Scheduling Jobs Across Geo-distributed Datacenters

PRESENTED BY HYEYUN SHIN

Motivation & challenges

- Growing data volumes across geo-distributed datacenters
- Inefficient to aggregate all data at a single datacenter
- Trend is to distribute computation for efficiency
- e.g. Data intensive jobs run by cluster computation systems such as Hadoop, Spark, etc.

Motivation & challenges (cont'd)

Centralized job execution

- How about when a job needs data from multiple datacenters?
  - Substantial network traffic
  - Increased job completion time
  - Data replication across multiple datacenters
  - Some data restricted to certain location

Motivation & challenges (cont'd)

Distributed job execution

- Bandwidth savings
- Shortened job completion time
- Faster data query
- Reduction in bandwidth costs

Classical SRPT scheduling is not optimal

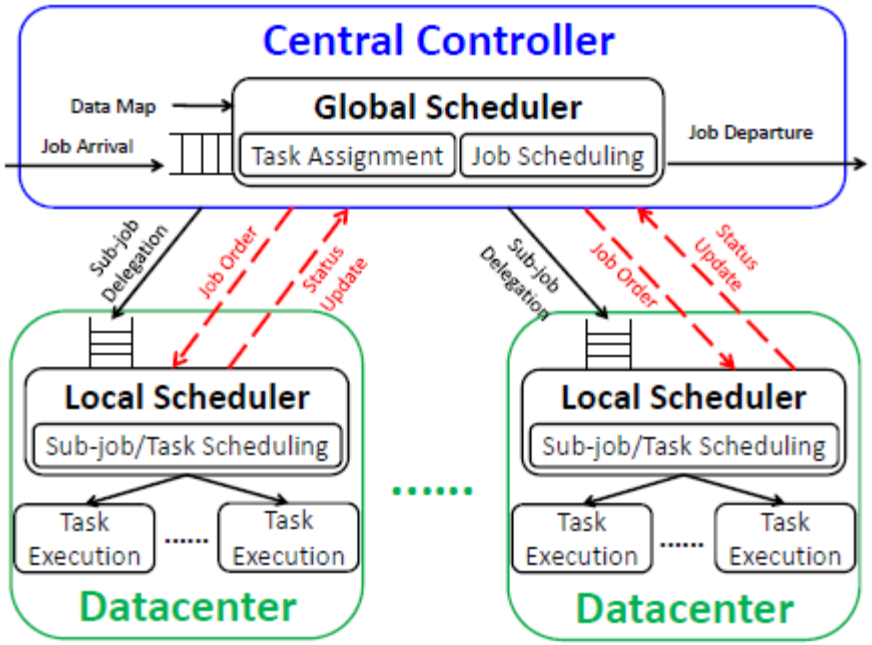
(Shortest-Remaining-Processing-Time)

Paper presents...

Reordering and SWAG

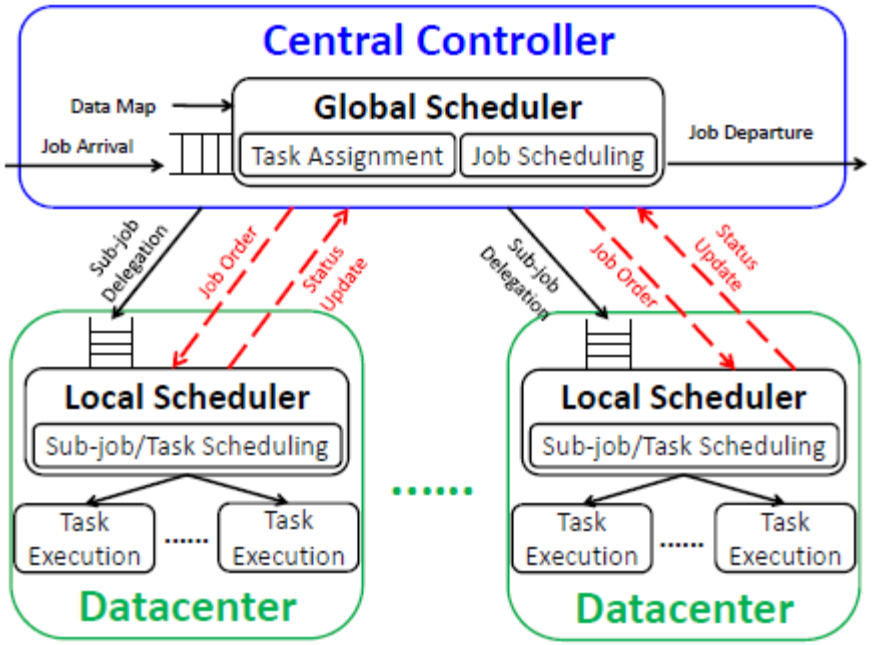
(Workload-Aware Greedy Scheduling)

