TEXAS TECH UNIVERSITY™

Log-assisted Straggler-aware I/O Scheduler for High-End Computing

Neda Tavakoli, Dong Dai, Yong Chen

Ninth International Workshop on
Parallel Programming Models and Systems Software for High-End Computing (P2S2), 2016

Outline

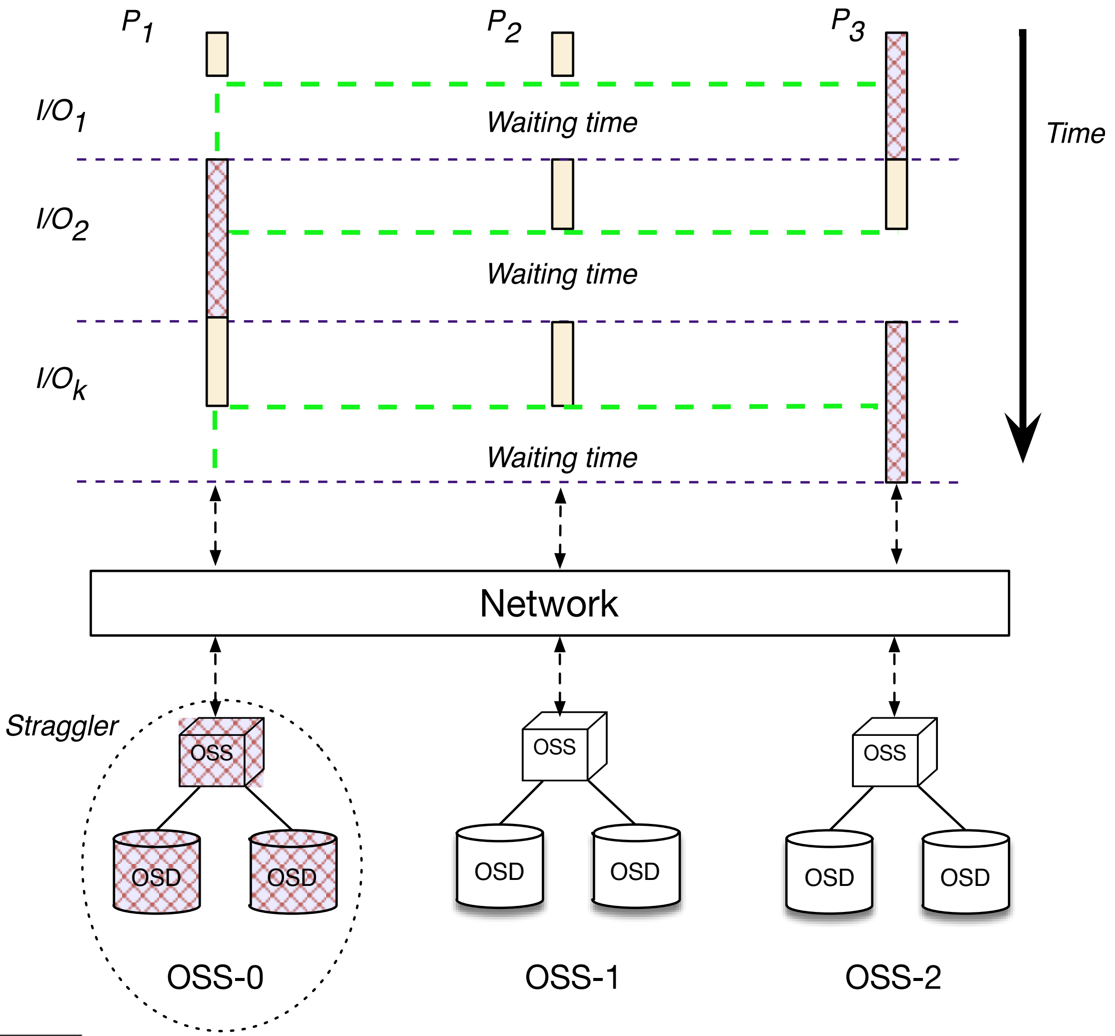
- ❑ Background and Motivation
  - ❑ Straggler Problem
- ❑ Overall Architecture
  - ❑ Scheduling Model
  - ❑ Scheduling Client-side Server Statistic Log
  - ❑ Servers Statistic Table
- ❑ Scheduling Algorithms
  - ❑ MLML
  - ❑ TRH
- ❑ Evaluation
- ❑ Conclusion

Background and Motivation

Background: Straggler Problem in I/O

- **Straggler**: Some of OSSs take a much longer time in responding to I/O requests than other servers.

- The occurrence of stragglers has significant affects on I/O performance of object storage systems.

