A Cleanup Algorithm for Implementing Storage Constraints in Scientific Workflow Executions

Sudarshan Srinivasan

Department of Computer Science and Engineering

Indian Institute of Technology, Hyderabad

email2sudarshan@gmail.com

Gideon Juve, **Rafael Ferreira da Silva,**

Karan Vahi, Ewa Deelman

Information Sciences Institute
University of Southern California
{gideon,rafsilva,vahi,deelman}@isi.edu

USC Viterbi
School of Engineering

Information Sciences Institute

Problem

- **Data-intensive workflow**

- **Disk space is limited (storage constraint)**
  - Machines may not have enough disk space
  - Quotas may impose caps on disk usage
  - Want to reduce or limit use of resources

- **Need to remove data as workflow is running in order to free enough space to finish the workflow**

- **It may not be possible to execute the workflow**
  - Identifying the minimum storage required is hard
  - But we can compute some bounds

USCViterbi
School of Engineering
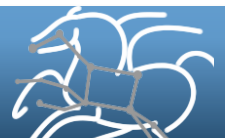
Assumptions

- **Storage constraint is given**

- **Workflow is modeled as a DAG**
  - **Nodes: Tasks**
  - **Edges: Data flow dependencies**

- **Input/output files for each task are known**

- **Size of each file is known**
  - **Or at least a reasonable estimate**
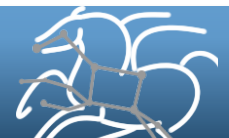
Previous Solutions

- **Manual dependencies and cleanup tasks**
  - Forces a certain ordering of tasks that results in smaller footprint
  - Cleanup removes data

- **Partitioning**
  - Split up tasks across several sites based on available storage
  - Does not work for a single site
  - Does not work if total available storage < workflow size
  - Transfers may cause performance problems (can be minimized)

- **Cleanup task algorithms**
  - Add tasks to the workflow that remove data when it is not needed
  - One task for each file – Generates lots of cleanup tasks
 - Clustering – Still may cleanup tasks (1 per task)
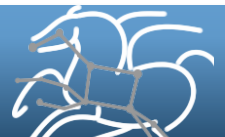
Problems with Previous Solutions

- **Typically require development of a data-aware scheduler**
  - May not be feasible on some infrastructures

- **Online solutions can result in deadlock**
  - Backtracking required to resolve the problem
  - Particularly problematic if no solution is possible

- **Cleanup approaches can hurt performance**
  - Often result in too many cleanup tasks
  - Can increase workflow makespan

- **Many don't provide any guarantees about disk usage**

USC Viterbi
School of Engineering

Goals

- **Provide some guarantee about storage used by workflow**
  - No deadlocks (if solution found and estimates are accurate)

- **No modifications to scheduler**
  - Only requires DAG engine

- **Minimize impact on performance**
  - Few cleanup tasks
  - Reduce bottlenecks
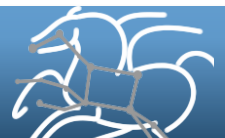
Approach

- **Storage-Constrained Cleanup Algorithm**

- **Adds cleanup tasks to the workflow at planning time**

- **Cleanup tasks added only when and where they are needed**

- **Makes non-cleanup tasks depend on cleanup tasks in order to ensure that space is available at each step of the workflow**
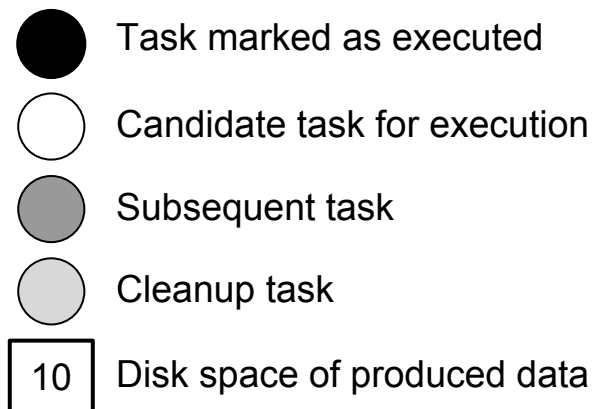
USC Viterbi
School of Engineering
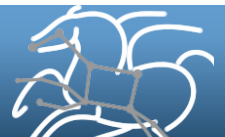
Storage-Constrained Cleanup Algorithm