Distributed job execution



- Global scheduler in Central controller
  - Job-level scheduling decisions for all jobs
  - Assigns a job's tasks to datacenters that host input data
- Local scheduler has a queue of tasks
  - Launches tasks at next available computing slot
  - Job order is determined by global scheduler or local scheduler
  - Report progress to central controller

Distributed job execution (cont'd)



- A job is completed when all its tasks are finished
- Job completion time is determined by its last completed task
- **Goal**: Reduce average job completion time

Terminology

- Sub-job: Subset of job's tasks assigned to the same datacenter
- Job completion time: Job finishing time – Job arrival time
- A sub-job's finish instant: queue index at which sub-job ends
- A job's finish instant: maximum finish instant of all its sub-jobs

Global-SRPT

- Heuristic that computes jobs priority on jobs' total remaining size across all datacenters

- Central controller:
  - Knows global state of current jobs' remaining tasks across all datacenters
  - Passes the job order to all datacenters

Example: Global-SRPT

- Three datacenters (DC)
- Three jobs arrived to systems in order of Job A, Job B, and Job C

Job ID	Arrival Seq	RT* in DC1	RT in DC2	RT in DC3	Total RT
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13

*RT: Remaining Tasks

- What would be the job order under Global-SRPT?

Independent-SRPT

- Heuristic that lets each datacenter independently decide on their jobs priority using SRPT

- Each datacenter:
  - Performs SRPT on its own
  - Prioritizes its sub-jobs based on their sizes
  - Updates the queue order independently

Example: Independent-SRPT

- Three datacenters (DC)
- Three jobs arrived to systems in order of Job A, Job B, and Job C

Job ID	Arrival Seq	RT* in DC1	RT in DC2	RT in DC3	Total RT
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13

*RT: Remaining Tasks

- What would be the job order under Independent-SRPT?

Example: Independent-SRPT

- Three datacenters (DC)
- Three jobs arrived to systems in order of Job A, Job B, and Job C

Job ID	Arrival Seq	RT* in DC1	RT in DC2	RT in DC3	Total RT
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13

*RT: Remaining Tasks

- What would be the job order under Independent-SRPT?

Example: Independent-SRPT

- Three datacenters (DC)
- Three jobs arrived to systems in order of Job A, Job B, and Job C

Job ID	Arrival Seq	RT* in DC1	RT in DC2	RT in DC3	Total RT
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13

*RT: Remaining Tasks

- What would be the job order under Independent-SRPT?

Shortcomings of SRPT

- Each job may have multiple sub-jobs across all datacenters
- Sizes of sub-jobs may be imbalanced

- Example: Global-SRPT

Job ID	Arrival Seq	RT* in DC1	RT in DC2	RT in DC3	Total RT
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13

Paper presents…

Reordering

A lightweight add-on to any scheduling approach

Reordering

- Continue moving sub-jobs later in a local queue
  - as long as delaying them does not increase the overall completion time of the job

1. Identify a datacenter with the longest queue length
2. Get a job from its queue with maximum finish instant
3. Add the job into N, a queue data structure
4. Remove all its related sub-jobs from all queues
5. Repeat until all current jobs in the system are added to N
6. The final job order by Reordering is the reverse of N

Example Revisited: Global-SRPT

- Three datacenters (DC)
- Three jobs arrived to systems in order of Job A, Job B, and Job C

Job ID	Arrival Seq	RT* in DC1	RT in DC2	RT in DC3	Total RT
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13

*RT: Remaining Tasks

- What would be the job order under Global-SRPT with Reordering?