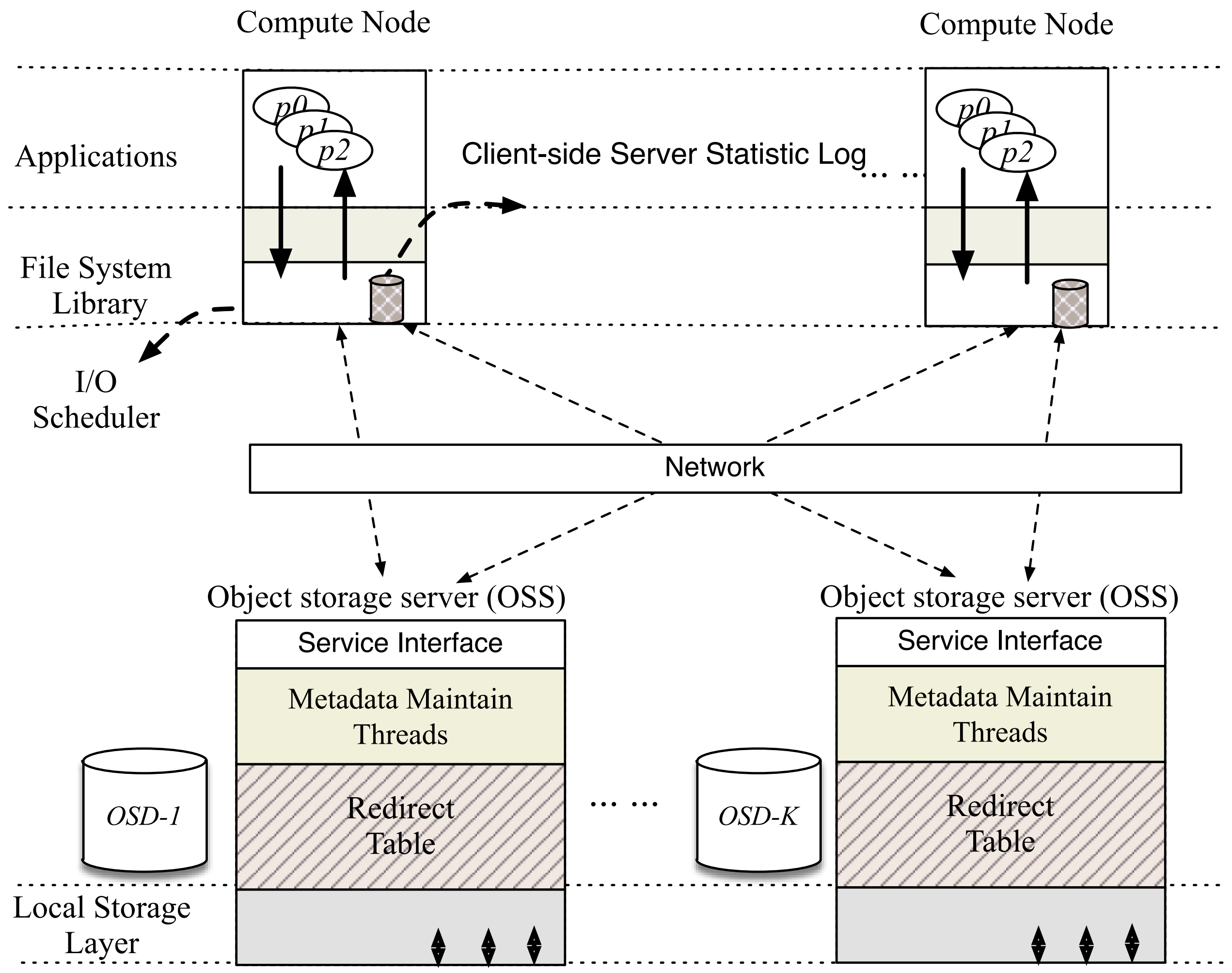


Background & Motivation

- **Our previous work**, two-choice randomized:
  - Solves the straggler problem.
  - For each I/O request, before making scheduling decision, probing strategy is used.
  - Two random servers will be probed, the one with the minimum load will be selected.
  - Suffer from expensive probing messages and communication overhead
- **In this study**:
  - Use client-side logs to assist *straggler-aware I/O scheduling*.

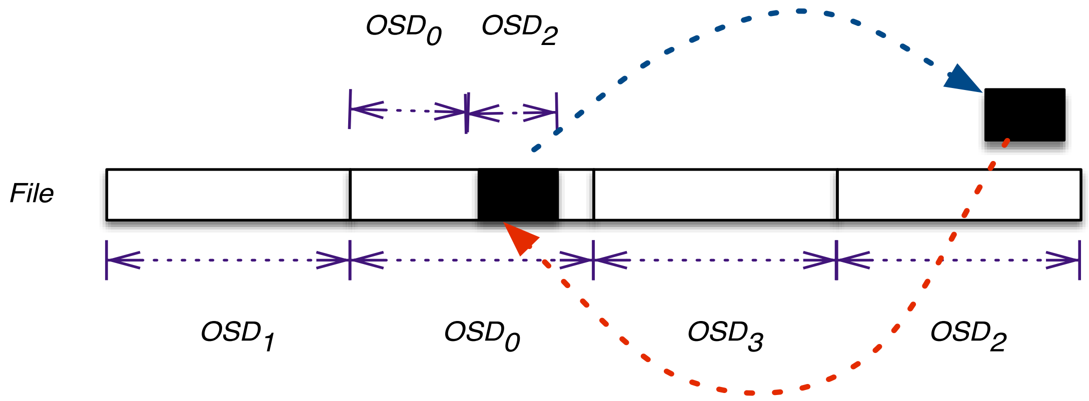
Overall Architecture

Proposed Architecture



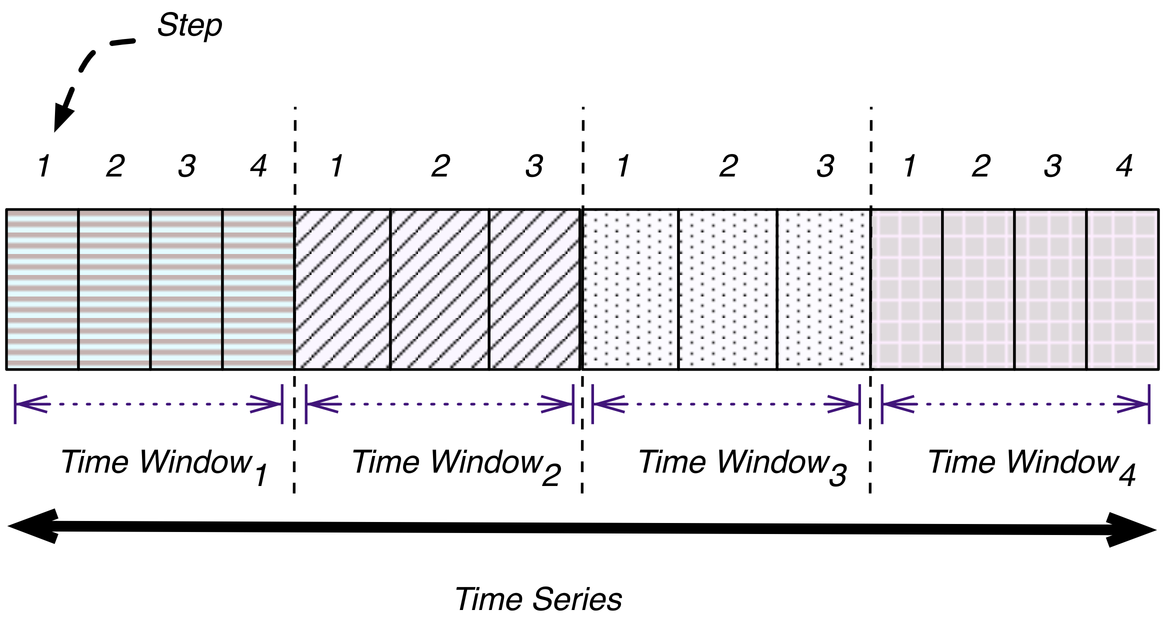
Proposed Architecture: Components

- ❑ **Client-side server statistic log**
    - ❑ Record current pending I/O requests and also server statistic
- ❑ **Metadata maintainer thread**
    - ❑ Runs in the background to move the redirected objects back to their default location when the file systems are idle.
- ❑ **Redirect Table**
    - ❑ Record the current location of chunk of data