1. Choose a ready task to schedule

2. If space is available: run the task

3. If enough space can be cleaned up to let the task run:

    3.1 Create one or more cleanup tasks to remove all of the eligible files

    3.2 Make queued jobs depend on cleanup tasks

    3.3 Make cleanup tasks depend on tasks that use cleaned up files

    3.4 Mark task as finished, queue additional tasks

4. If no more data can be cleaned up:

    4.1 Report failure

5. If more ready tasks: goto 1

6. Add leaf cleanup task, return updated DAG

Example

- **Storage limit set to 200 units**

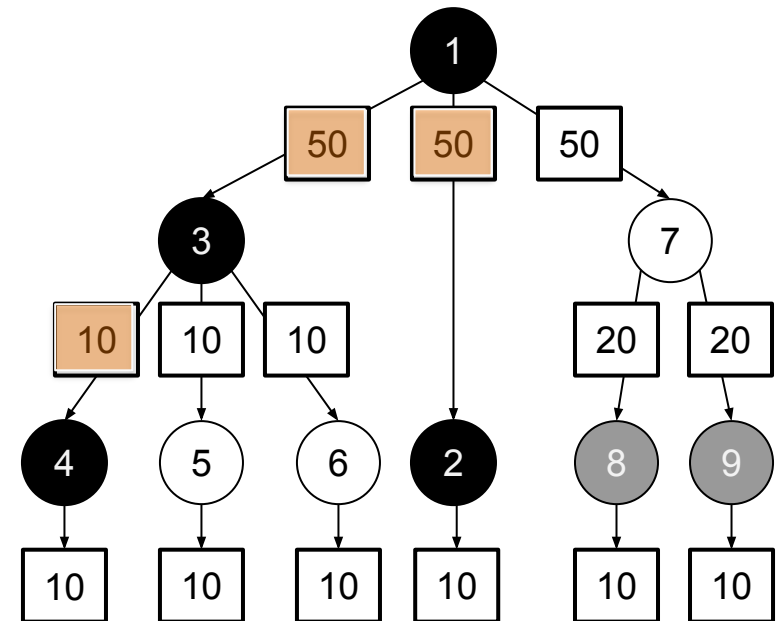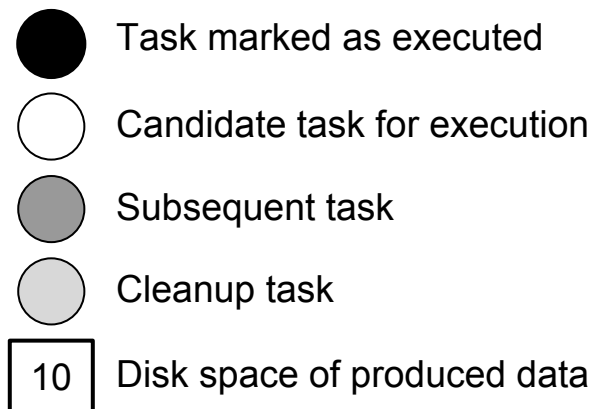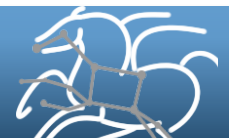- **Algorithm proceeds until there is insufficient disk space to run the next task**

Legend:

- ● Task marked as executed
- ○ Candidate task for execution
- ● Subsequent task
- ○ Cleanup task
- | 10 | Disk space of produced data
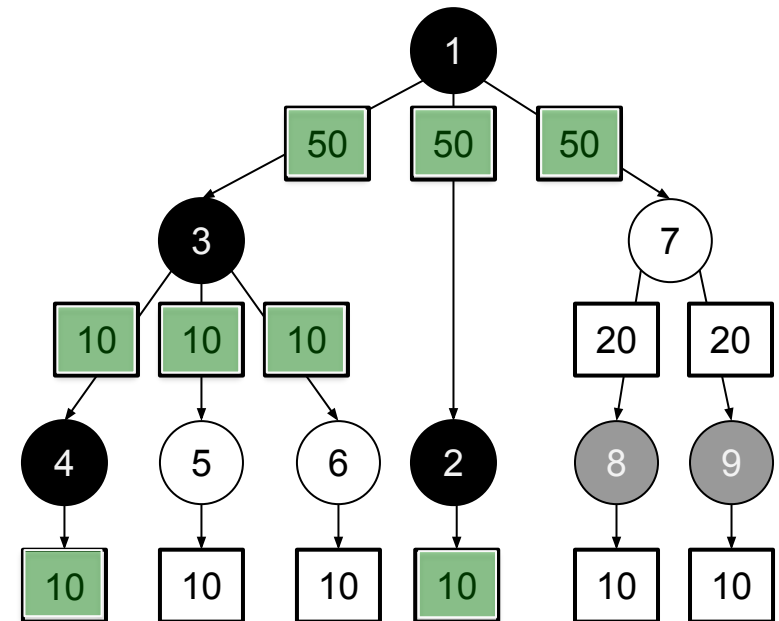
200 units are used