Workload-aware approach

Theorem: Reordering does not degrade performance for any scheduling algorithm

Can we do better than B->C->A?

Job ID	Arrival Seq	RT* in DC1	RT in DC2	RT in DC3	Total RT
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13

Paper presents...

SWAG

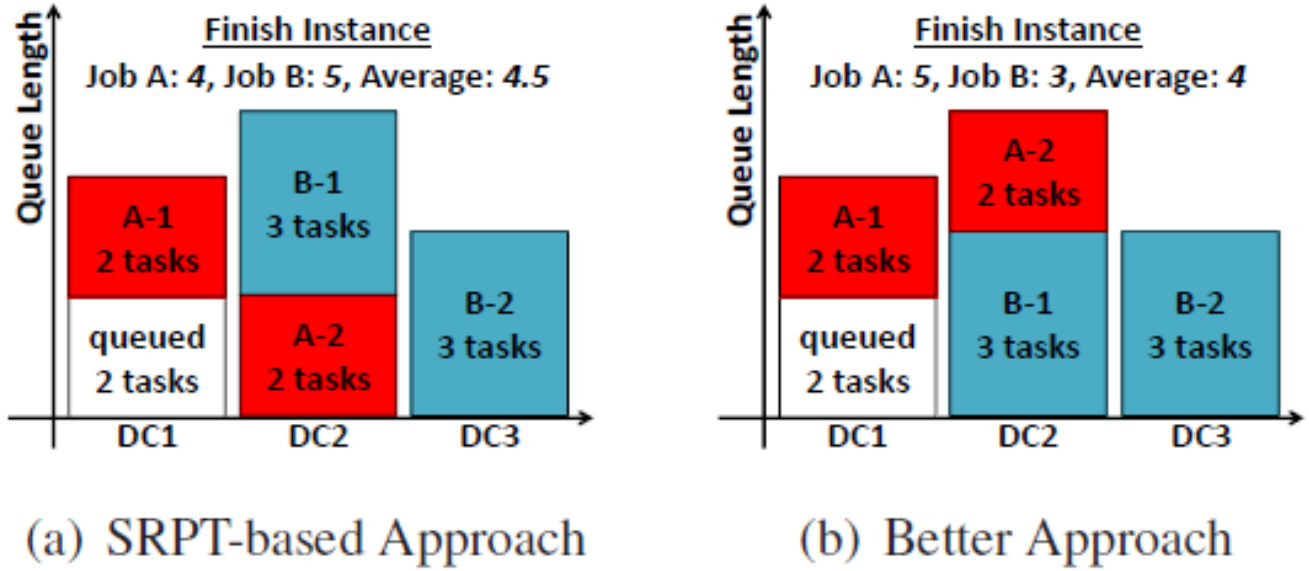
Workload-Aware Greedy Scheduling

SWAG Design principle

1. Jobs that can finish quickly should be schedules before others
2. Consider scheduling based on sub-job sizes rather than the size of the overall job
3. Consider the local queue sizes in assessing the finish times of sub-jobs

SWAG Design principle (cont'd)

3. Consider the local queue sizes in assessing the finish times of sub-jobs



(a) SRPT-based Approach (b) Better Approach

SWAG Algorithm

- Greedily prioritizes jobs by computing their estimated finish times based on the current queue length, as well as the job's remaining size

- Central controller runs SWAG at new job arrival or departure
- The new order is computed from scratch based on the estimated job finish times

Terminology

- Makespan(j): For each datacenter, add the current queue length and the size of sub-jobs of job *J*. Makespan(j) is the biggest value out of all

