Coordination of Data Movement with Computation Scheduling on a Cluster

Alex Romosan, Doron Rotem,

Arie Shoshani and John Bent*

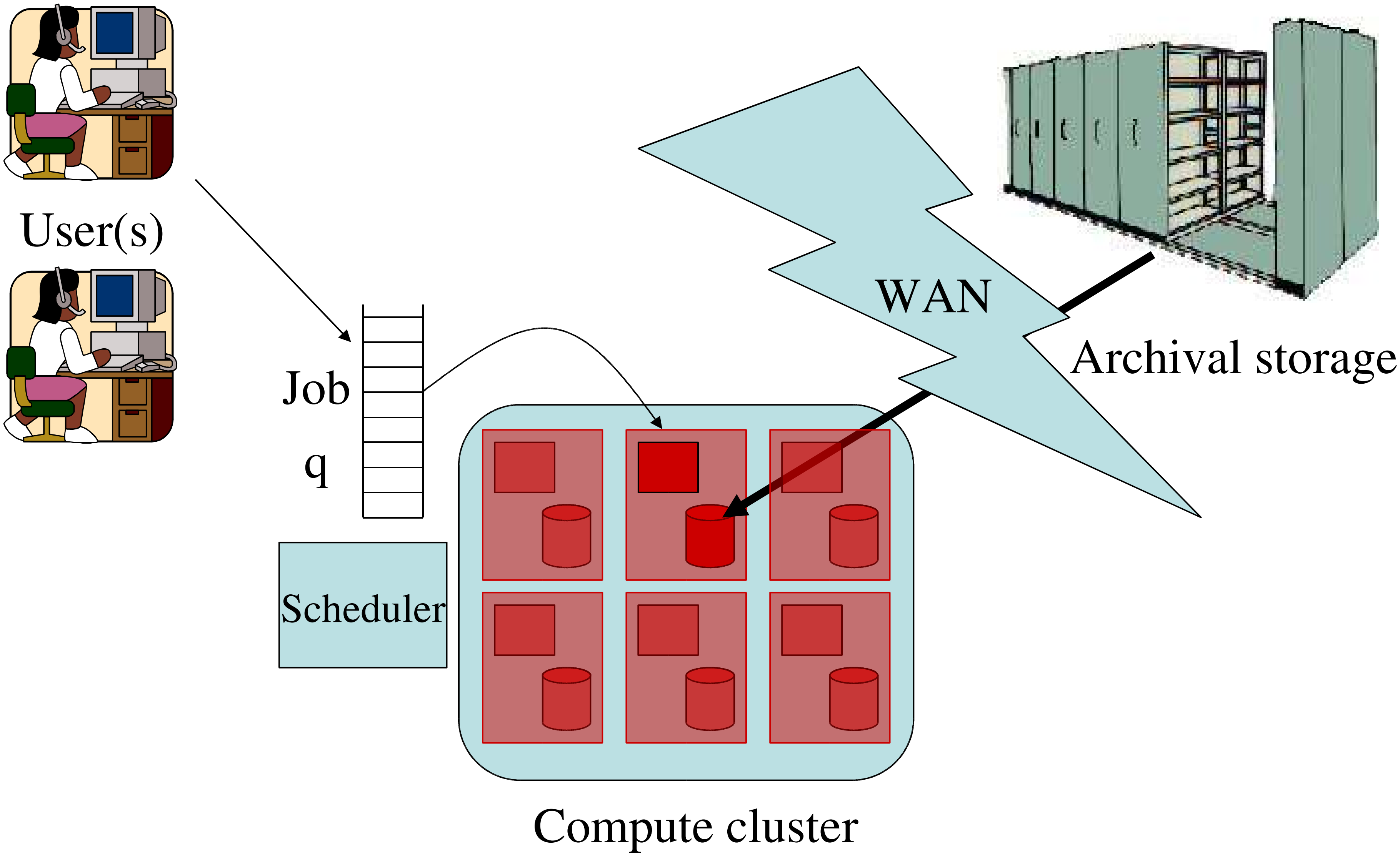
Lawrence Berkeley National Laboratory

*University of Wisconsin

Outline

- Problem Description
- Architecture
- Scheduling Strategies
  - FIFO
  - Shortest Job First
  - Linear Programming
- Network Flow Representation
- Simulation Environment
- Results

Problem Domain



Problem Description

- **Schedule a collection of jobs, each requiring one or more input files, to run on a group of servers, each server having one or more compute slots and a disk cache that can hold some fraction of the data**
- **A job can be scheduled on a server if:**
  - the server has at least one available compute slot
  - all data files needed are available on the server's disk cache
- **Need to coordinate data movement with**

Goal

- **Automatically match each job to the machine that has the file needed for the job**
  - ability to schedule jobs and fetch files from tape
  - need information on the content in each disk cache
- **Optimize parallel analysis on the cluster**
  - Minimize movement of files between cluster nodes
  - Use node cluster as evenly as possible
  - Automatic replication of hot files
  - Automatic management of disk space
  - Automatic removal of cold files
    (Automatic garbage collection)

