Communication-aware Job Scheduling using SLURM

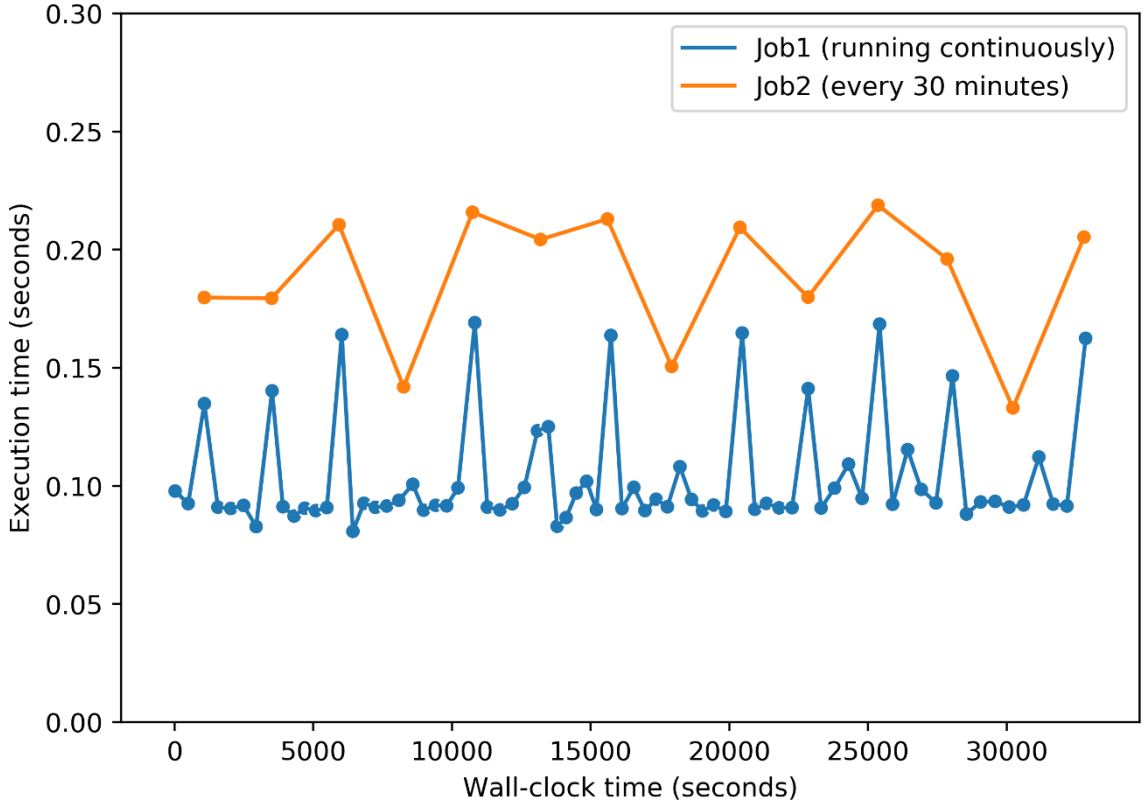
Priya Mishra, Tushar Agrawal, Preeti Malakar
Indian Institute of Technology Kanpur

16th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems

Introduction

- Job Scheduling deals with cluster management and resource allocation as per job requirements
- Users submit jobs specifying nodes and wall-clock time required
- Current job schedulers do not consider job-specific characteristics or communication-patterns of a job
  - May lead to interference from other communication-intensive jobs
  - Placing frequently communicating node-pairs several hops away leads to high communication times

Effect of network contention



- J1 and J2 are two parallel MPI[1] jobs
- J1 executed repeatedly on 8 nodes (4 nodes on 2 switches)
- J2 executed every 30 minutes on 12 nodes spread across same two switches
- Sharp increase in execution time of J1 when J2 is executed
- Sharing switches/links degrades performance