SWAG Algorithm (cont'd)

1. Compute Makespan for each job
2. Select the job with minimal makespan
3. Append the job into the job order
4. Update the queue length based on the selected job's sub-job sizes (if more than one job with minimal makespan, pick the one with the smallest total remaining size)
5. Repeat

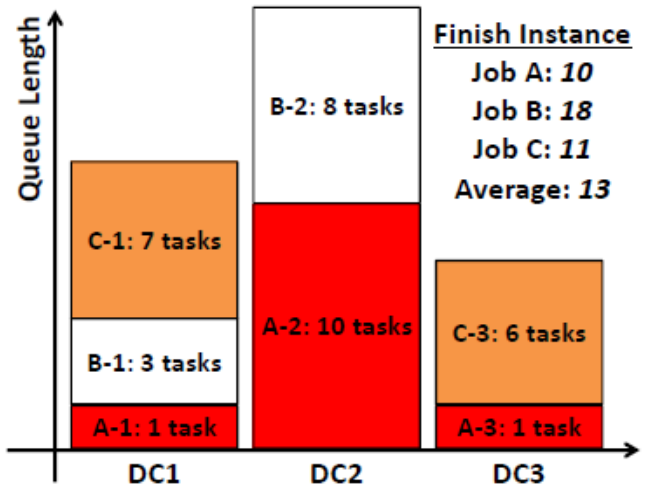
Example: SWAG

- Three datacenters (DC)
- Three jobs arrived to systems in order of Job A, Job B, and Job C

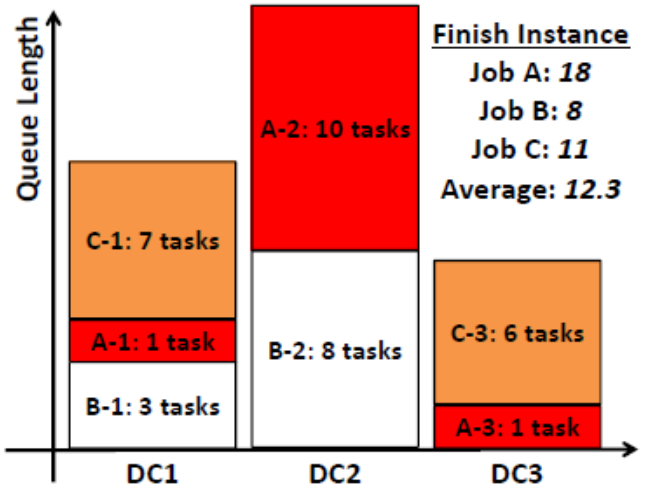
Job ID	Arrival Seq	RT* in DC1	RT in DC2	RT in DC3	Total RT
A	1	1	10	1	12
B	2	3	8	0	11
C	3	7	0	6	13

*RT: Remaining Tasks

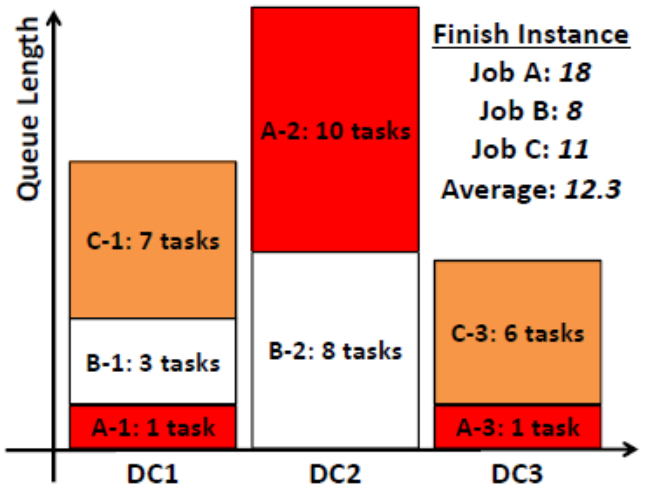
- What would be the job order under SWAG?



