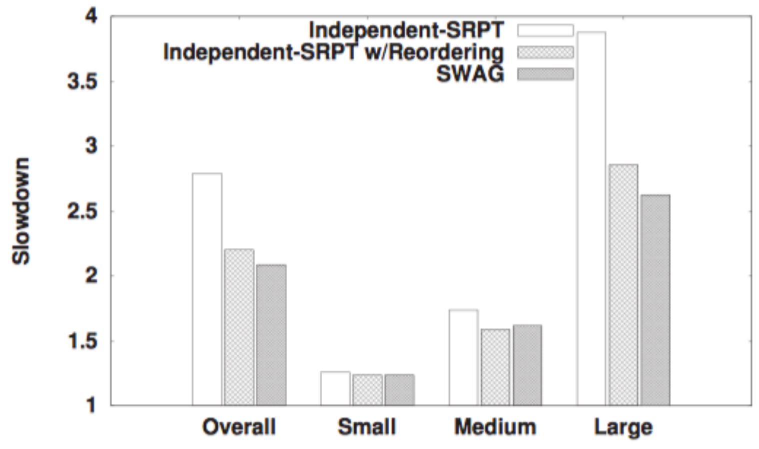
PERFORMANCE EVALUATION - FAIRNESS

- Classified the jobs based on their sizes: small jobs (1-150 tasks), medium jobs (151-500 tasks) and large jobs (501 or more tasks)
- All scheduling approaches have the same trends, i.e., that small jobs have the smallest slowdown while large jobs have the largest slowdown

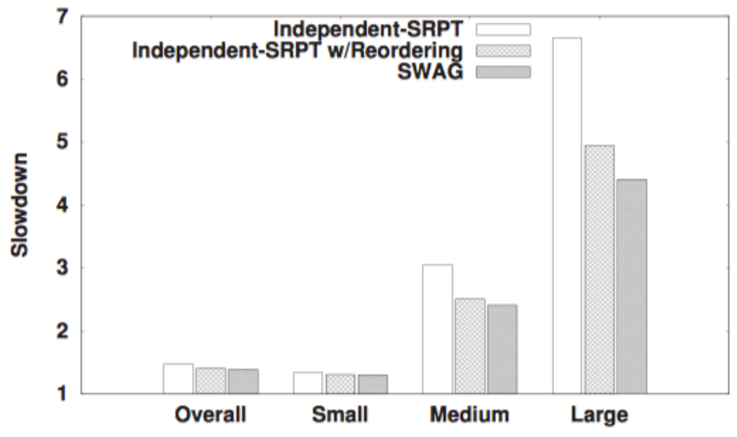
PERFORMANCE EVALUATION - FAIRNESS



(b) Fairness with Facebook Trace



(f) Fairness with Exponential Trace



(d) Fairness with Google Trace

OVERHEAD EVALUATION

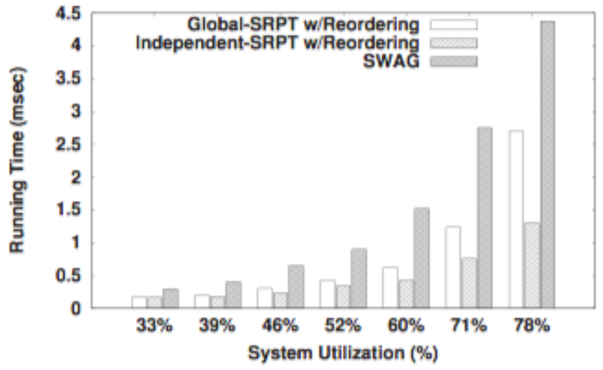
Evaluated the system overhead on 2 aspects:

1. Computational Overhead

2. Communication Overhead

COMPUTATIONAL OVERHEAD

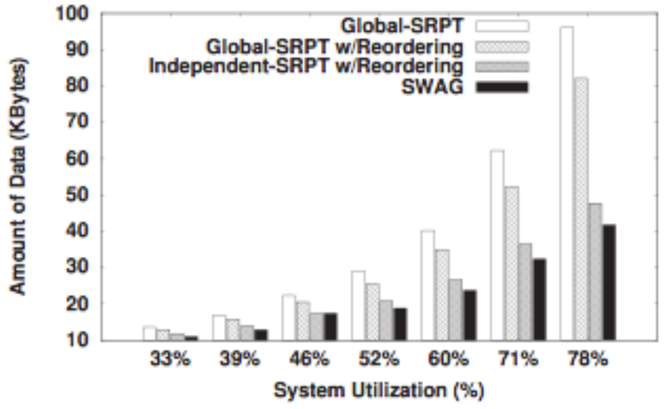
- Obtained this by monitoring the execution time due to running of the scheduling algorithms during each scheduling decision point
- Scheduled running time of SWAG (4.5ms) is relatively small compared to the average task duration time (2s)
- Actual difference in computational overhead between SWAG and SRPT-based heuristics with Reordering is not significant



(a) Scheduling Running Time

COMMUNICATION OVERHEAD

- Communication overhead essentially depends on the number of current jobs in the system
- SWAG succeeds in keeping the number of current jobs small, it achieves the smallest communication overhead.

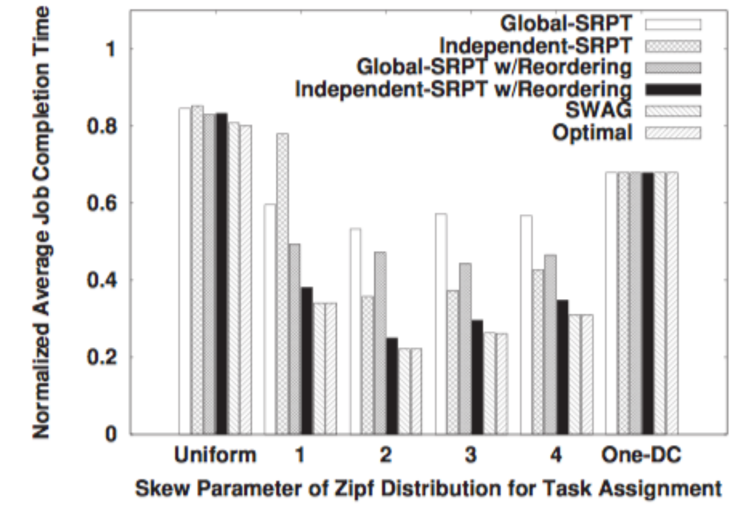


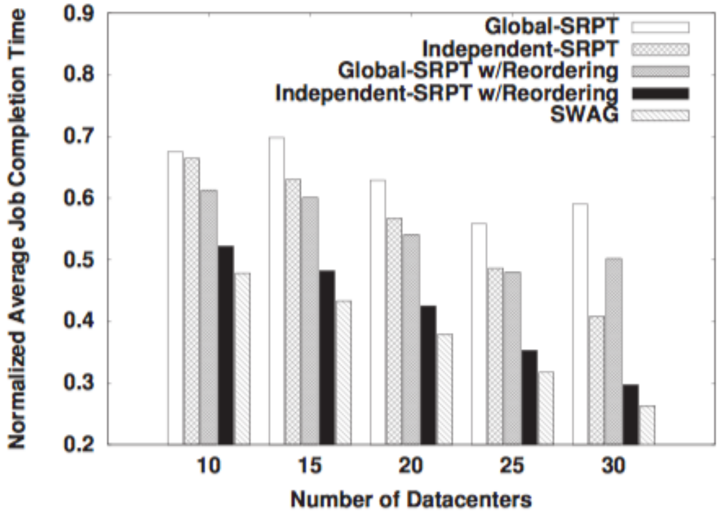
(b) Communication Overhead

PERFORMANCE SENSITIVITY ANALYSIS

Measured performance sensitivity on 2 scales:

- Impact of Task Assignment
- Number of Datacenters





THE END