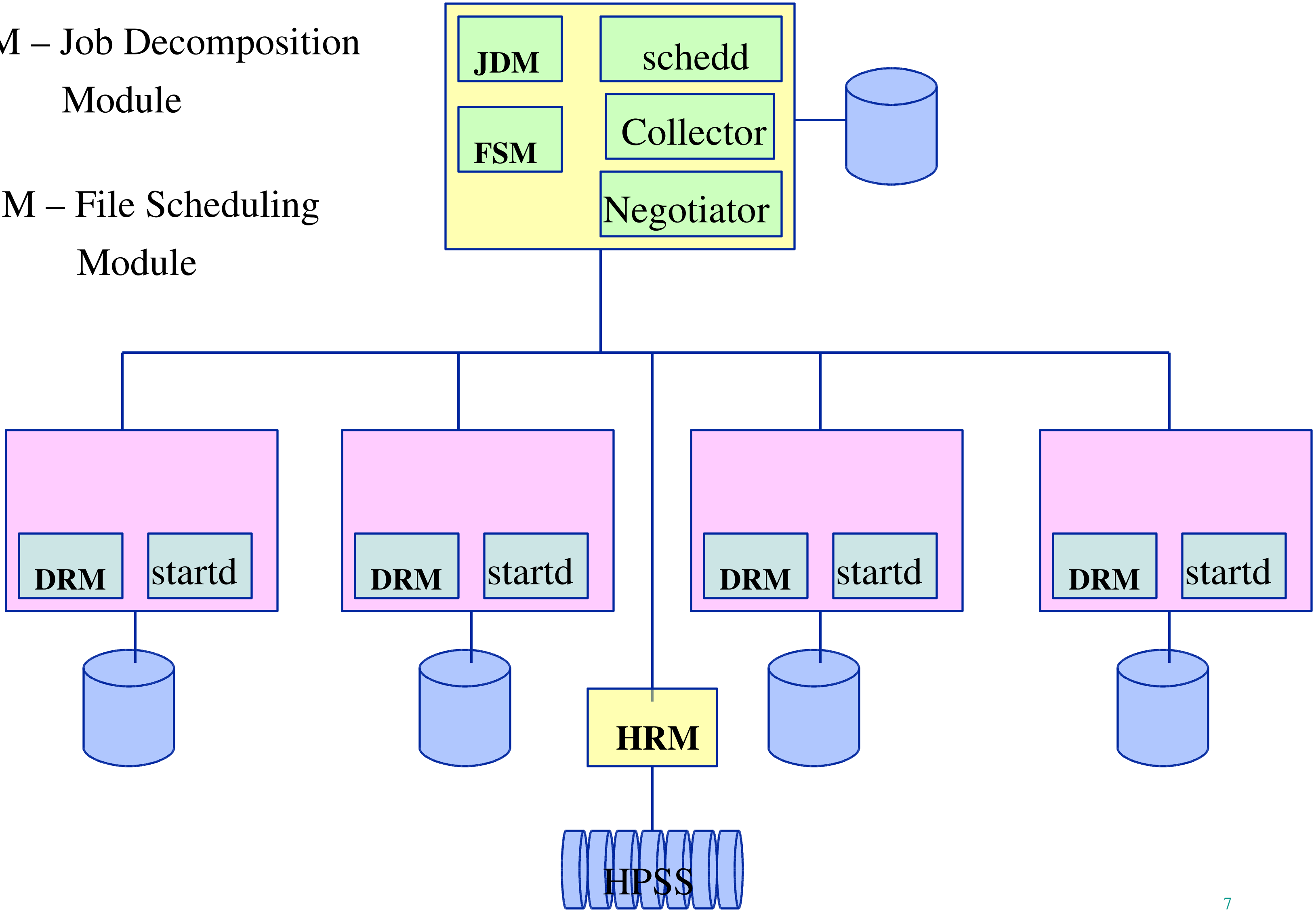
Solution

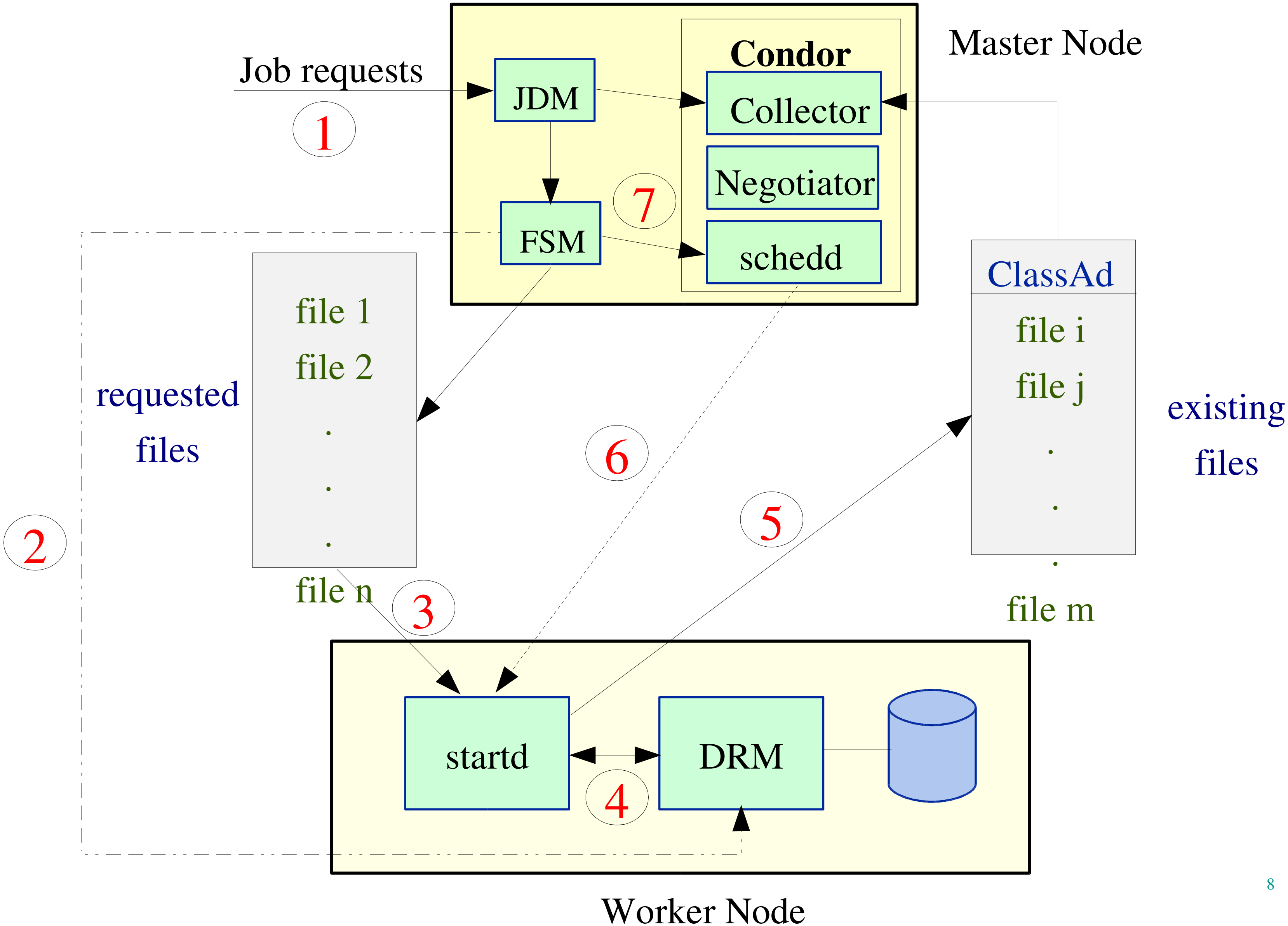
- **Use existing software components:**
  - Condor for job scheduling
  - Condor for matchmaking of slots and files
    - open-ended description of what to match on (classAd)
  - DRMs for disk management
    - dynamic storage allocation
    - ability to “pin” and “release” files
  - HRMs for fetching files from tape
- **Developed “glue” component to achieve co-scheduling**

Architecture

JDM – Job Decomposition Module

FSM – File Scheduling Module




Master Node
Job requests
1
JDM
Condor
Collector
Negotiator
7
FSM
schedd
ClassAd
file i
file j
.
.
.
file m
existing
files
requested
files
file 1
file 2
.
.
.
file n
2
6
5
3
startd
DRM
4
Worker Node

Scheduling Algorithms

- **FIFO**
  - Grab job at head of queue in FIFO order
  - Simplest and fairest
- **Shortest Job First**
  - Estimate time to completion for each job
  - Schedule shortest job first
  - Overhead *O*(*#jobs* x *#servers*)
  - Exit “early” if shortest job found
- **Linear Programming**
  - Represent problem as network flow and solve a

FIFO

- **Schedule jobs as they arrive**
  - no scheduling overhead
- **Choose next server in round-robin fashion**
  - many unneeded local and remote replications
  - server underutilization
  - low throughput