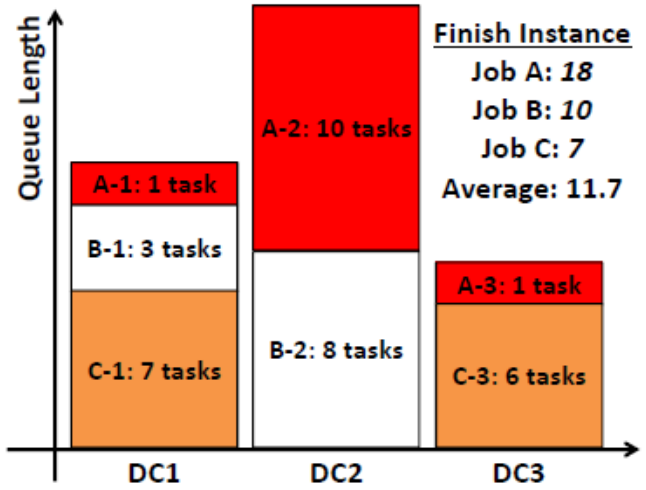
(a) FCFS



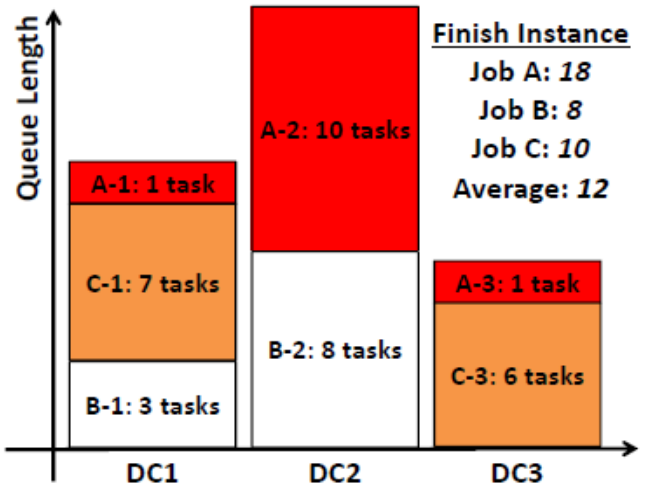
(b) Global-SRPT



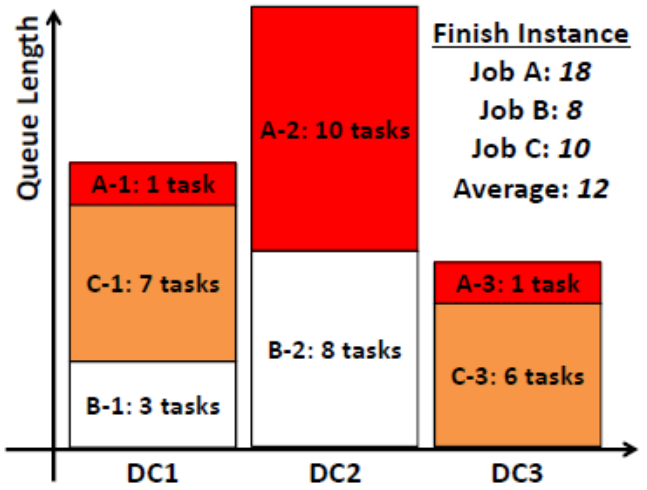
(c) Independent-SRPT



(d) *SWAG*



(e) Global-SRPT w/*Reordering*



(f) Independent-SRPT w/*Reordering*

Experiment settings

- Synthetic workloads with job size distributions from:
  - Facebook's production Hadoop cluster
  - Google cluster workload trace
  - Exponential Distributions
- Adjusted job's inter-arrival times of workloads based on Poisson Process
- Tasks duration modeled by Pareto distribution according to Facebook workload information (average task duration being 2 seconds)
- Zipf distribution to model the skewness of task assignment among datacenters
- Default number of datacenters is 30, with 300 computing slots per datacenter (78% system utilization)