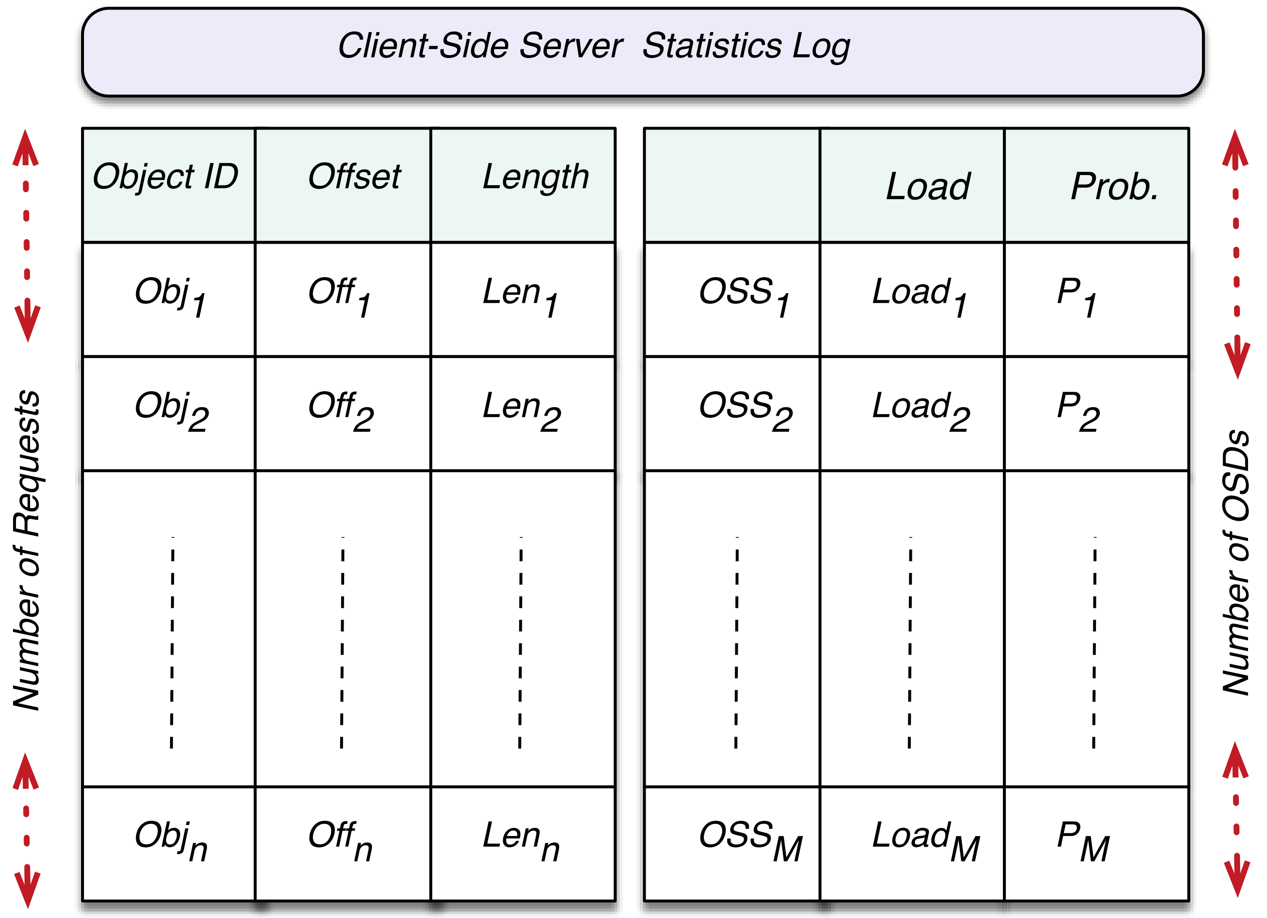


Scheduling Model

- **Time Series:** all the queued I/O requests.
- **Time Window:** we schedule I/O in a fixed time interval, called time window
- **Time Step:** each time window can contain multiple steps, each of them contains the I/O requests on the same object



Client-side Server Statistic Log Data Structure



Object ID	Offset	Length
Obj_1	Off_1	Len_1
Obj_2	Off_2	Len_2
⋮	⋮	⋮
Obj_n	Off_n	Len_n

	Load	Prob.
OSS_1	$Load_1$	P_1
OSS_2	$Load_2$	P_2
⋮	⋮	⋮
OSS_M	$Load_M$	P_M

Update Servers Statistic Logs

- After scheduling I/O request of a single step, the server loads in the server statistic log is updated to:

$$l = l' + Len$$

l : Expected server load at the end of each step.

l': Server load from the previous step.

Len: Length of scheduled I/O requests.

- Update the probability of selecting server i:

$$p_i = p_i' * e^{-l_i}$$

p_i': Indicates the current probability of selecting server i

Scheduling Algorithms

- Not only update the load of the selected server
- It needs to update the probability of choosing another server to prepare for the next scheduling. The following equation shows how this is done:

$$p_j = p'_j + \frac{(p'_i - p'_i * e^{-l_i})}{M-1}, j \neq i$$

M : indicates the total number of storage servers.

i : The server that is chosen to serve the current I/O request.

j: refers to all other object storage servers.