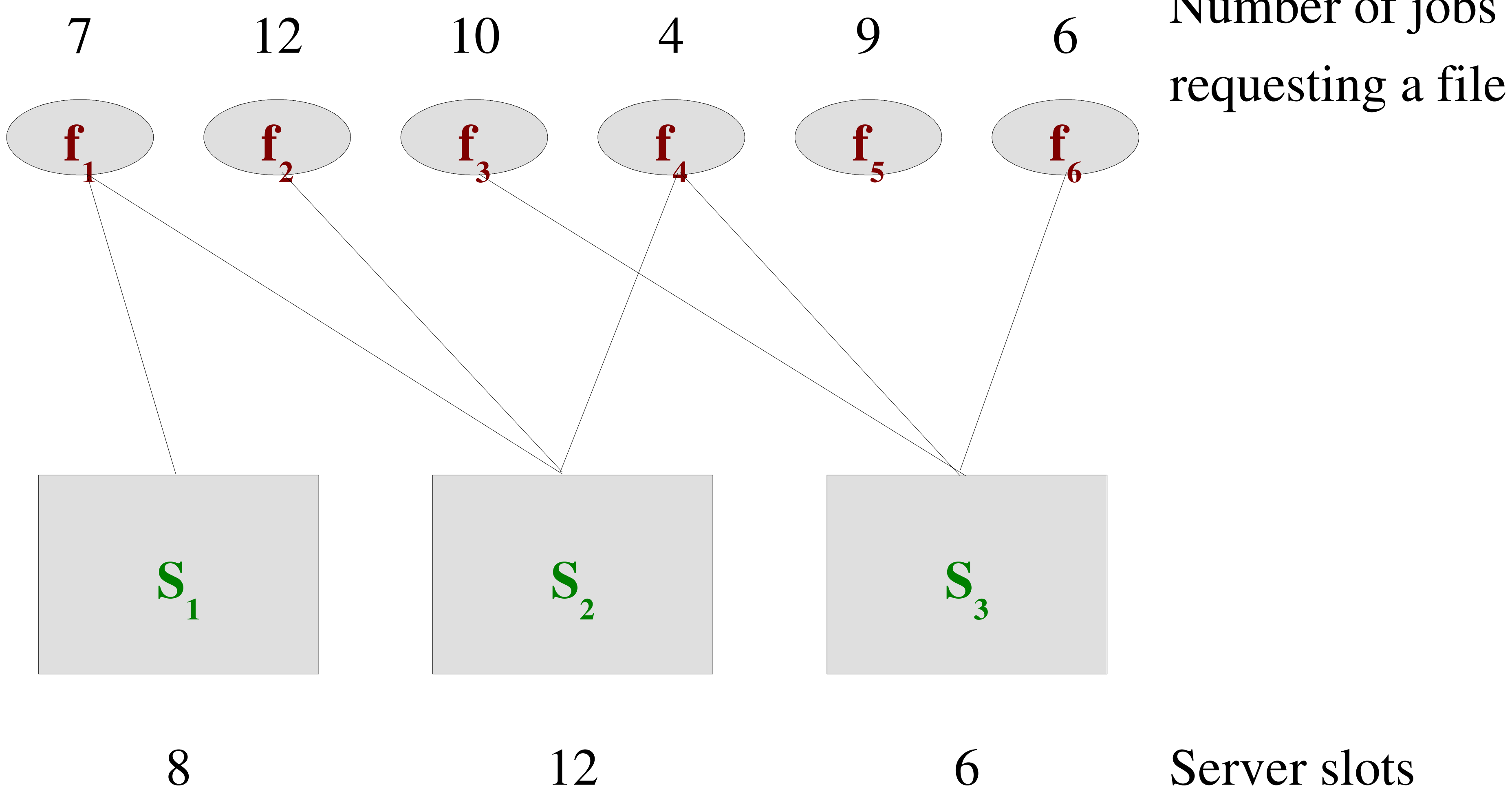
Shortest Job First Algorithm

- **Optimally minimizes average waiting time**
  - possible starvation of long jobs
- **Use data movement as first-order approximation of job runtime.**
- **Compute data cost incurred if job were to be scheduled on each server:**
  - 0, if file already on server
  - File size weighted by either local or remote cost
- **Schedule job that requires cheapest amount of data movement**

Scheduling Using LP models

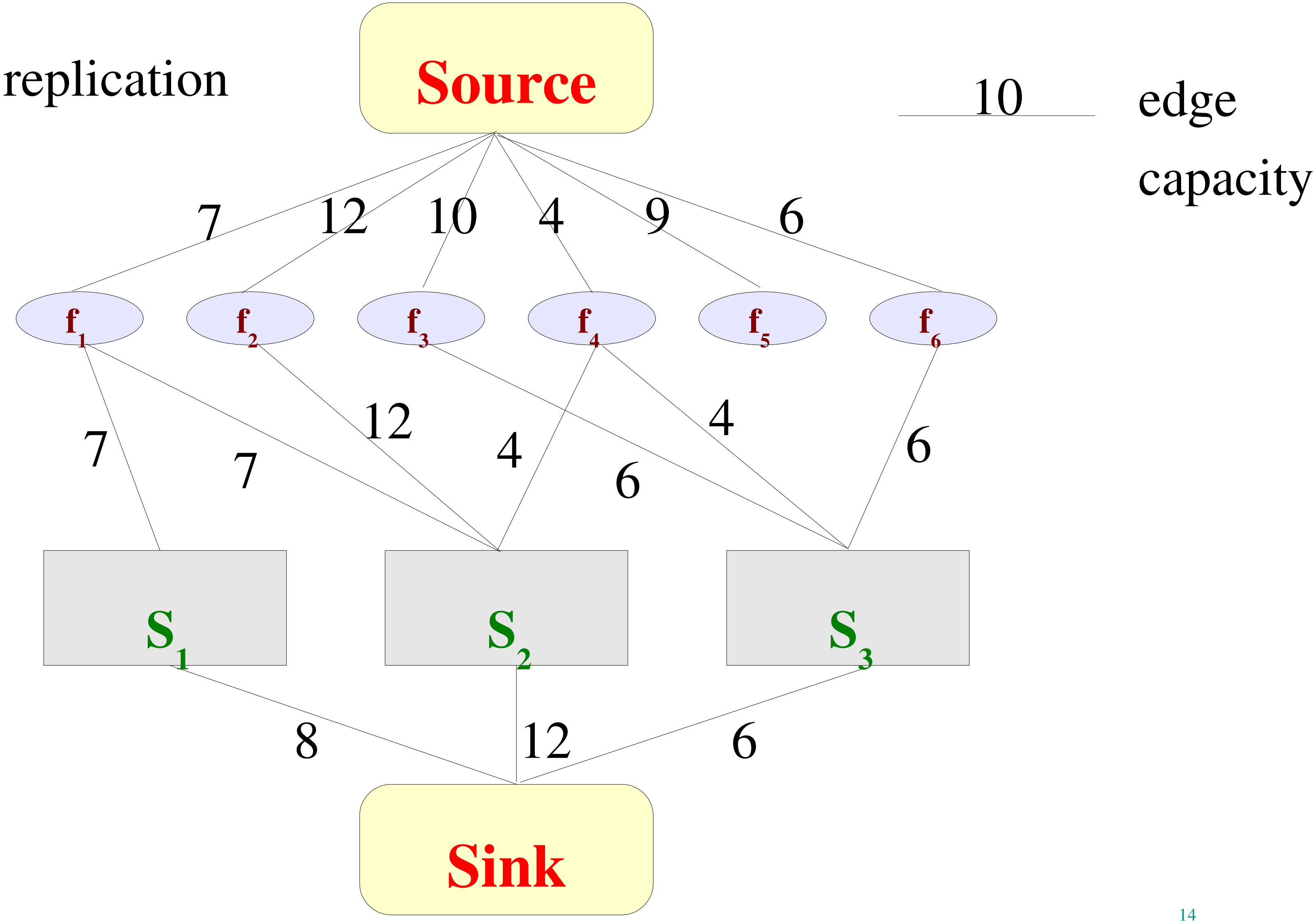
- **Create node weighted bipartite graph *B(F,S,E)***
  - *F* – files requested by the queued jobs
  - *S* – servers in the cluster
  - *E* – edges $e(f_i, s_j)$
- **Define costs and constraints for edges**
- **Articulate an objective function**
- **Find an LP library to do the heavy lifting**