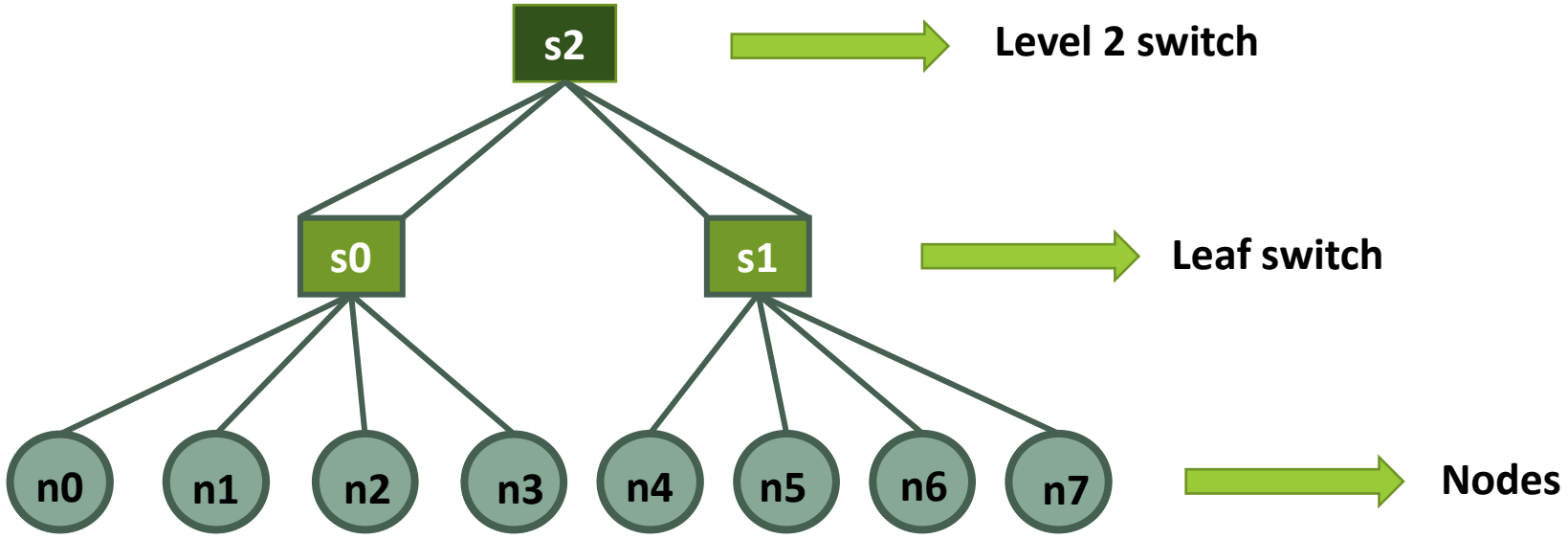
[1] 2020. MPICH. https://www.mpich.org.

OBJECTIVE

Developing node-allocation algorithms that consider the job's behaviour during resource allocation to improve the performance of communication-intensive jobs

Network Topology

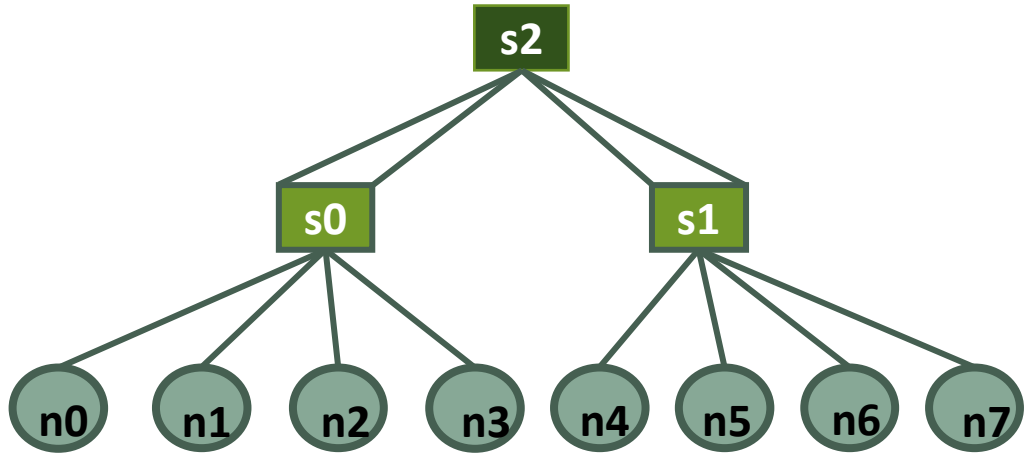
We use fat-tree[1] based network topology in our study



[1] C. E. Leiserson. 1985. Fat-trees: Universal networks for hardware-efficient super-computing. IEEE Trans. Comput.10 (1985), 892–901.

SLURM – Simple Linux Utility for Resource Management[1]

- Select/linear plugin allocates entire nodes to jobs
- Supports tree/fat-tree network topology via topology/tree plugin
- Default SLURM algorithm uses best-fit allocation



[1] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In Job Scheduling Strategies for Parallel Processing.