Example

- **Storage limit set to 200 units**

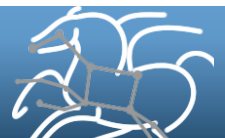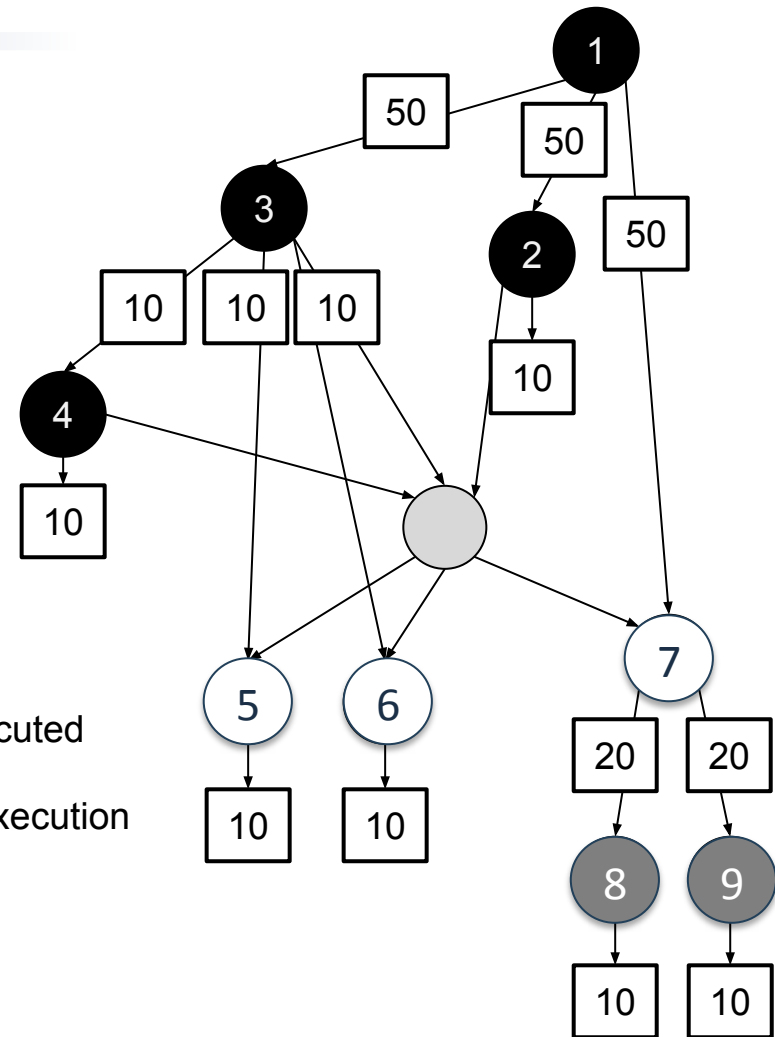- **Algorithm proceeds until there is insufficient disk space to run the next task**

Task marked as executed

Candidate task for execution

Subsequent task

Cleanup task
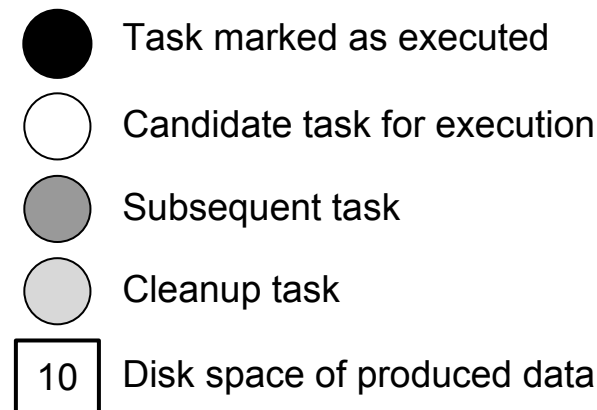
10   Disk space of produced data
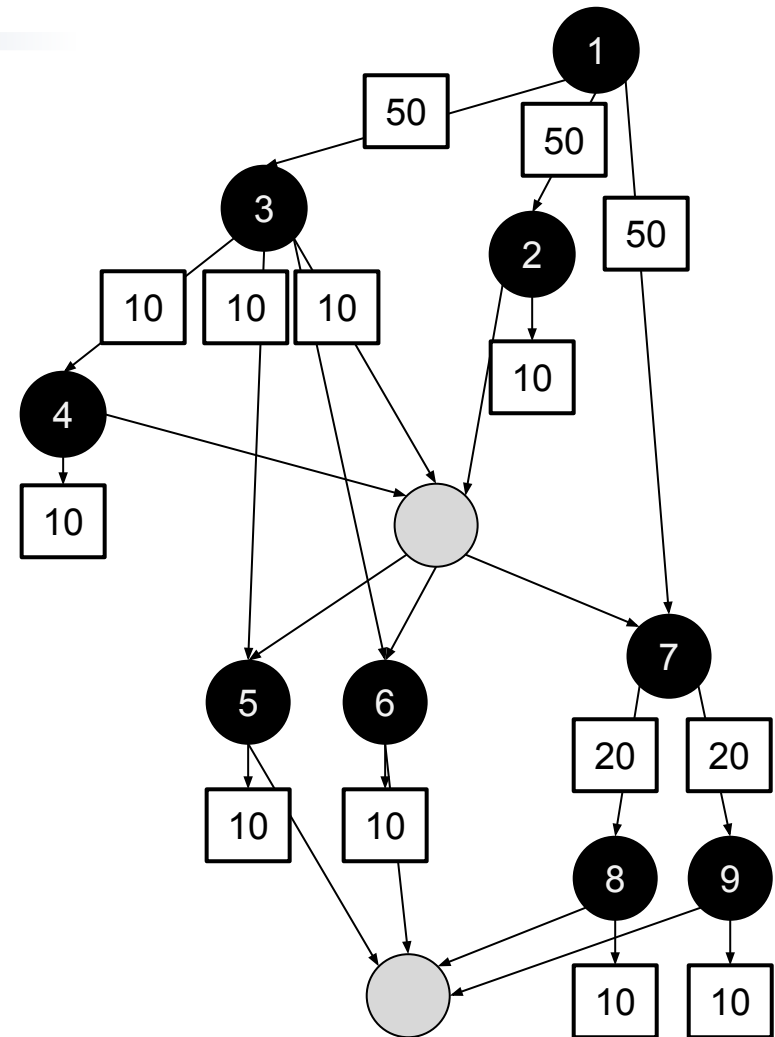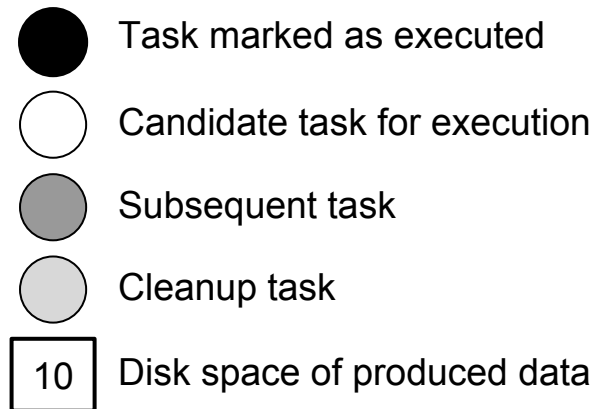
110 units can be removed