Bipartite Graph Representation

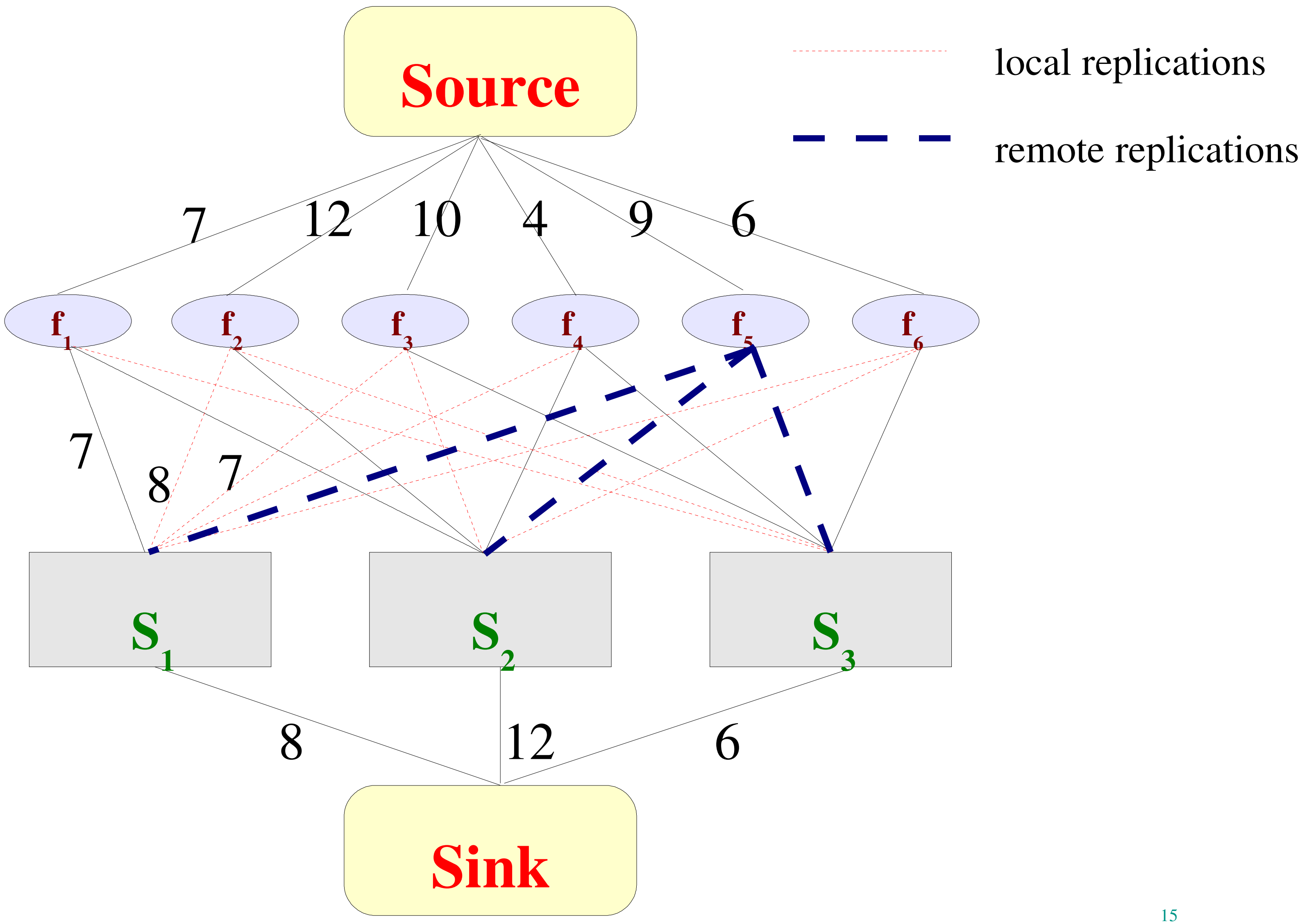


Network Flow Representation

without replication



Local and Remote Replications



Mathematical Programming Formulation

- **Let *l* denote local and *r* remote replication costs**
- **For an edge (f_i, s_j), the cost $C(f_i, s_j)$ of connecting file f_i to server s_j is represented by:**

$$C(f_i, s_j) = \begin{cases} l, & f_i \text{ does not reside on } s_j \text{, but a copy resides on some server} \\ r, & f_i \text{ does not reside on any server} \end{cases}$$

- **Minimize**

$$\sum_{i,j} x(f_i, s_j) C(f_i, s_j)$$

$$x(f_i, s_j) = \begin{cases} 1, & \text{if } flow(f_i, s_j) > 0 \\ 0, & \text{otherwise} \end{cases}$$

Mathematical Programming Formulation (cont'd)

- **Flow on an edge cannot exceed its capacity**

$$flow(f_i, s_j) \leqslant capacity(f_i, s_j) \; \forall i, j$$

- **Flow into a node equals flow out of it**

$$flow(source, f_i) = \sum_j flow(f_i, s_j) \; \forall i$$

- **Flow from each server to a sink equals total flow into that server**

$$flow(s_j, sink) = \sum_i flow(f_i, s_j) \; \forall j$$

- **Require the maximum possible flow**

$$\sum_j flow(s_j, sink) = \min\left(\sum_i N(f_i), \sum_j S(s_j)\right)$$

An Approximation

- Previous formulation is known to **NP-hard** by reduction from set cover
- Replace $x(f_i, s_j)$ with the ratio of actual flow on the edge to its total capacity
- New objective function:

$$minimize\left(\sum_{i,j}\frac{flow(f_i,s_j)}{Capacity(f_i,s_j)}C(f_i,s_j)\right)$$

- This is a **linear program** solvable in polynomial

LP Algorithm Implementation

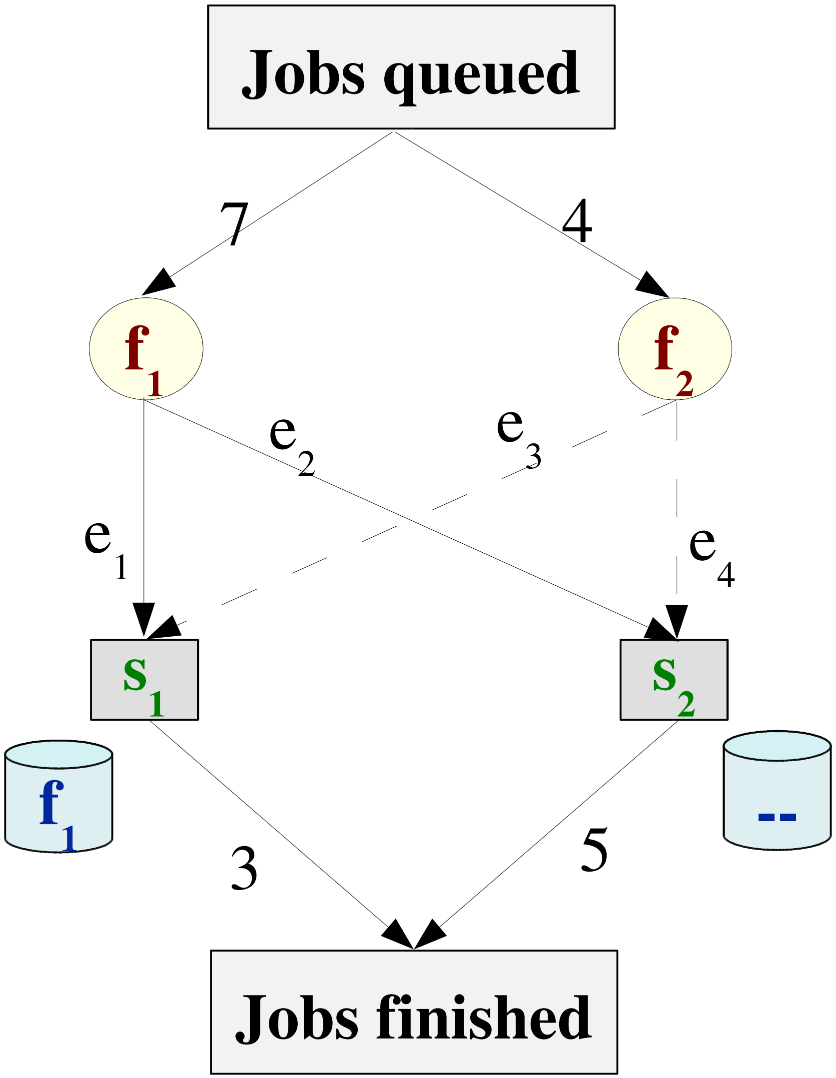
- Create a fully connected bipartite graph with an edge from each file to each server
  - ignore unpopular files and unavailable servers
- Constrain each edge to be min of file popularity and server capacity
- Constrain all edges exiting a file to not exceed its popularity
- Constrain all edges entering a server to not exceed its capacity

LP Algorithm Implementation (cont'd)

- **Compute maximum throughput to be min of jobs in queue and available server slots**
  - set constraint that flow equals max throughput
- **Define cost of using each edge:**
  - *C*(e) = X / max_edge_capacity
  - where X is
    - 0 if that edge exists
    - LOCAL_COST if another server has a copy
    - REMOTE_COST if file is not cached locally