Scheduling Algorithms

Using Client-side Servers Statistic Logs

- ❑Based on the information stored in the server statistic log, we propose two scheduling algorithms.
- ❑These algorithms leverage the statistic log to develop an efficient straggler-aware I/O scheduler.
  - ❑**Firstly, *Max Length - Min Load (MLML)***
    - ❑select the server with lighter workloads with a higher probability.
  - ❑**Secondly, *Two Random from Top Half (TRH)***
    - ❑takes advantage of the random strategy in addition to server statistic log.

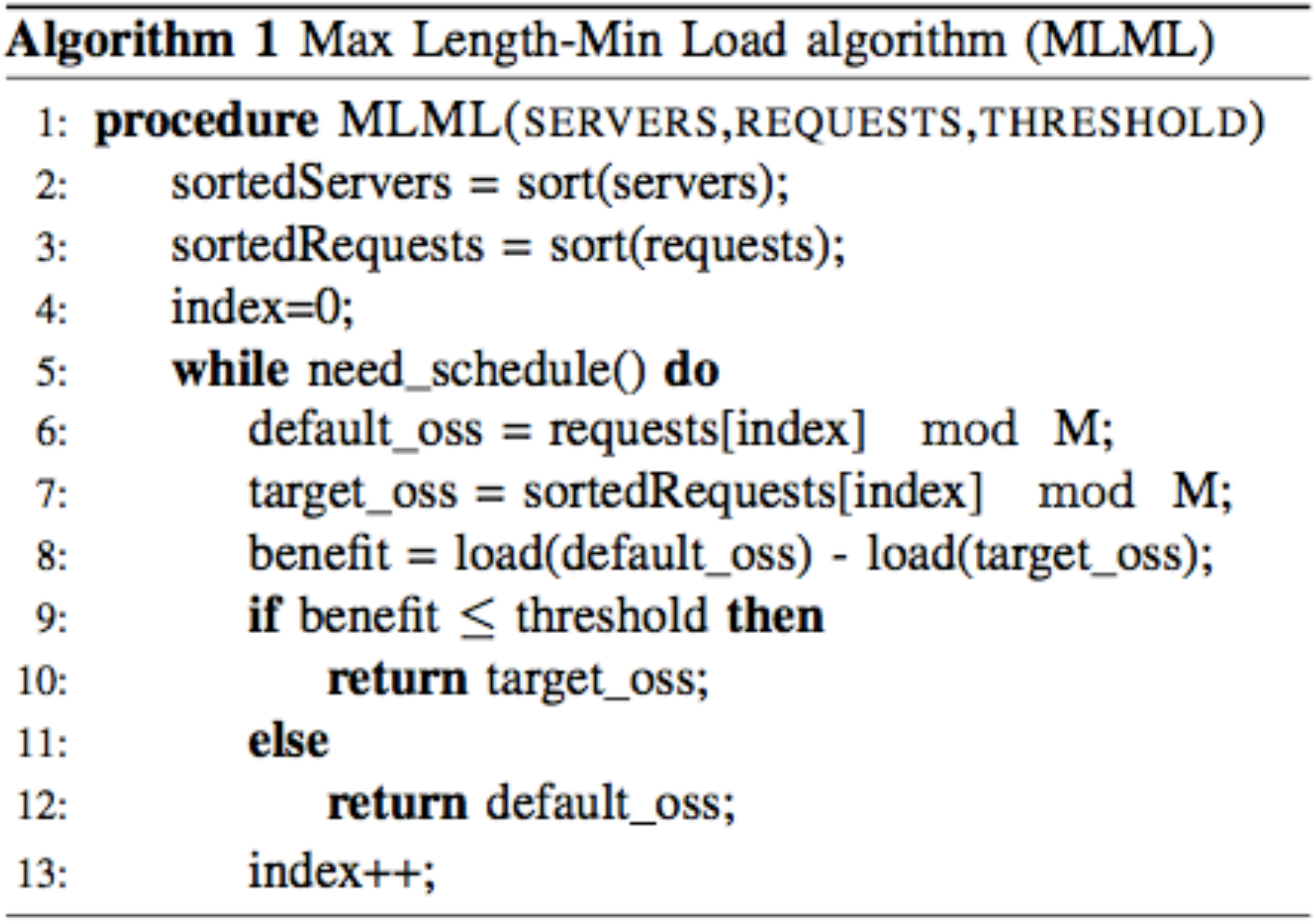
Proposed Algorithm: Max Length - Min Load MLML

- ❑It **sorts the storage servers** based on their probabilities from the highest to the lowest
- ❑**Sort the I/O requests** from the maximum length to the minimum one.
- ❑These two sorted lists are processed sequentially, from the top to the bottom of lists then starting again from the top of the lists in a circular manner.
- ❑ The maximum length I/O request will be scheduled to the server with lighter load and the minimum length I/O request will be distributed to the server with heavier load.

Proposed Algorithm: Max Length - Min Load MLML Cnt.

- Two sorted lists: ***sortedServers*** and ***sortedRequests***
- **Default_oss** describes the default location of requested objects (default RR strategy).
- **Target_oss** represents the storage server calculated based on this MLML scheduling algorithm.
- If the target_oss and the default_oss are not the same server, we then need to consider whether it is worth of scheduling this request to the target oss as this might increase read overhead due to the redirection.