Example

- **Cleanup task removes all data that is no longer required**

- **Depends on tasks that used the files that were removed**

- **All queued tasks depend on cleanup task**

Legend:
- ● Task marked as executed
- ○ Candidate task for execution
- (dark gray) Subsequent task
- (light gray) Cleanup task
- [10] Disk space of produced data

Example

- **A final cleanup task is inserted to ensure that all intermediate data is removed**

Task marked as executed

Candidate task for execution

Subsequent task

Cleanup task

| 10 | Disk space of produced data

Heuristics for selecting a task (Step 1)

- **Max Freed**
  - Select the task that maximizes the amount of data that can be cleaned up

- **Min Required**
  - Select the task that requires the least amount of storage space (smallest output) **– Make more progress before cleanup**

- **Max Required**
  - Select the task that requires the largest amount of storage space (largest output) **– Most difficult to accommodate**

- **Balance Factor**
  - Select task with largest "balance factor" **– Difference between space freed, and space required**
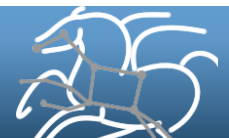
Heuristics for creating cleanup tasks (Step 3.1)

- **Single Task**
  - Create one cleanup task to remove all of the files

- **Queued Tasks**
  - Create one cleanup task for each queued task

- **Random Tasks**
  - Adds a random number between 1 and the number of queued tasks

- **Resources Tasks**
  - Adds cleanup tasks up to the number of resources

- **Note:**
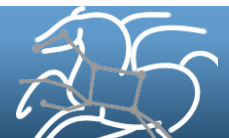 - Not more than than the number of files being removed
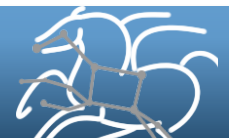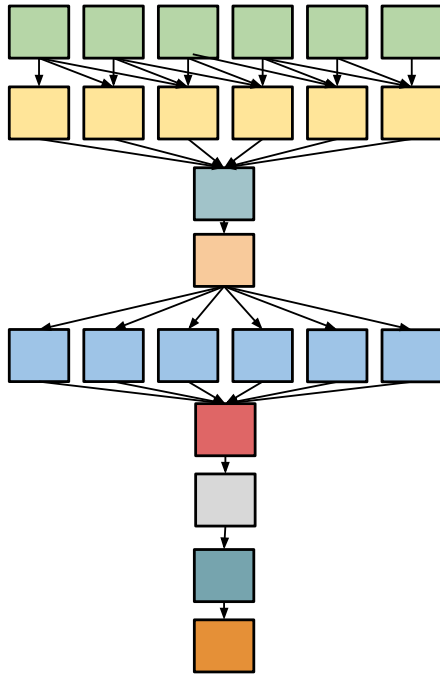
Evaluation – Alternative algorithm