- **Set objective function to minimize cost**

Example



- **Create fully connected bipartite graph with an edge from each file to each server**
- **Constrain each edge to be min of file popularity and server capacity**

$$0 \leqslant e_1 \leqslant \min(3,7)=3$$
$$0 \leqslant e_2 \leqslant \min(5,7)=5$$
$$0 \leqslant e_3 \leqslant \min(3,4)=3$$

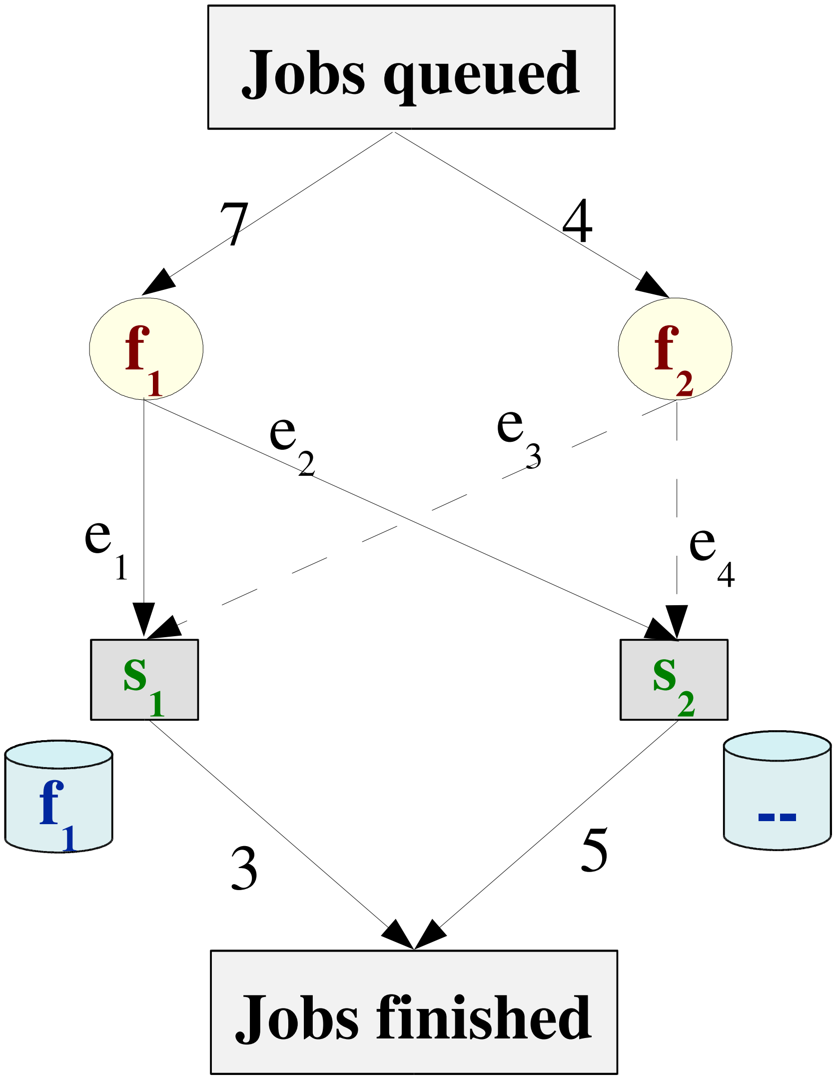
$$e_1+e_2 \leqslant 7$$

- **Constrain all edges exiting a file to not exceed its popularity**

$$e_1+e_3 \leqslant 3$$

- **Constrain all edges entering a**

Example (cont'd)



- **Compute max throughput to be min of jobs in queue and available sever slots**

$$e_1 + e_2 + e_3 + e_4 = 8$$

- **Define cost of using each edge**

$$C(e_1) = 0$$
$$C(e_2) = 1$$
$$C(e_3) = 10$$

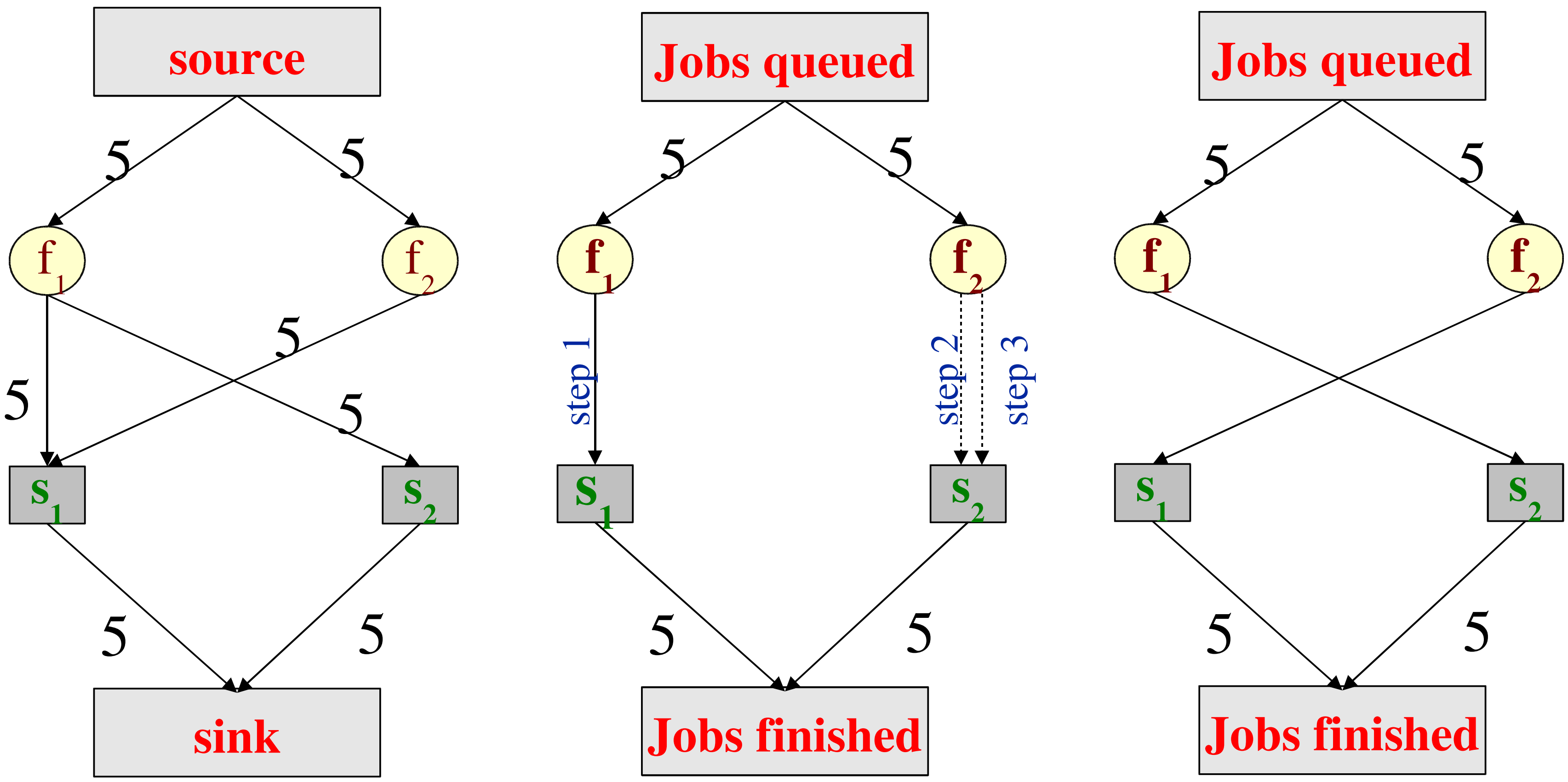
$$\text{minimize}(0 \times e_1 + 1 \times e_2 + 10 \times e_3 + 10 \times e_4)$$