Communication Patterns

- We assume that submitted parallel jobs use MPI for communication
- Global communication matrix
  - May not reflect most crucial communications
  - Temporal communication information is not considered
- We consider the underlying algorithms of MPI collectives
- We consider three standard communication patterns – recursive doubling (RD), recursive halving with vector doubling (RHVD) and binomial tree algorithm

Communication Patterns

- Gives a more definitive communication pattern without incurring profiling cost
  - Important for applications where the collective communication costs dominate the execution times
- Our strategies consider all stages of algorithms (RD, RHVD, Binomial) and allocate based on the costliest communication step/stage
  - Difficult to achieve using a communication matrix

Communication-aware Scheduler

- We propose mainly two node-allocation algorithms – greedy, balanced
- Every job is categorized as compute or communication intensive
  - Can be deduced using MPI profiles of MPI application[1] or through user input
- Algorithms identify lowest-level common switch with requested number of nodes available
- If this lowest-level switch is leaf-switch → requested number of nodes allocated to the job

[1]Benjamin Klenk and Holger Fröning. 2017. An Overview of MPI Characteristics of Exascale Proxy Applications. In High Performance Computing. Springer International Publishing,

Common Notations

Notation	Description
i	Node index
L_i	Leaf Switch connected to node *i*
L_nodes	Total number of nodes on the leaf switch
L_comm	Number of nodes running communication-intensive jobs on the leaf switch
L_busy	Number of nodes allocated on the leaf switch

Greedy Allocation

- Minimize network contention by minimizing link/switch sharing
- For communication-intensive job select the leaf switches which:
  - Have maximum number of free nodes
  - Minimum number of running communication-intensive jobs

We characterize leaf switches using their communication ratio

$$Communication\ Ratio\ (L) = \frac{L\_comm}{L\_busy} + \frac{L\_busy}{L\_nodes}$$