- **Compare proposed algorithm with algorithm by Singh, et al.**

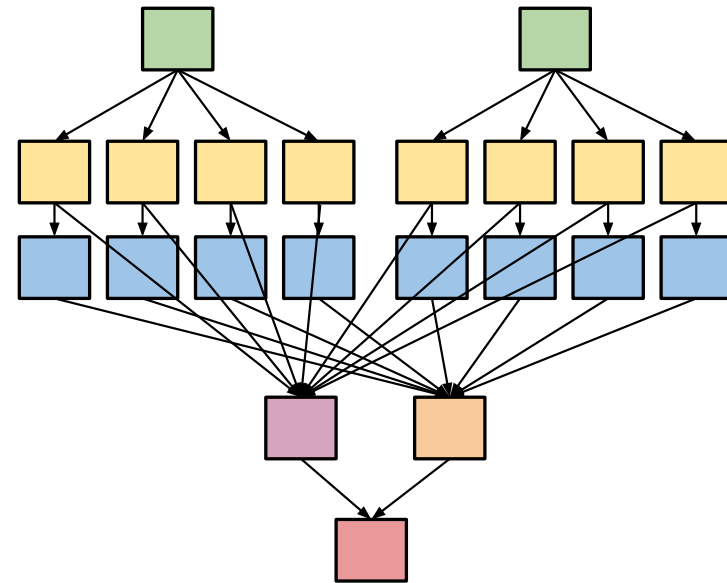- **Singh's algorithm is the default cleanup algorithm in Pegasus**
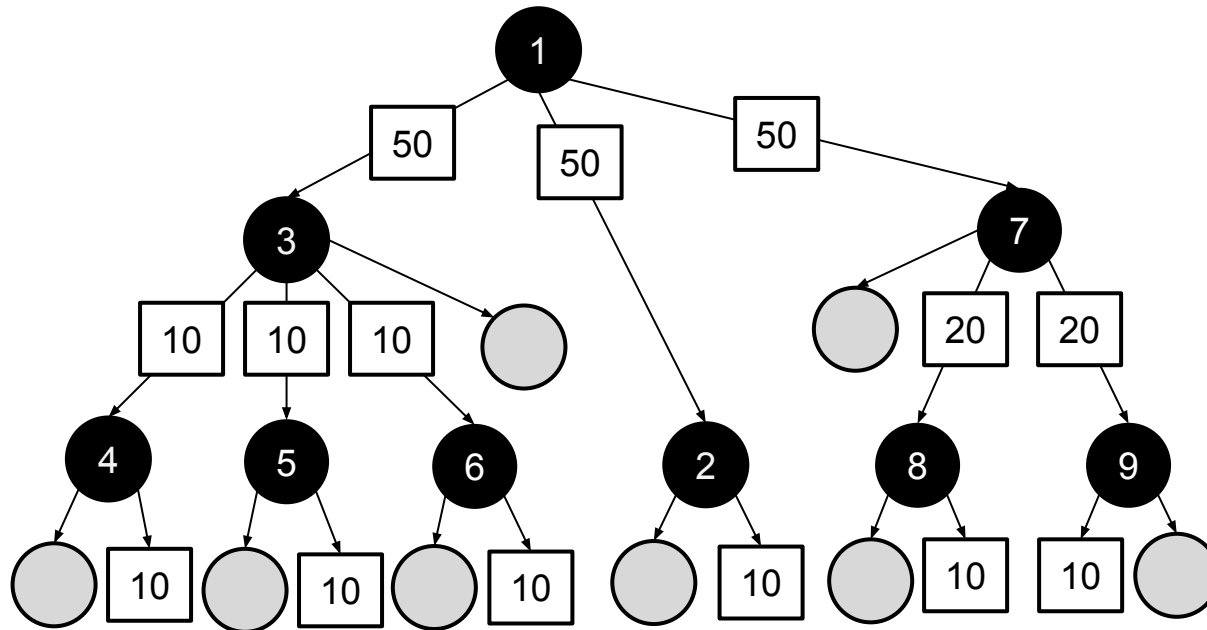
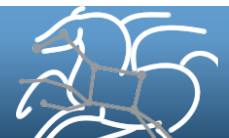**DAG generated by Singh's algorithm**

# Evaluation – Applications

Montage

CyberShake

- Generated synthetic workflows based on real application

- Most experiments used workflows with 1000 tasks

Evaluation – Simulator

- **Simulator based on CloudSim framework**

- **Parameterized with values from a previous paper on workflow overheads, and some experiments**

- **Priority based scheduling with randomization**
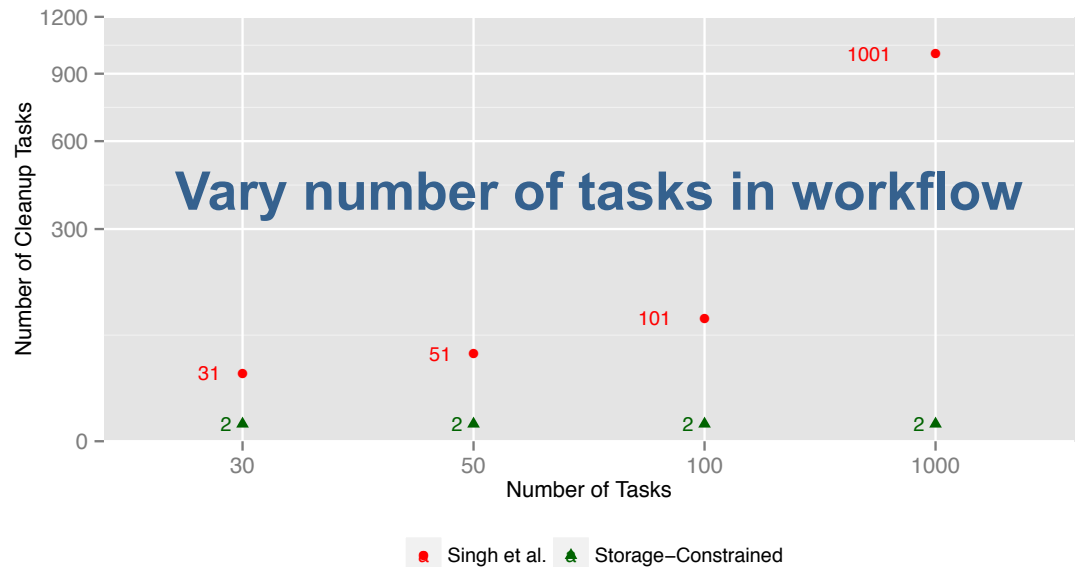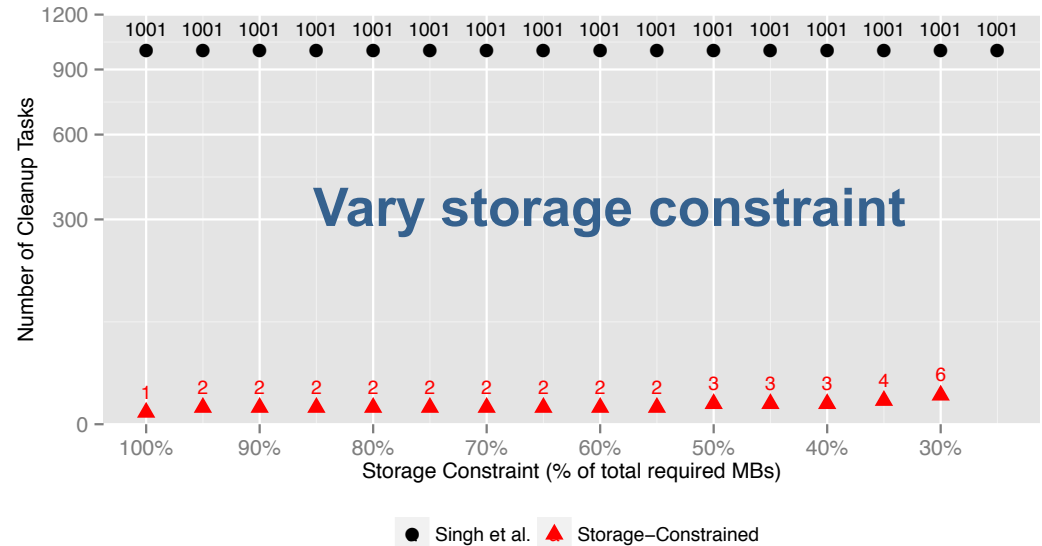
- **100 simulation runs for most data points**
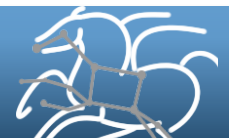
Experiment 1 – Ability to meet storage constraint