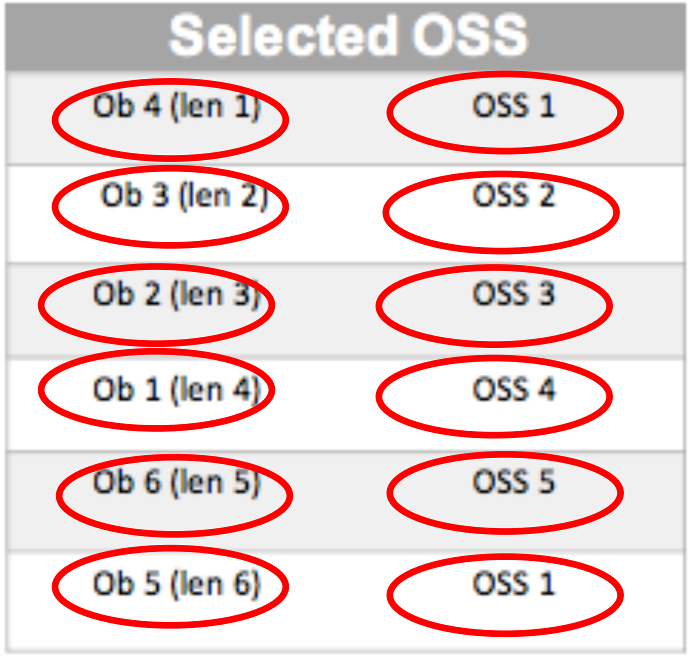
Max Length - Min Load



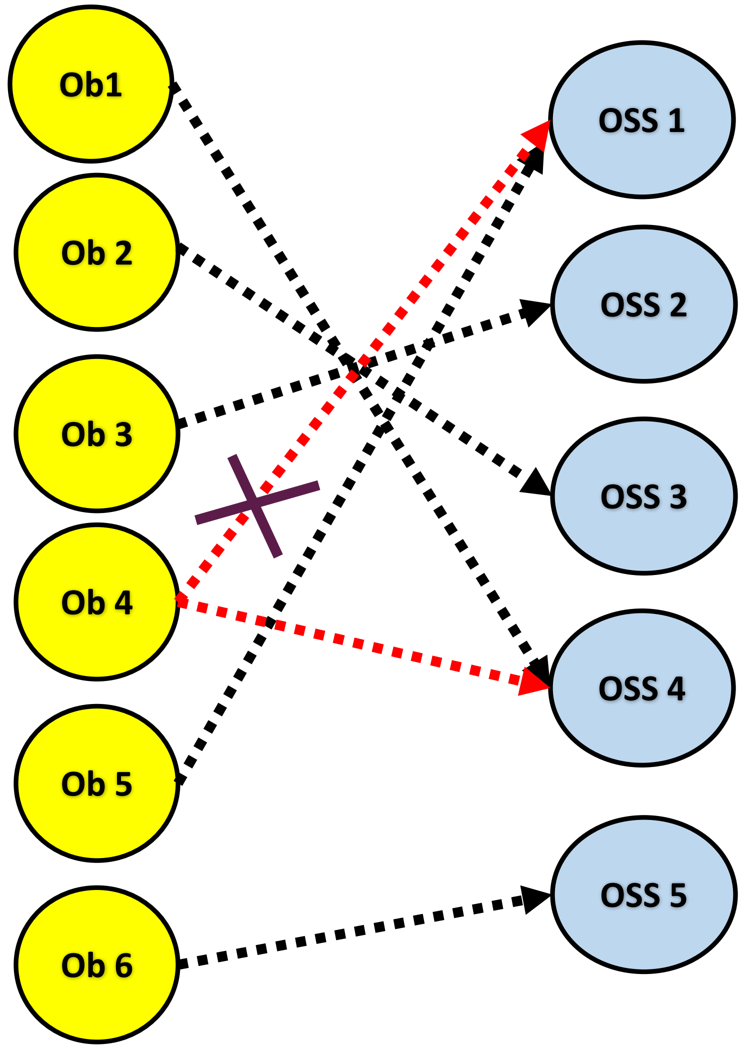
Algorithm 1 Max Length-Min Load algorithm (MLML)

```
procedure MLML(SERVERS,REQUESTS,THRESHOLD)
    sortedServers = sort(servers);
    sortedRequests = sort(requests);
    index=0;
    while need_schedule() do
        default_oss = requests[index]   mod   M;
        target_oss = sortedRequests[index]   mod   M;
        benefit = load(default_oss) - load(target_oss);
        if benefit $\leq$ threshold then
            return target_oss;
        else
            return default_oss;
        index++;
```

Example of MLML algorithm



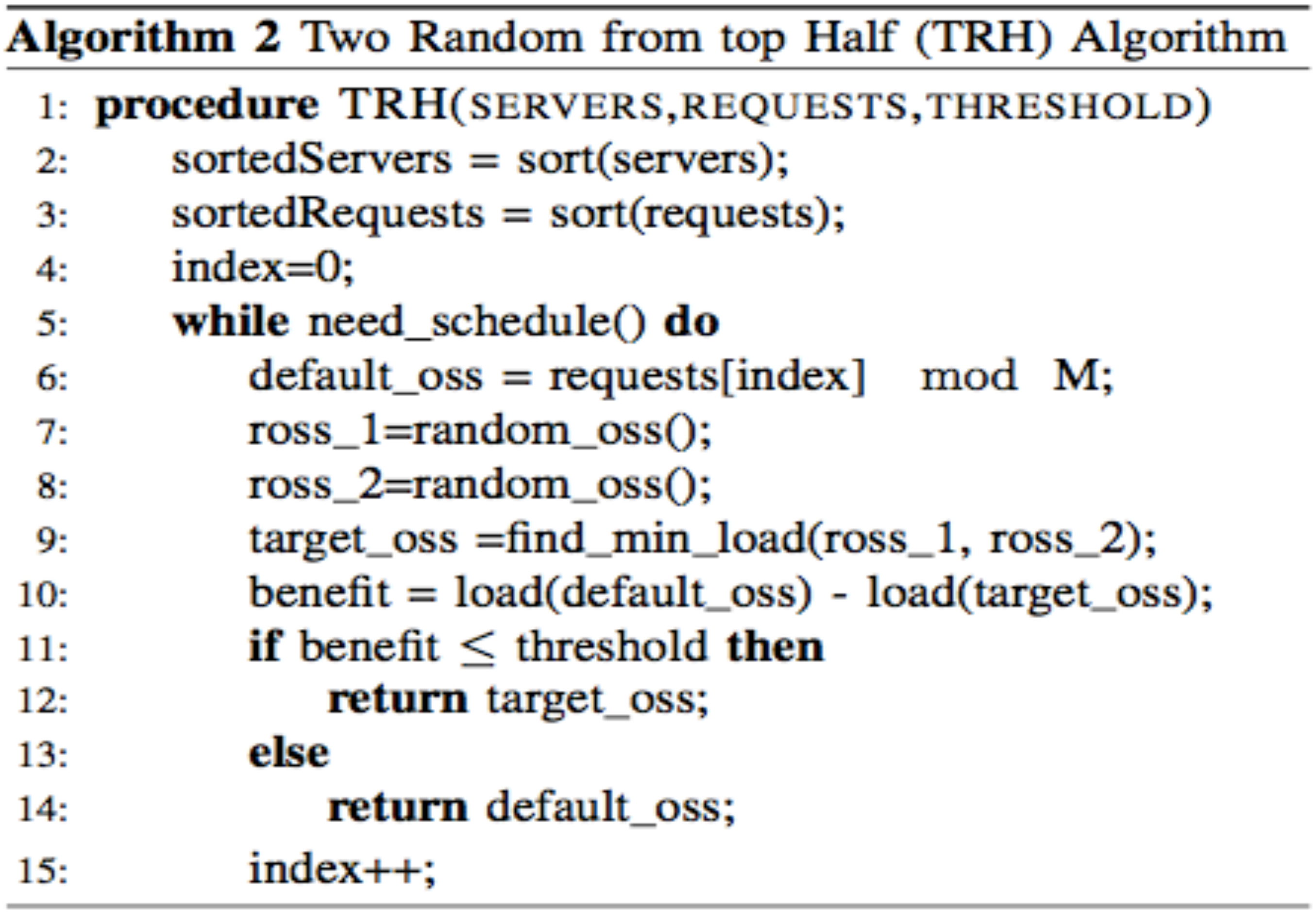
Selected OSS	
Ob 4 (len 1)	OSS 1
Ob 3 (len 2)	OSS 2
Ob 2 (len 3)	OSS 3
Ob 1 (len 4)	OSS 4
Ob 6 (len 5)	OSS 5
Ob 5 (len 6)	OSS 1