- **Set objective to minimize cost**

$$e_1 = 3$$
$$e_2 = 4$$
$$e_3 = 0$$

- **Let LP do the heavy lifting**

SJF and LP Comparison



Network diagram **SJF** **LP**

Finding an LP Solver

- **Researched different libraries**
  - http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html#Q2
  - http://www.cs.sunysb.edu/~algorith/files/linear-programming.shtml
- **Selected lp_solve**
  - Written in ANSI C
  - Ported to *nix
  - Solve up to 30K variables, 50K constraints
  - FREE for non-commercial use
  - Generally considered best free code available

Simulation Environment

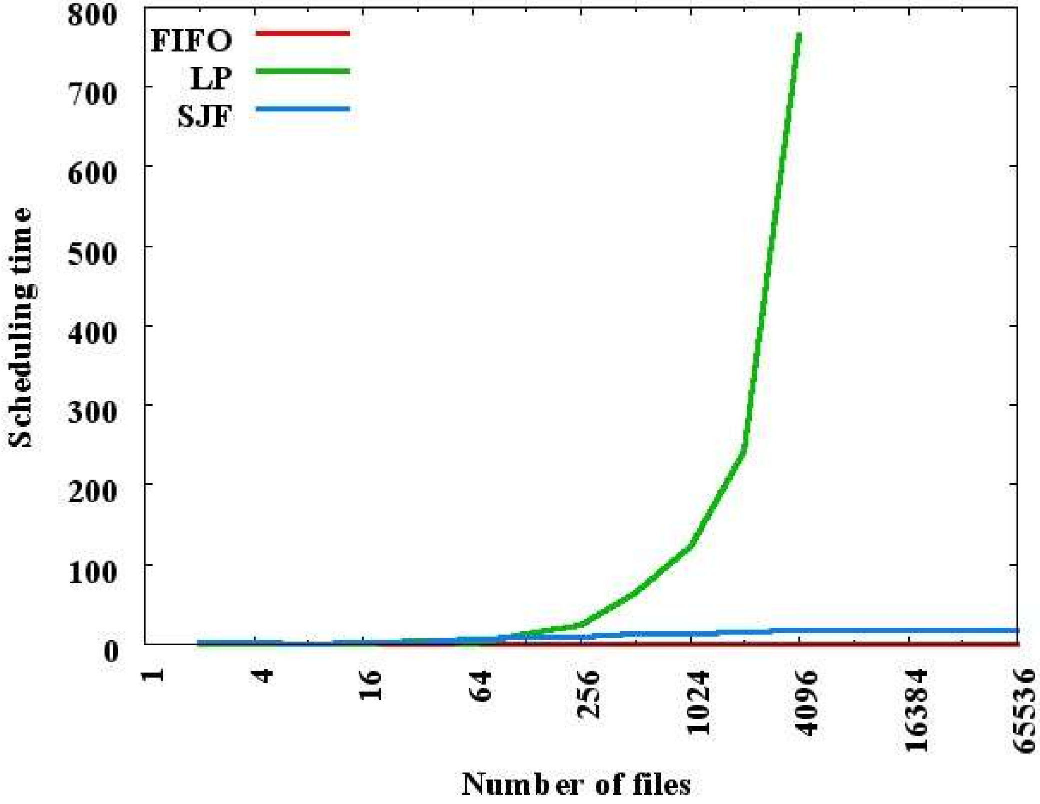
- **8 single and 1 dual CPU 1.5GHz Athlons**
  - 20GB disk cache and 2GB of RAM per node
- ***Sched_sim*** **written in C++ with extensive use of STL**
  - geared towards simulation of shared-nothing clusters
- **Used the Condor batch scheduling system**
  - ran 6643 simulations
  - consumed 27.92 CPU days
- **Two types of measurements**
  - measurements of simulated systems

Simulation Parameters

- **Server configuration**
  - Capacity for jobs and data
  - Cache policy
- **Dataset**
  - Size of complete dataset and of each file
  - Characteristics of popularity distribution
- **Network**
  - Bandwidth to archive server and within local network
- **Jobs**