Number of communication-intensive jobs relative to the busy nodes on leaf switch

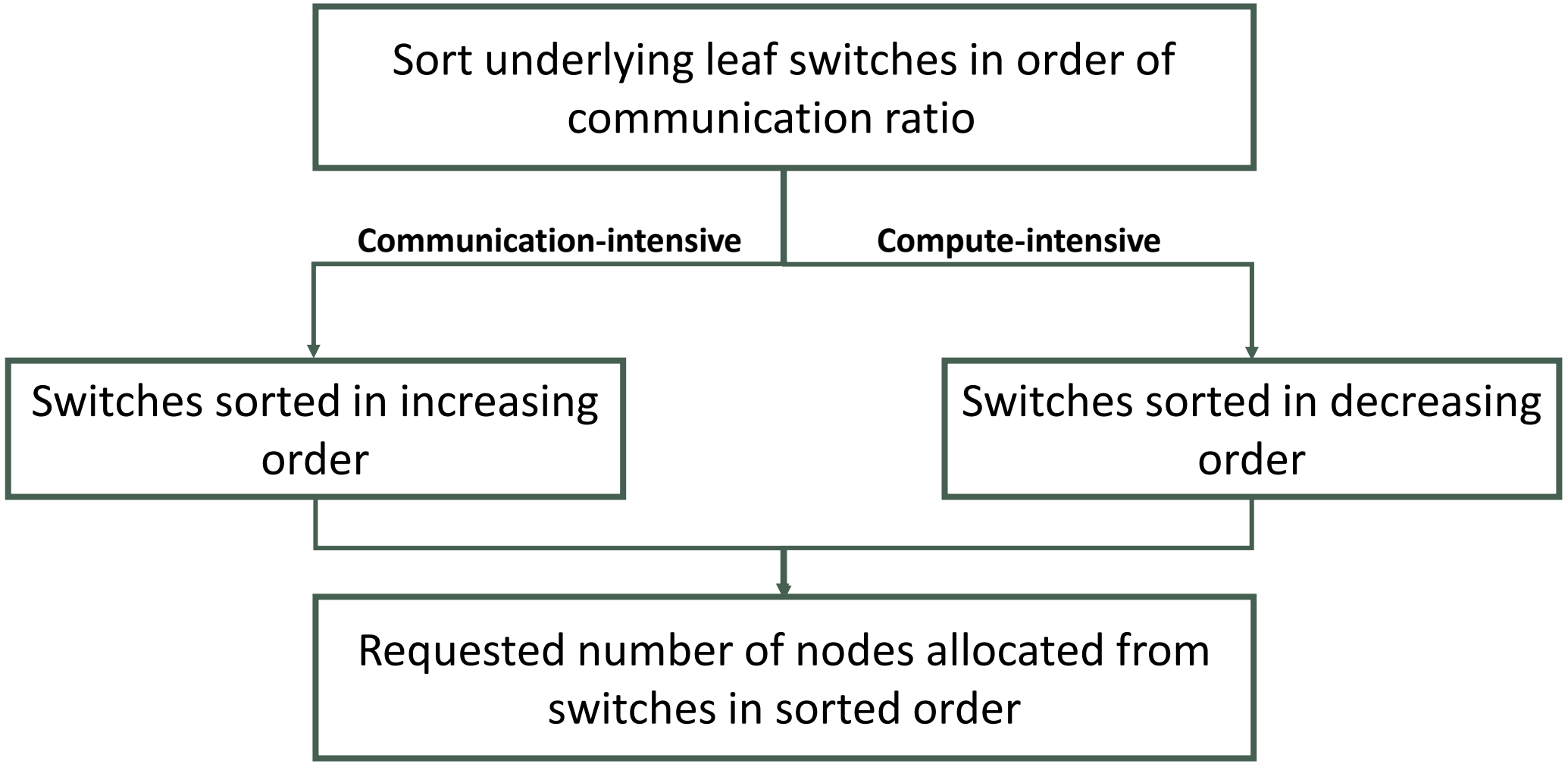
Measure of contention

Measure of available nodes on leaf switch

Controls node-spread

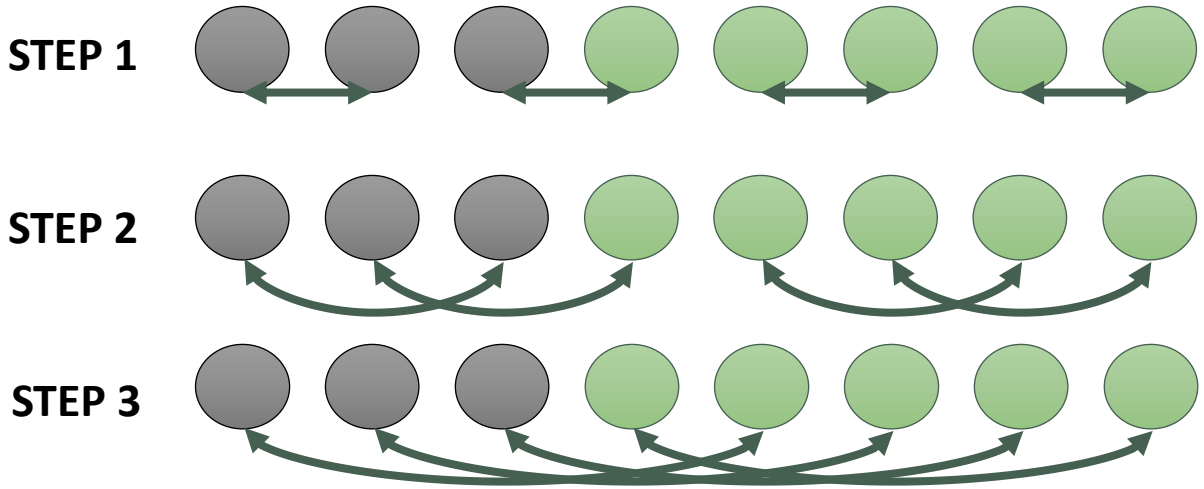
Lower communication-ratio → lower contention and higher number of free nodes

Design of Greedy Allocation

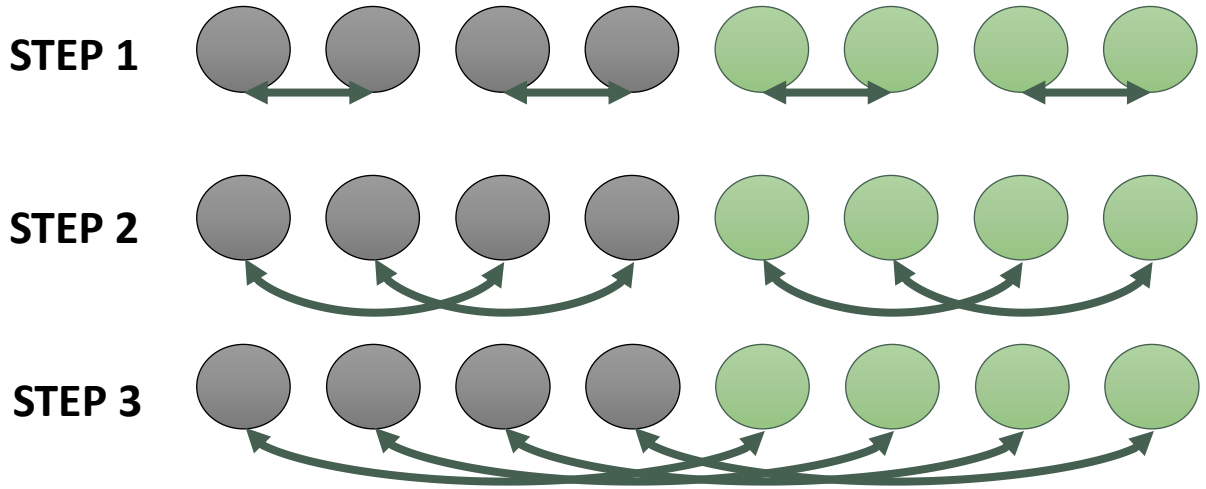


Balanced Allocation

- Aims at allocating nodes in powers of two to minimize inter-switch communication