- **Cleanup tasks are prioritized**

- **Constraint set to 40% of maximum storage**

- **Montage results (CS is similar)**

- **New algorithm doesn't exceed constraint. Existing algorithm is ok on fewer resources.**
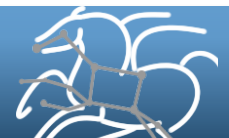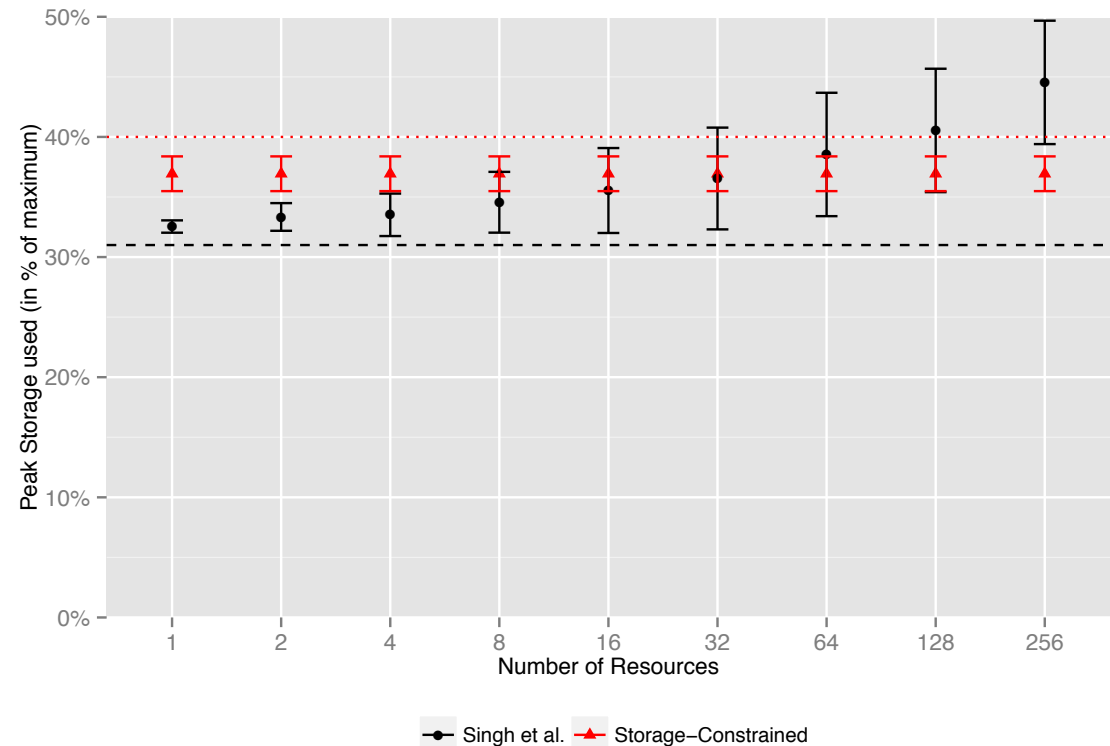
Experiment 2 – Number of cleanup tasks