Scheduling Sensitivity

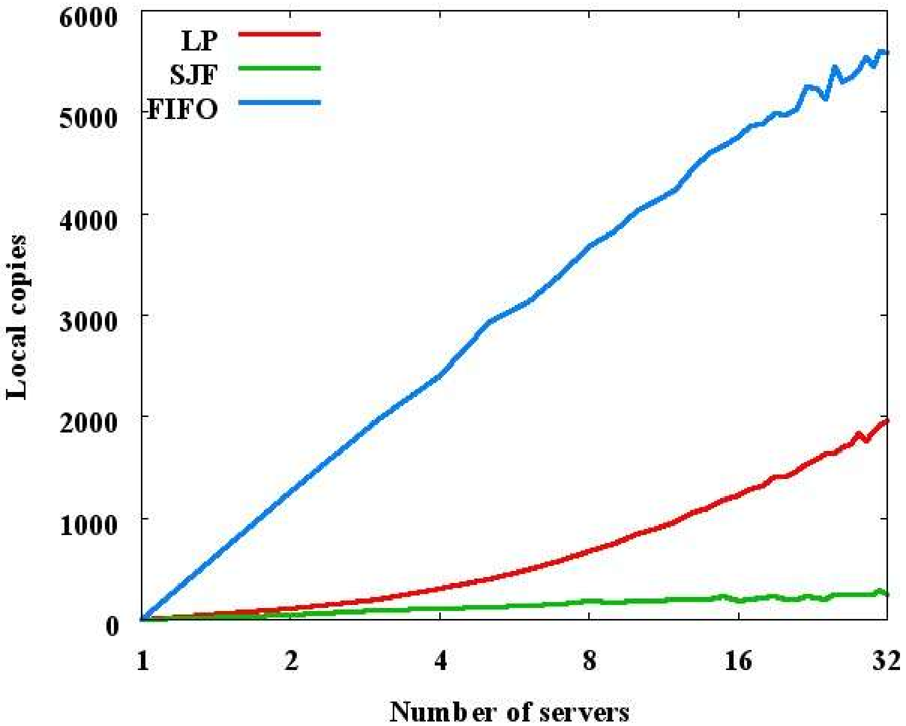
- 10,000 jobs, 2-64K files, 32 servers

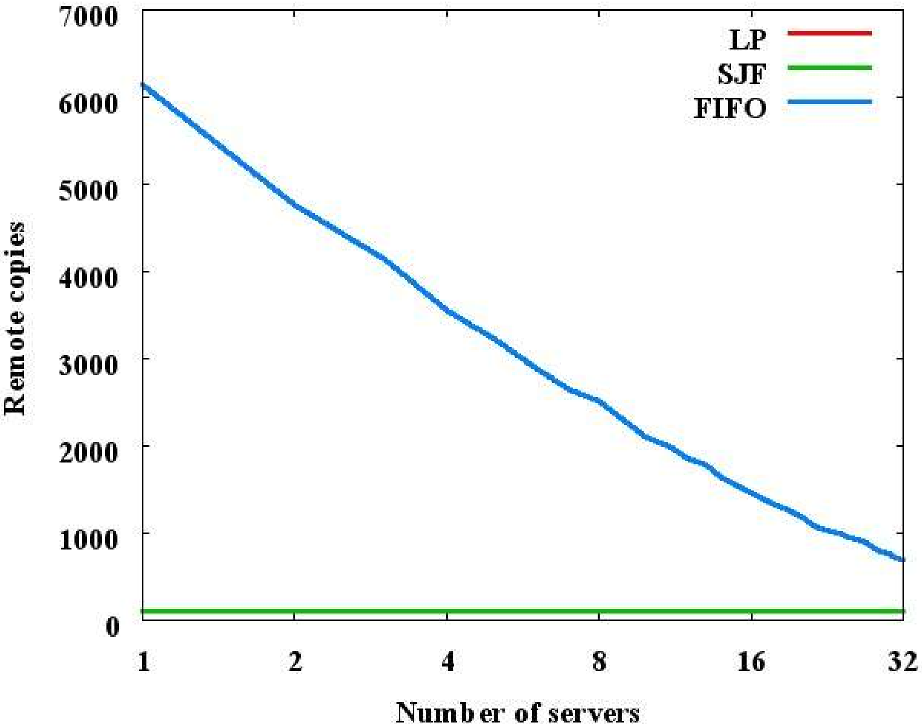


- LP is very sensitive to the number of edges
- Unable to run when edges = 8K*32

Number of File Replications

- 10,000 jobs, 100 files, 1-32 servers

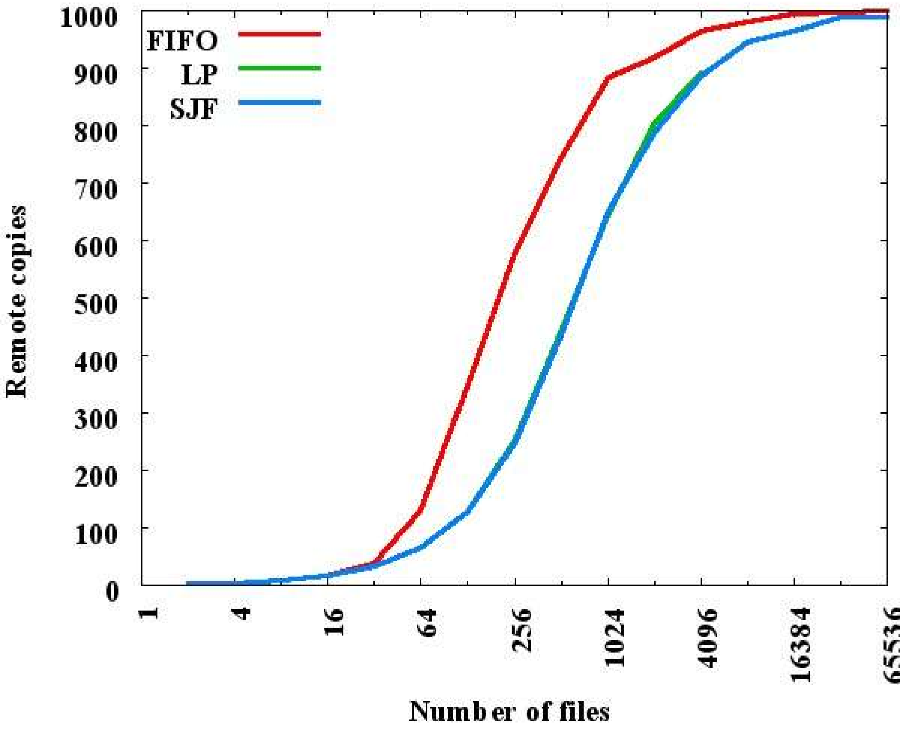




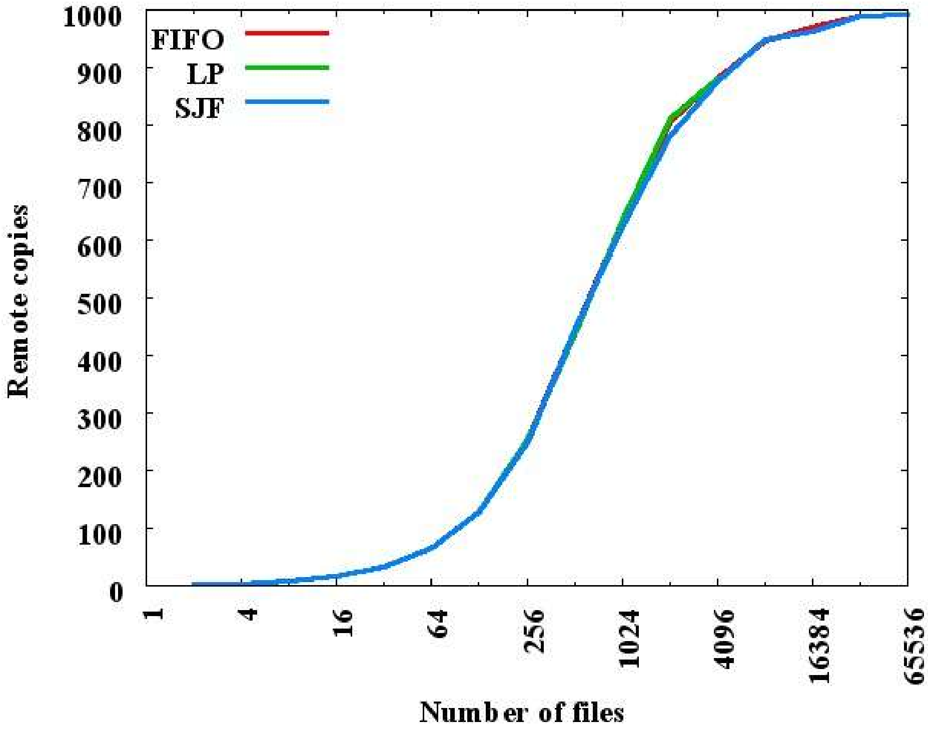
- SJF makes fewer local copies than LP
- FIFO very wasteful