Experiment settings (cont'd)

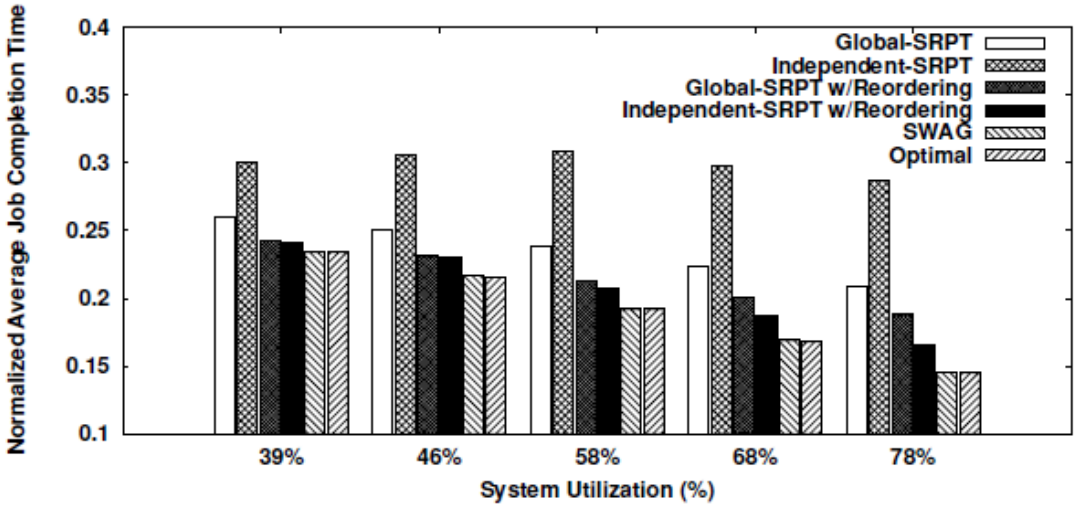
- FCFS (baseline)
- Global-SRPT
- Independent-SRPT
- Global-SRPT with Reordering
- Independent-SRPT with Reordering
- SWAG
- Optimal Scheduling (offline brute-force search with full knowledge of future job arrivals and actual tasks duration)

Performance results

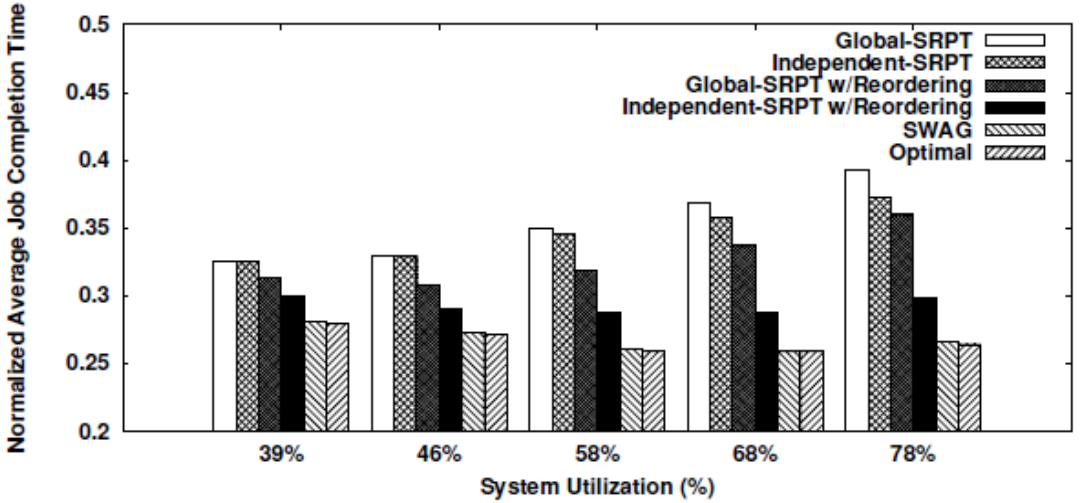
- Reordering
  - Improves by as much as 27% under highly utilized settings, and 17% under lower utilization
  - More beneficial to Independent-SRPT than to Global-SRPT
- SWAG (compared with SRPT)
  - Improves under higher utilization up to:
    - 50% (Facebook)
    - 29% (Google)
    - 35% (Exponential)
  - At least 12% improvement under lower utilization