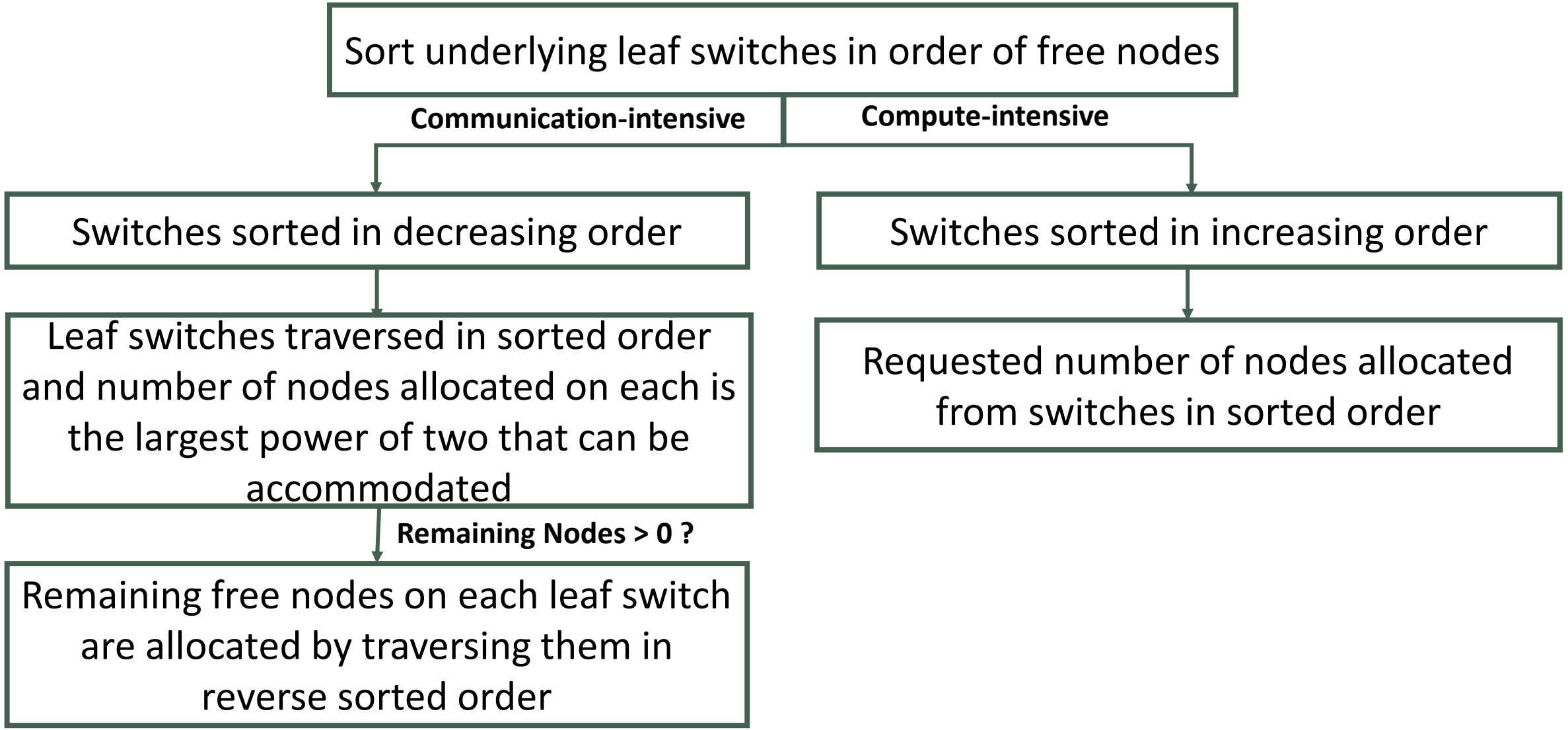


Unbalanced Allocation



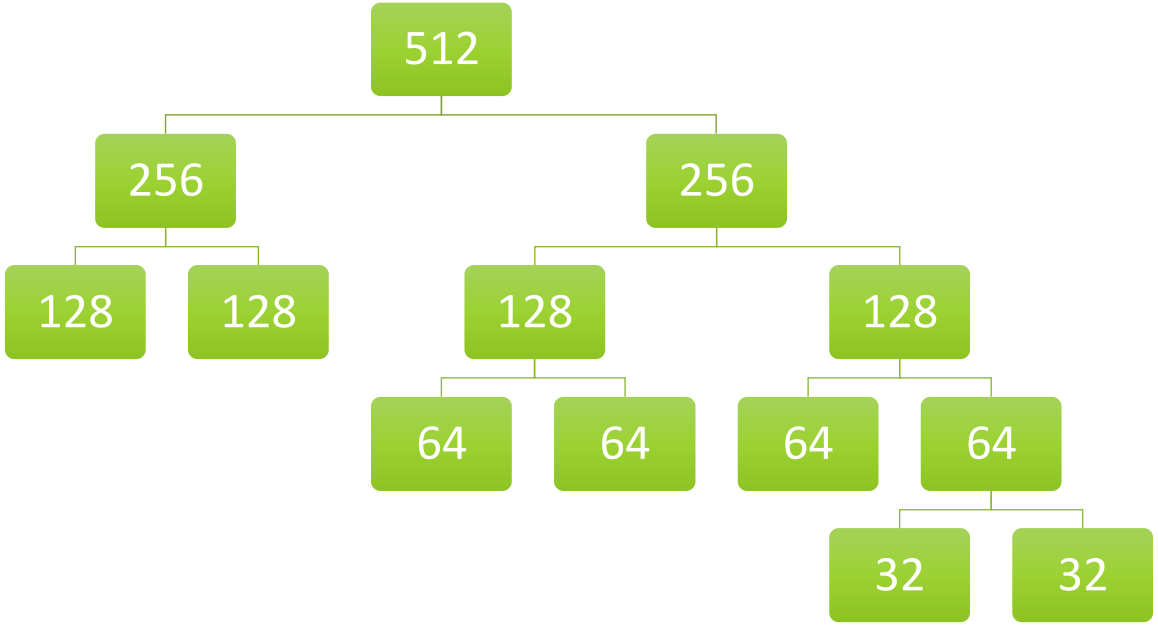
Balanced Allocation

Design of Balanced Allocation



Consider a job that requires 512 nodes

Leaf Switch	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]	L[7]
Free Nodes	160	150	100	80	70	50	40
Allocated Nodes	128	128	64	64	64	32	32



Adaptive Allocation

- Greedy allocation minimizes contention and fragmentation
  - Unbalanced, more inter-switch communication
- Balanced allocation minimizes inter-switch communication
  - More fragmentation
- Adaptive allocation compares both allocations and selects the more optimal node allocation based on their cost of communication

Experimental Setup

- We evaluate using job logs of Intrepid, Theta and Mira[1] supercomputers
  - Intrepid logs from Parallel Workload Archive[2]
  - Theta and Mira logs from Argonne Leadership Computing Facility[3]
- Contain job name, nodes requested, submission times, start times etc.
- 1000 jobs from each log

[1] 2020. Mira and Theta. https://www.alcf.anl.gov/alcf-resources

[2] 2005. Parallel Workload Archive. www.cse.huji.ac.il/labs/parallel/workload/

[3] 2019. ALCF, ANL. https://reports.alcf.anl.gov/data/index.html

Experimental Setup

- Do not have any information about nature of the job
  - Some jobs are assumed communication-intensive, others as compute-intensive
  - Percentage of communication-intensive jobs varied from 30% - 90%
- Jobs with power-of-two node requirements considered
- Job logs emulated by configuring SLURM with enable-front-end option
  - Run for same duration as their execution times

Runtime Estimates

The runtime of a job can be modelled as:

$$Total\ Runtime\ (T) = T_{compute} + T_{comm}$$

where:

$T_{compute}$: Compute time of a job

T_{comm} : Communication time of a job

Contention Factor *C(i,j)*

- Communicating nodes *i* and *j* are present on same leaf switch ($L_i = L_j$)

$$C(i,j) = \frac{L_i\_comm}{L_i\_nodes}$$

- Communicating nodes *i* and *j* are present on different leaf switches ($L_i \neq L_j$)