Popularity Distribution

- **Zipf's popularity distribution, 1000 jobs**



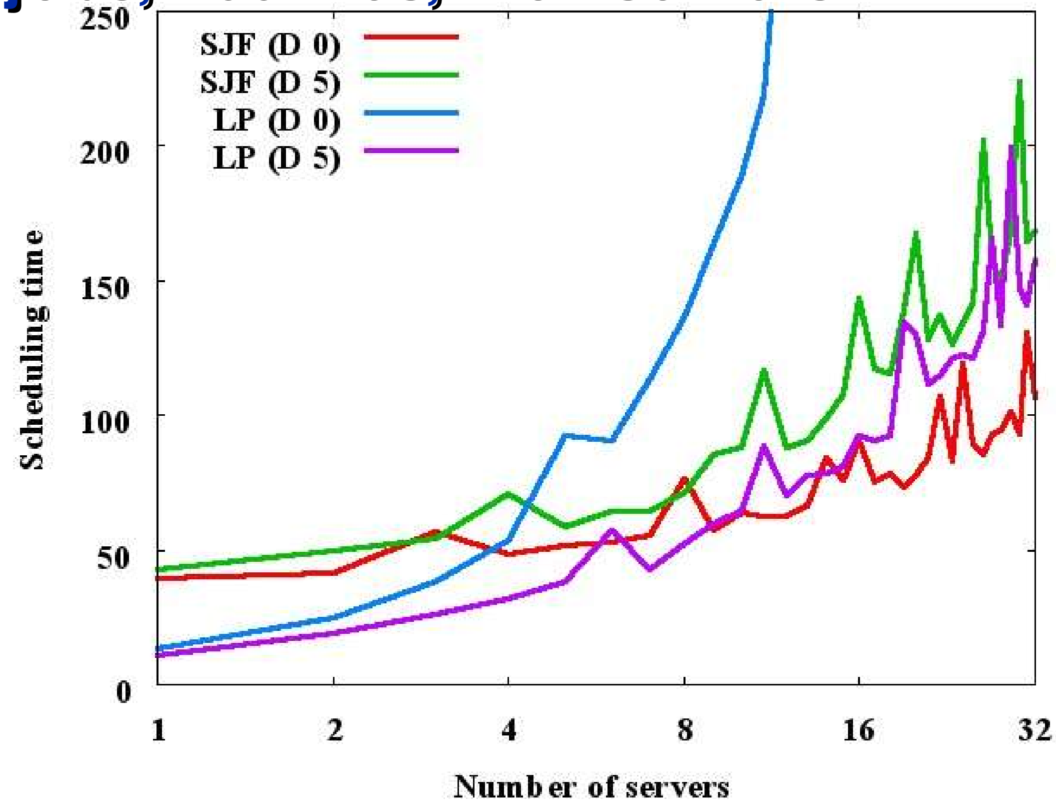
Small cache



Infinite Cache

Scheduling Overhead

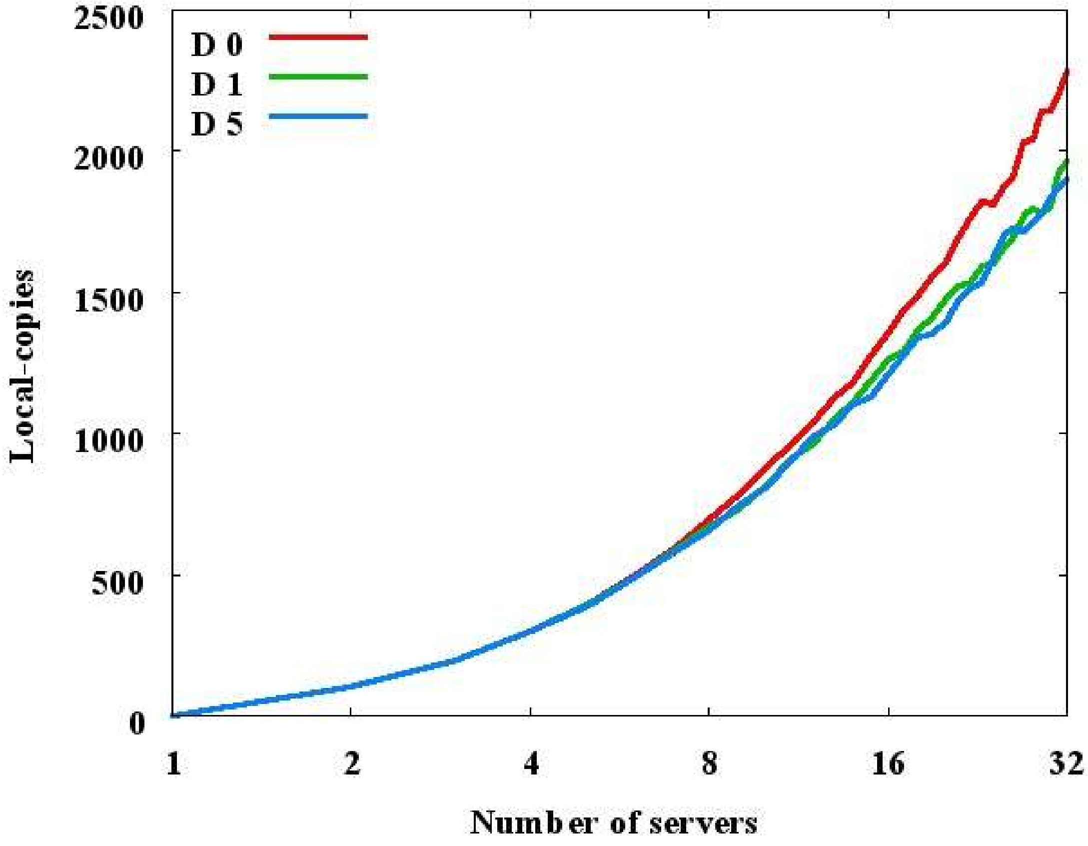
- **Delay scheduling to minimize calls to LP**
- **10,000 jobs, 100 files, 1-32 servers**



- **Delays reduce scheduling overhead**

LP Local copies with delays

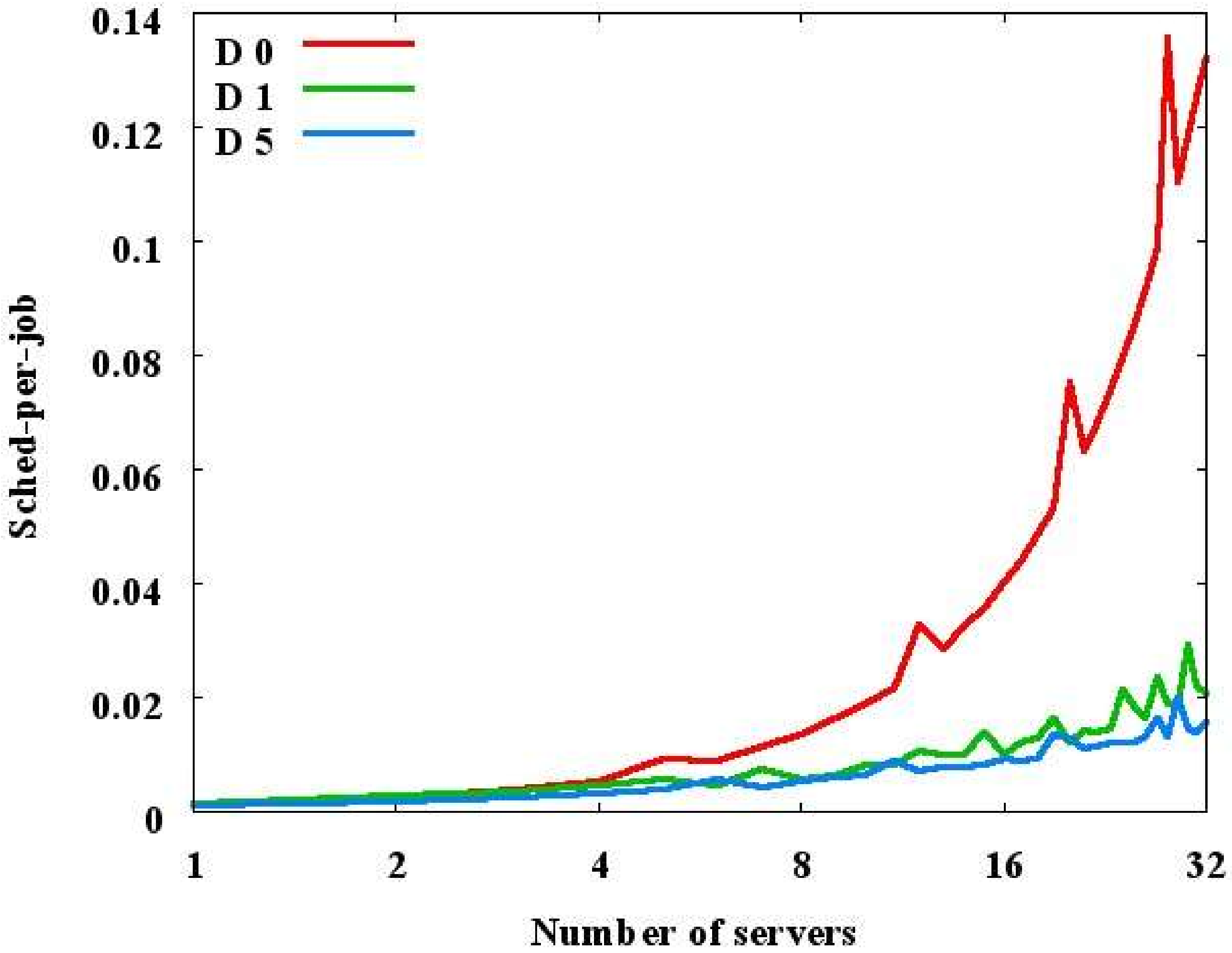
- **10,000 jobs, 100 files, 1-32 servers**



Delay to LP results in fewer local copies.

LP Scheduling time with delays

- **10,000 jobs, 100 files, 1-32 servers**



Conclusions

- **FIFO performs the most local and remote copies**
- **FIFO has the lowest server utilization**
  - the three methods converge with increasing number of servers
- **SJF and LP are equivalent in the number of replications performed and run times**
- **Longer delays between successive LP's significantly reduce scheduling time**
- **Increasing the number of servers causes**
  - fewer remote copies and more local copies

Future Work

- **More simulations needed to study the effects of the different variables**
- **Use real workflows**
- **Study workflows with dependencies between inputs and outputs of successive jobs**
- **Test additional algorithms**
  - prefetch files to compute nodes
- **Take into account "remote transfer in progress" events**

- **Use scheduling algorithms to drive real system**