Proposed Algorithm: Two Random from Top Half (TRH)

- Two OSSs are randomly selected as potential targets and the one with the lighter load will be chosen.
- The sorted list of object storage servers is also used, same as in the MLML algorithm.
- For a given I/O request, two object storage servers will be randomly chosen from the top half of all object storage servers.
- The top half indicates M servers that have lighter loads.
- A threshold is used to decide whether choosing the calculated **target_oss** or the **default_oss**.

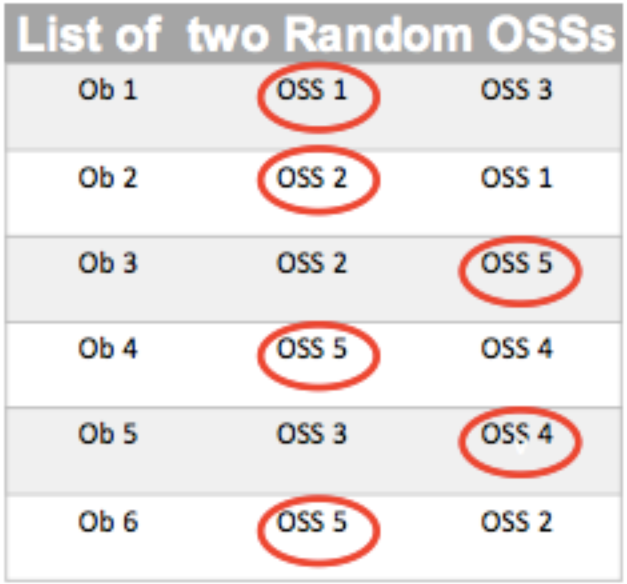
Proposed Algorithm: Two Random from Top Half (TRH)



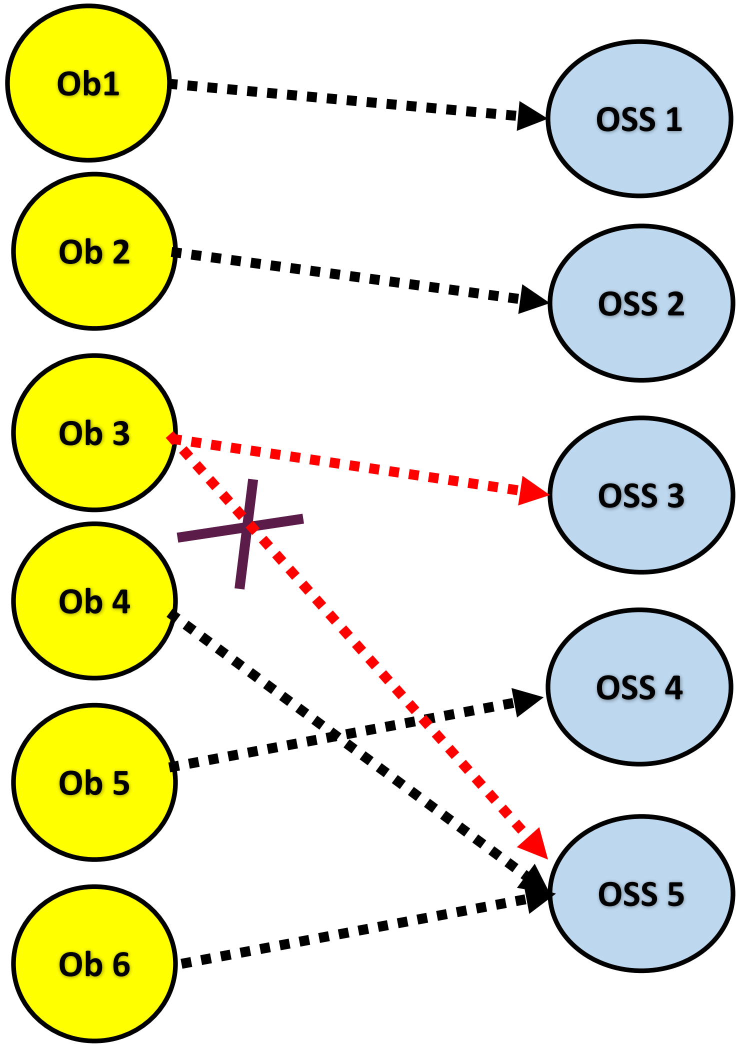
Algorithm 2 Two Random from top Half (TRH) Algorithm

```
procedure TRH(SERVERS,REQUESTS,THRESHOLD)
    sortedServers = sort(servers);
    sortedRequests = sort(requests);
    index=0;
    while need_schedule() do
        default_oss = requests[index]   mod   M;
        ross_1=random_oss();
        ross_2=random_oss();
        target_oss =find_min_load(ross_1, ross_2);
        benefit = load(default_oss) - load(target_oss);
        if benefit $\leq$ threshold then
            return target_oss;
        else
            return default_oss;
        index++;
```