- **Compare the number of cleanup tasks generated by both algorithms**

- **CyberShake results (Montage is similar)**

- **New algorithm generated far fewer cleanup jobs**

Vary storage constraint

Legend: ● Singh et al. ▲ Storage-Constrained

Vary number of tasks in workflow

Legend: ● Singh et al. ▲ Storage-Constrained

Experiment 3 – Effect of cleanup on makespan

- **Vary the number of resources**

- **Storage constraint set to 75% of total workflow size**

- **New algorithm is much better for CyberShake, mixed results for Montage**
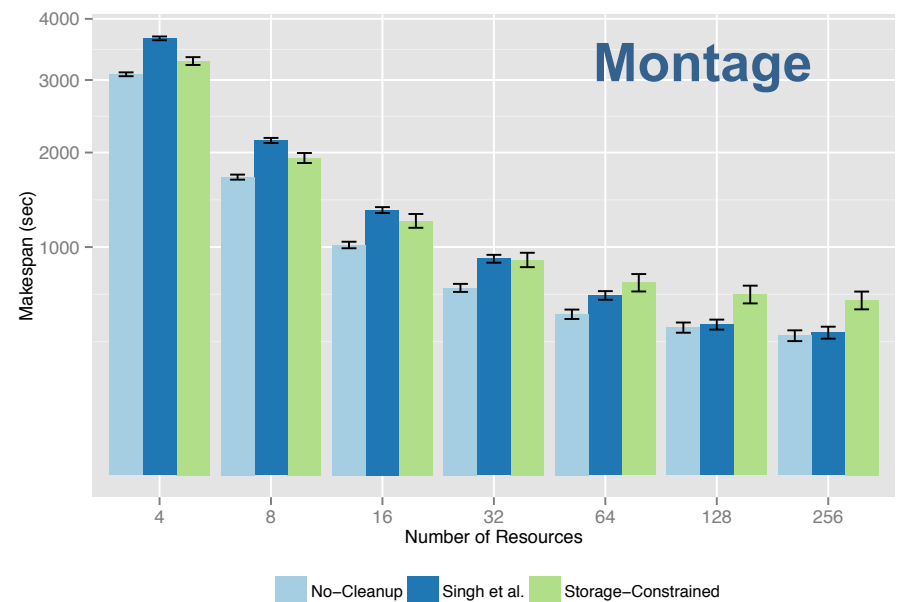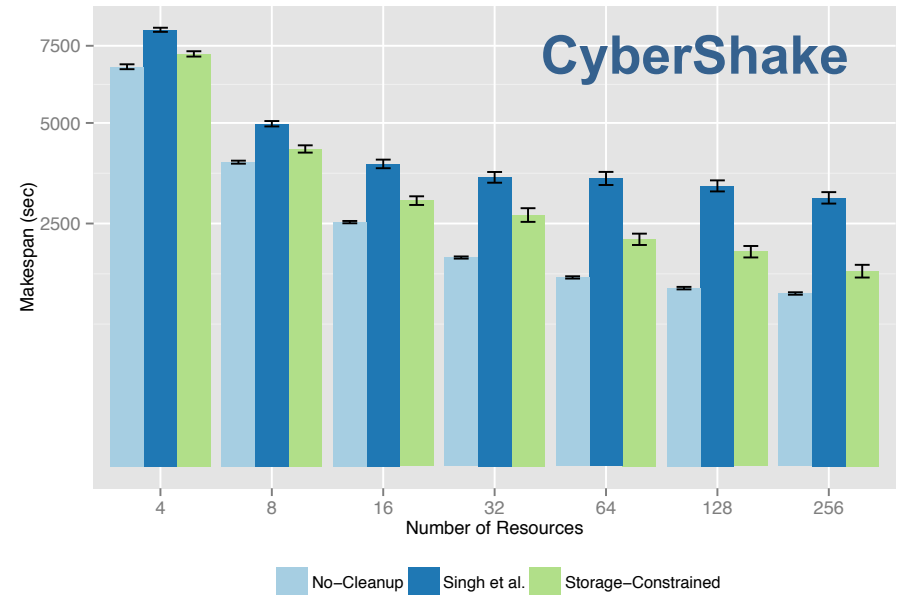
Experiment 4 – Heuristics for task selection