Performance results (cont'd)

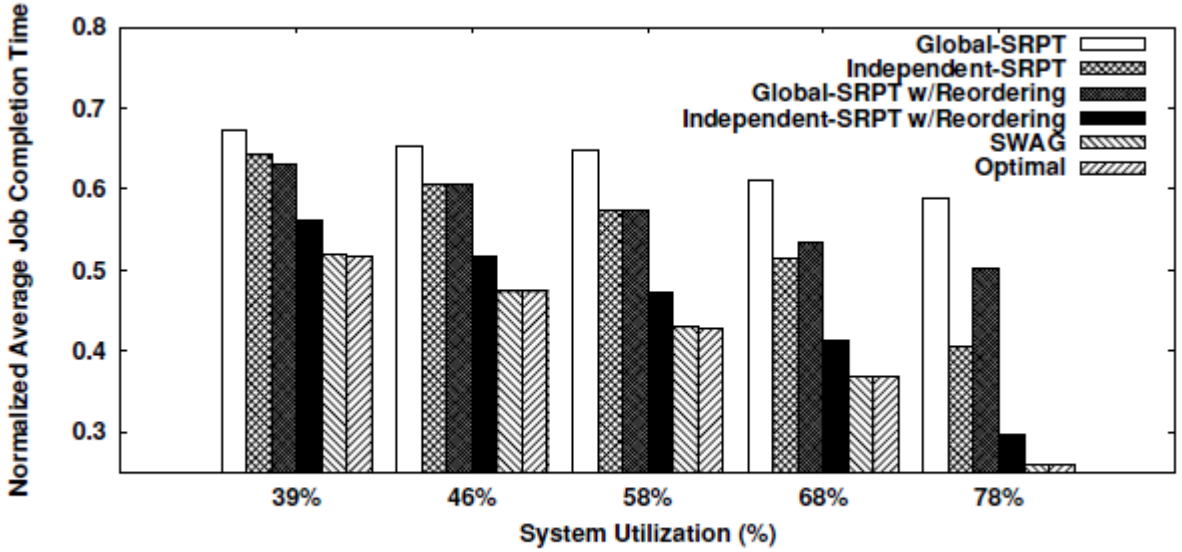
- Job fairness with Facebook trace
  - Slowdown of large jobs for Independent-SRPT is 40% more than its overall slowdown
  - Gap is no more than 30% for Independent-SRPT with Reordering
  - Gap is no more than 25% for SWAG
- Job fairness with Facebook trace
  - Google trace shows huge gap of slowdown as most jobs are small
  - Still, Independent-SRPT and Reordering and SWAG does relatively well