Example of TRH algorithm



List of two Random OSSs		
Ob 1	[✓ OSS 1]	OSS 3
Ob 2	[✓ OSS 2]	OSS 1
Ob 3	OSS 2	[✓ OSS 5]
Ob 4	[✓ OSS 5]	OSS 4
Ob 5	OSS 3	[✓ OSS 4]
Ob 6	[✓ OSS 5]	OSS 2



Evaluations

Evaluation Platform

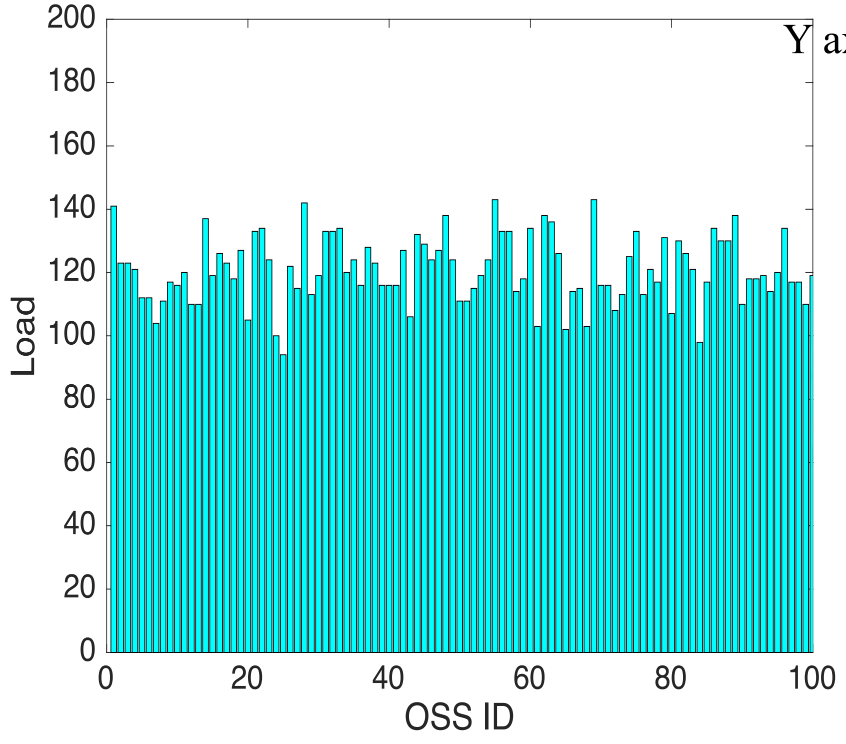
- We use simulation to evaluate the proposed scheduling algorithms.
- Synthetic workloads generated by simulating real-world applications.
  - Combining three different types of I/O requests:
    - large I/O (each request is greater than O(10MB))
    - medium I/O (each request is between 4MB and 10MB)
    - Small I/O (each request is less than 4MB)
  - The simulation is run in an HEC cluster with 100 object storage servers and 200 compute nodes.

Evaluation Platform

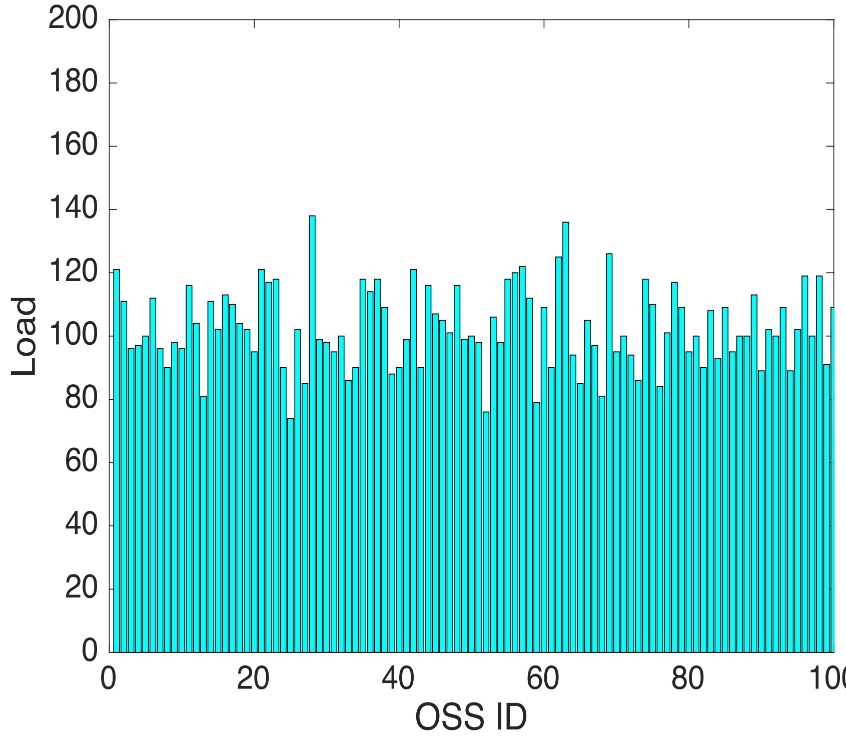
- We issued 2,000 I/O requests in each run with various sizes for each request.
- The total data written for all large I/O case can be between O(20GB) and O(2TB) depending on the size of each request.

Evaluation Results

X-axis: Storage servers
Y-axis: Load of each storage server (i.e, write data size in MB).