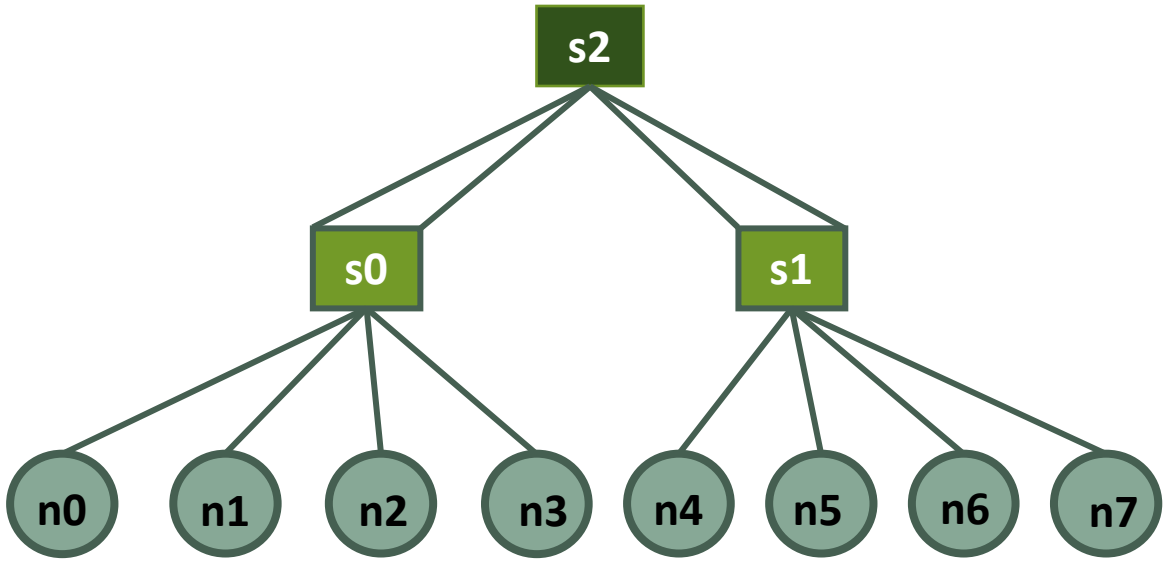
$$C(i,j) = \frac{L_i\_comm}{L_i\_nodes} + \frac{L_j\_comm}{L_j\_nodes} + \frac{1}{2}\frac{L_i\_comm + L_j\_comm}{L_i\_nodes + L_j\_nodes}$$

Contention on individual leaf switches

Contention on lowest-level common switch connecting the two leaf switches

Distance *d(i,j)*

$$d(i,j) = 2 * Lowest\ level\ of\ common\ switch$$



d(n0, n1) = 2

d(n0, n5) = 4

Cost of communication

Effective hops between communicating nodes *i* and *j* is:

$$Hops(i,j) = d(i,j) * \big(1 + C(i,j)\big)$$

Total cost of communication:

$$Cost = \sum_{n=1}^{N} \max_{i,j \in S_n} Hops(i,j)$$

where:

- S_n: Set of all node-pairs communicating at *n*th step
- *N*: Total number of steps in the communication algorithm

Modified Runtime

$$Modified\ Runtime(T') = T_{compute} + T_{comm} * \frac{Cost\_Jobaware}{Cost\_Default}$$

where:

$Cost\_Jobaware$: Cost of communication for job-aware algorithm

$Cost\_Default$: Cost of communication for default algorithm

Evaluation metrics

1. Execution time – Time between start and completion of job
2. Wait time – Time between submission and start of job
3. Turnaround time – Time between submission and completion of job
4. Node Hour – Number of nodes * Execution time
5. Cost of communication

Types of Experiments

- Continuous Runs
  - 1000 jobs are run using the submission times derived from logs
- Individual Runs
  - Jobs are submitted one at a time to a partially occupied cluster
  - Provides common starting point to compare allocation of each job

Impact on Execution Time and Wait Time

Job Log	Algorithms	Execution Time (Hours)			
		Default	Greedy	Balanced	Adaptive
Intrepid	RHVD	1382	1351	1256	1251
	RD		1345	1264	1257
Theta	RHVD	2189	1740	1700	1663
	RD		1810	1731	1706
Mira	RHVD	3289	3956	2342	2435
	RD		3285	2559	2637

Table: Execution times (in hours) in all three job logs

Average improvement in execution time – 9%
Average improvement in wait time – 31%

- 90% jobs considered communication-intensive
- Balanced and adaptive always perform better than default and greedy
- Decrease in execution times makes resources available faster → wait times decrease
- Average wait time reductions were 35%, 26% and 32% for Intrepid, Theta and Mira
- Little or negative improvement for Mira under greedy allocation
  - Communicating node-pairs on same switch in default but not greedy
  - Difference in available links/switches – hence, we also compare using individual runs