- **CyberShake results (Montage is similar)**

- **Not much effect on peak storage, but Max Freed is as you would expect**

- **For makespan, balance factor is usually better**
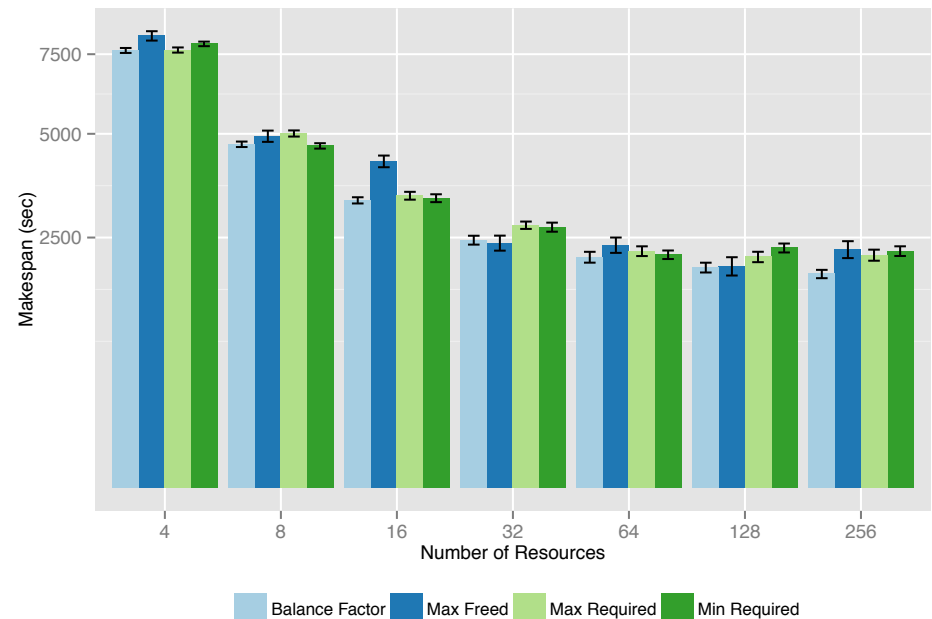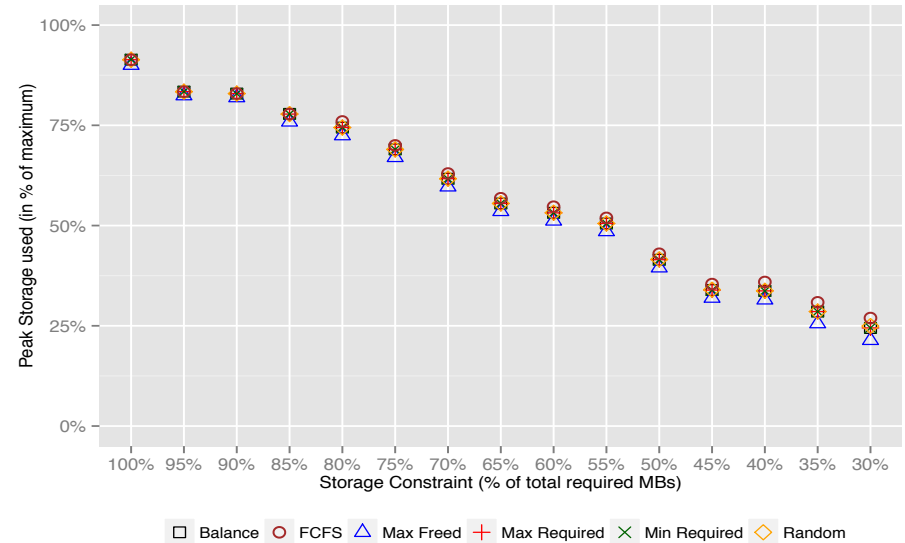
Experiment 5 – Heuristics for no. of cleanup tasks

- **30% storage constraint**

- **CyberShake results (Montage difference is relatively insignificant)**

- **Heuristic based on number of resources is best**
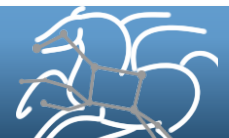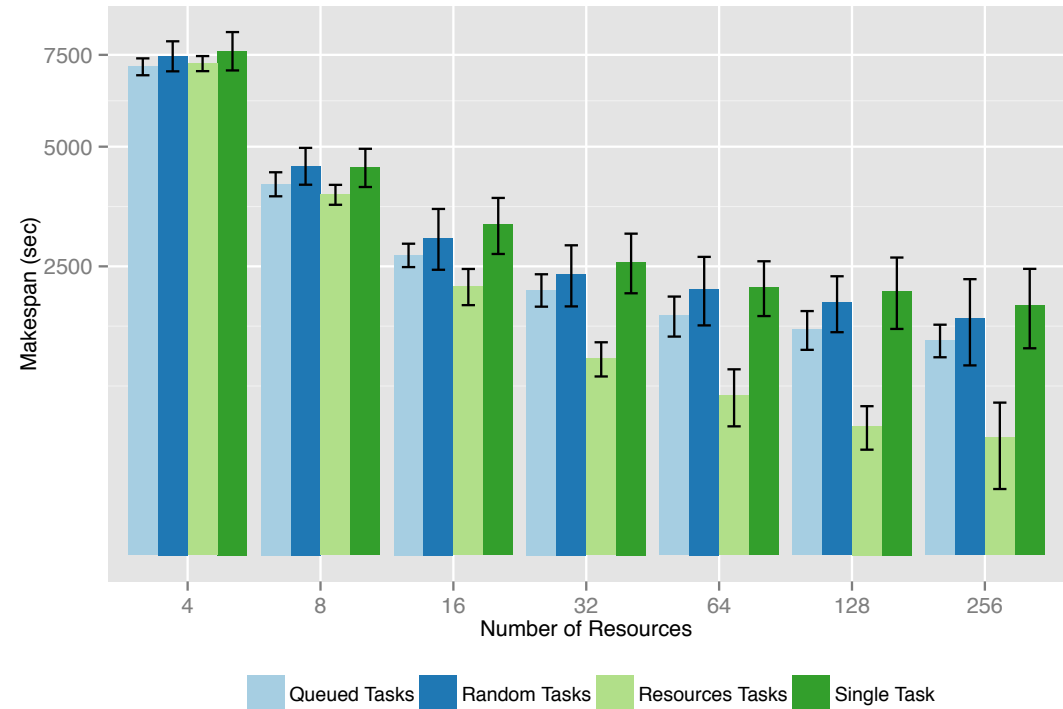
Conclusion

- **Proposed a new algorithm for storage constrained workflows that:**
  - Does not require a data-aware scheduler
  - Provides more guarantees about storage space used
  - Generates far fewer cleanup jobs that existing approaches
  - Often results in smaller makespan than existing cleanup approaches (depends on application)

- **Future work**
  - What if size estimates are wrong?
  - Handling workflows executed on multiple sites
  - Enhancements to reduce dependencies and improve parallelism