Load distribution with the RR scheduling algorithm

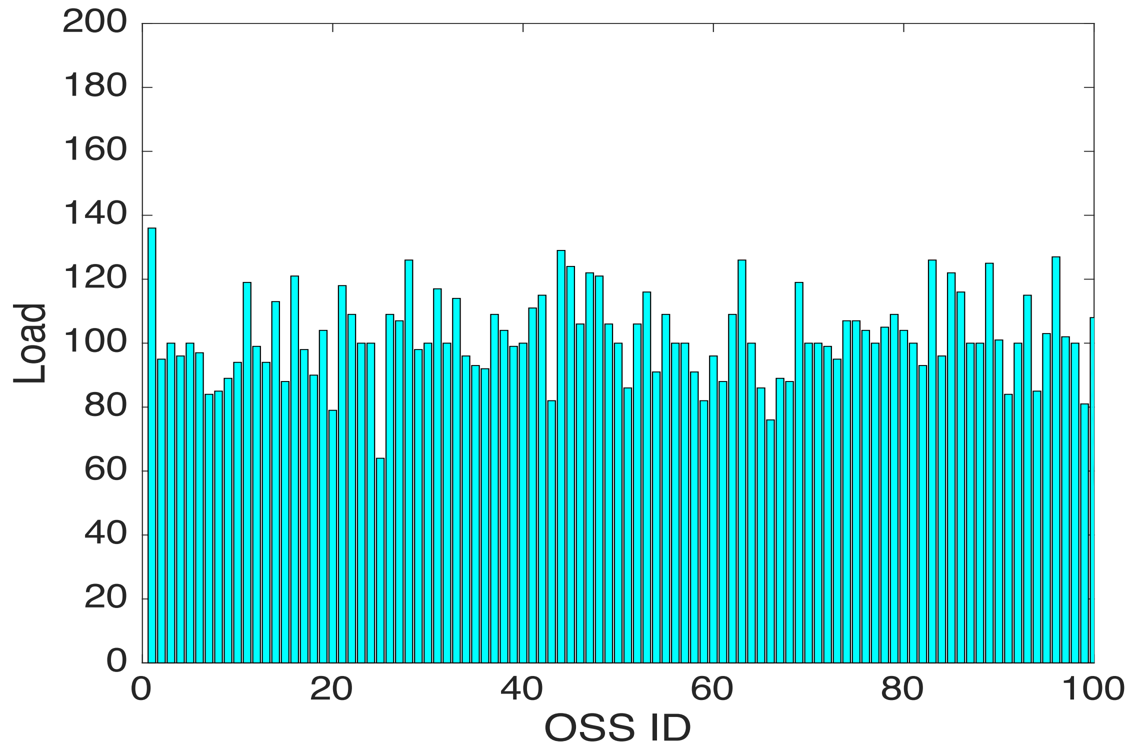


Load distribution with the log-assisted straggler-aware scheduling with the MLML algorithm

TEXAS TECH UNIVERSITY.

Evaluation Results

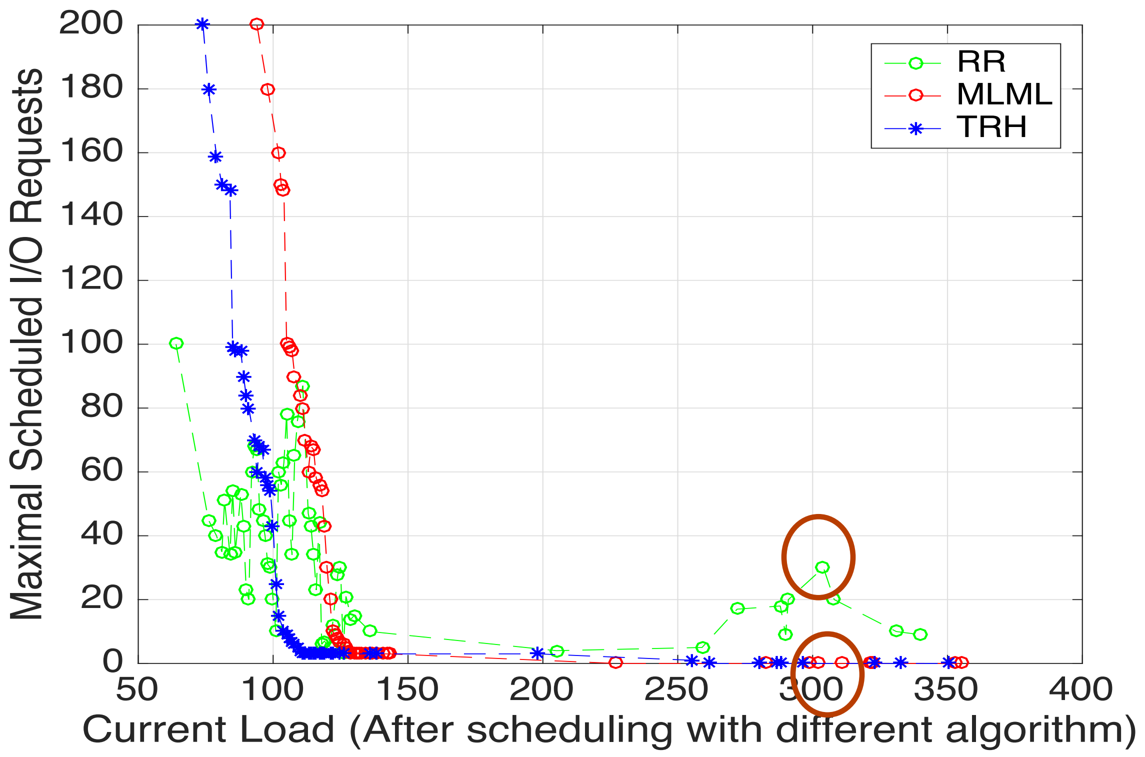
X-axis: Storage servers
Y-axis: Load of each storage server (i.e, write data size in MB).



Load distribution with the log-assisted straggler-aware scheduling with the TRH algorithm.

Evaluation Results

X-axis: Current Load of Storage servers (i.e, write data size in MB).
Y-axis: Maximum number of scheduled I/O requests.



Comparison of RR, MLML, and TRH scheduling algorithms.

Conclusion

- ❑ A new log-assisted straggler- aware I/O scheduler is introduced.
- ❑ A client-side server statistic log is introduced.
- ❑ Two straggler-aware scheduling algorithms based on the proposed client-side server statistic log:
  - ❑ Max Length - Min Load (MLML) algorithm
  - ❑ Two Random from Top Half (TRH) algorithm.
- ❑ The evaluations confirm that the log-assisted straggler-aware scheduling achieves a better load balance on storage servers while avoiding stragglers.

Questions