(a) Performance with Facebook Trace

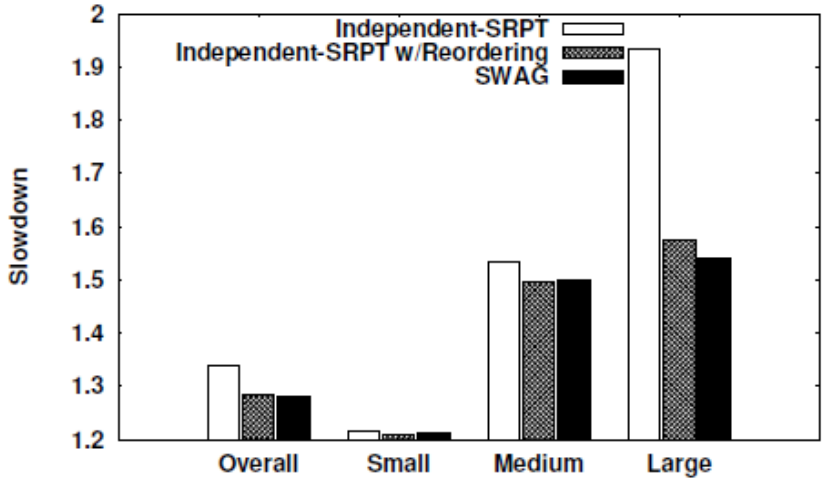


(c) Performance with Google Trace

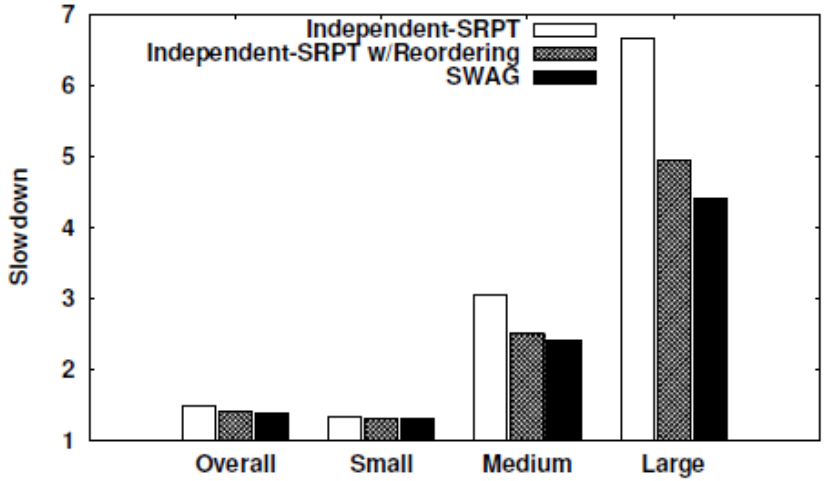


(e) Performance with Exponential Trace

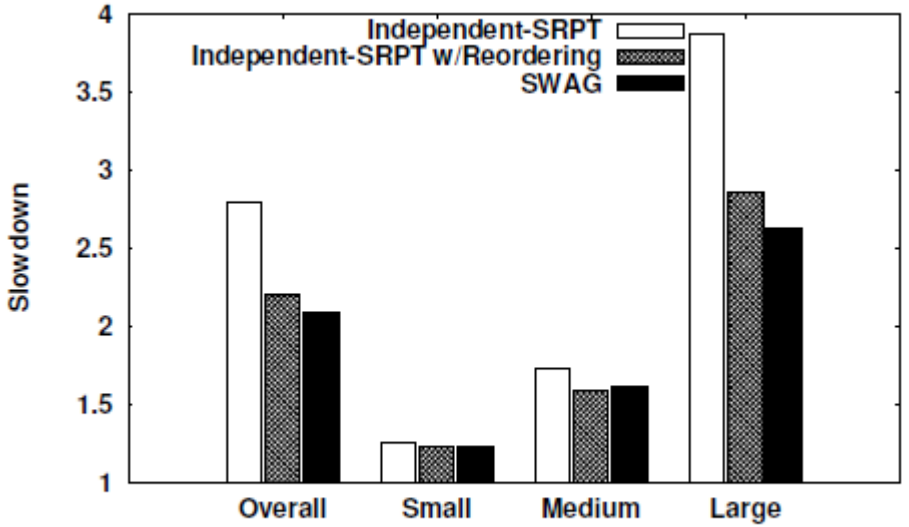
Performance



(b) Fairness with Facebook Trace



(d) Fairness with Google Trace



(f) Fairness with Exponential Trace

Fairness

Conclusions

- As data volumes increase, running jobs across geo-distributed datacenters emerges as the promising trend
- Reordering improves scheduling algorithm by adjusting job order
- SWAG improves the average job completion time and achieves near-optimal performance
- Reordering and SWAG improved the average job completion time up to 27% and 50% respectively compared to SRPT-based approach