Continuous vs Individual Runs

Job Log	Algorithms	Execution Time (% improvement)		
		Greedy	Balanced	Adaptive
Intrepid	RHVD	3.65	7.23	7.81
	RD	1.70	8.12	8.29
Theta	RHVD	9.65	9.65	9.65
	RD	13.56	13.56	13.56
Mira	RHVD	10.84	19.69	21.71
	RD	9.45	24.32	24.91

Table: Improvements in execution times

- For a given state of cluster, proposed algorithms always perform better than default
  - 2-13% improvement using greedy allocation
  - 7-25% improvement using balanced and adaptive allocation
- Similar to continuous, balanced and adaptive perform better than greedy

Variation in Communication Patterns

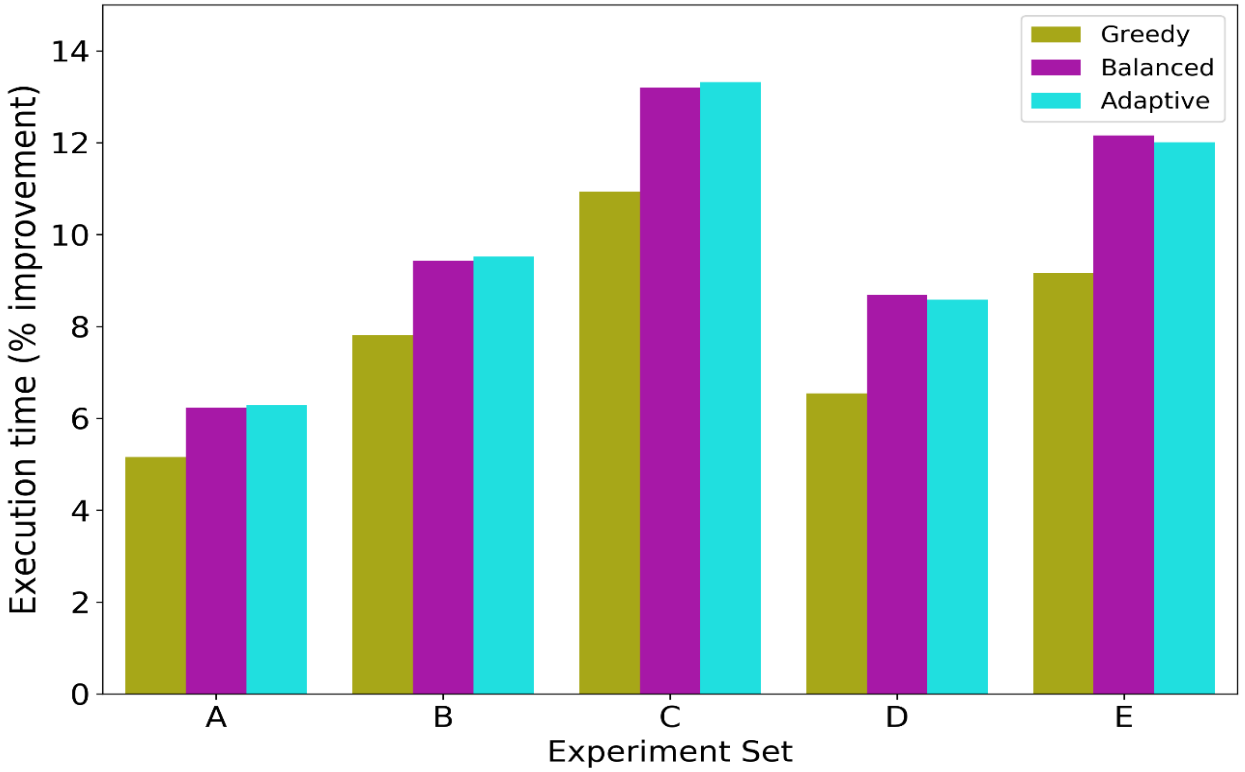


Figure: Reduction in execution time using various communication patterns for Theta

- **A** – 67% compute, 33% RHVD
  B – 50% compute, 50% RHVD
  C – 30% compute, 70% RHVD
  D – 50% compute, 15% RD, 35% Binomial
  E – 30% compute, 21% RD, 49% Binomial
- For same communication pattern, as communication ratio increases from A (33%) to C (70%), gain increases
  - Larger fraction of execution time reduces
  - Similarly between D and E

Conclusion

- Proposed three node allocation algorithms to improve performance of communication-intensive jobs
- Evaluated algorithms using three supercomputer job logs
- Demonstrate that proposed algorithms improve execution times, wait times and system throughput

Future Work

- Include other communication patterns
- Explore process mapping after node allocation
- Extend to other topologies

Thank You!