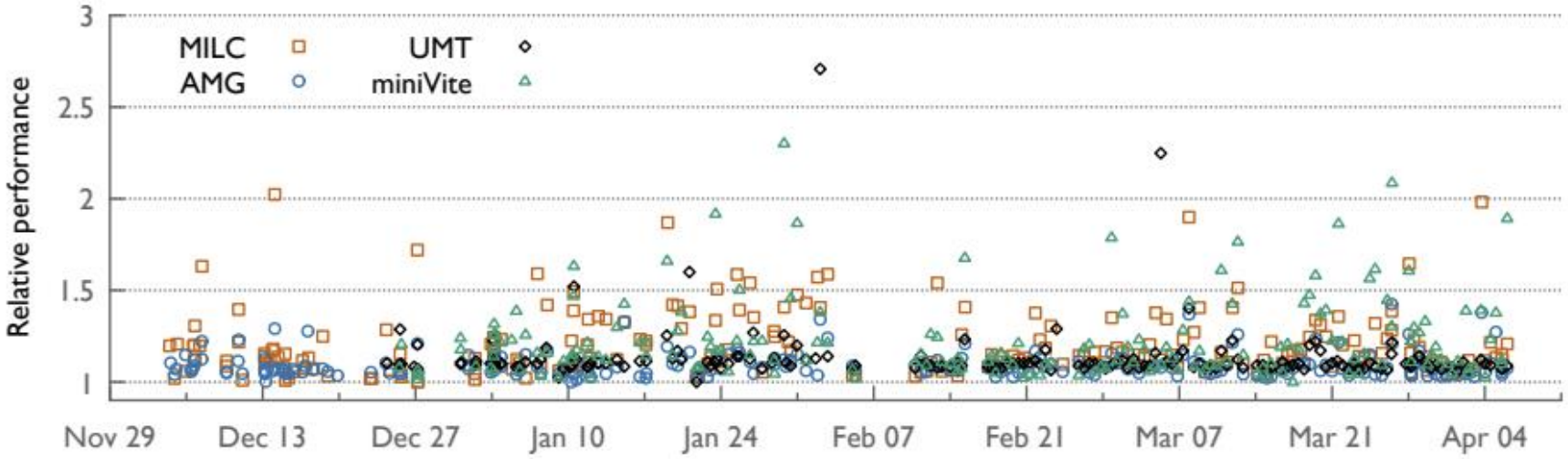
Communication-balanced Job Allocation using SLURM

Gagandeep Mangat[1*] and Preeti Malakar[2]

[1]Qualcomm Inc.
[2]Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
**Work done while at IIT Kanpur*

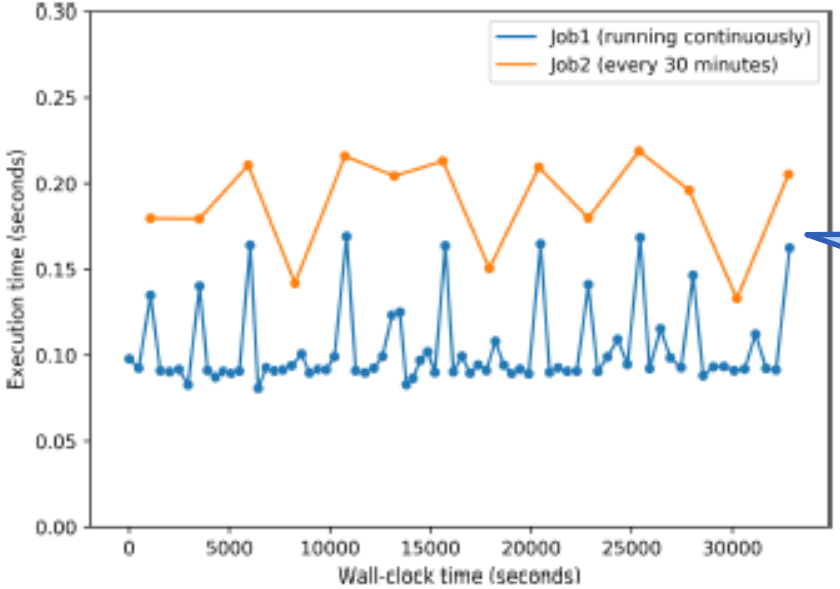
28th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)
IPDPS 2025

Performance Variability in HPC



Bhatele et al., “Case of Performance Variability on Dragonfly-based Systems”, IPDPS 2020

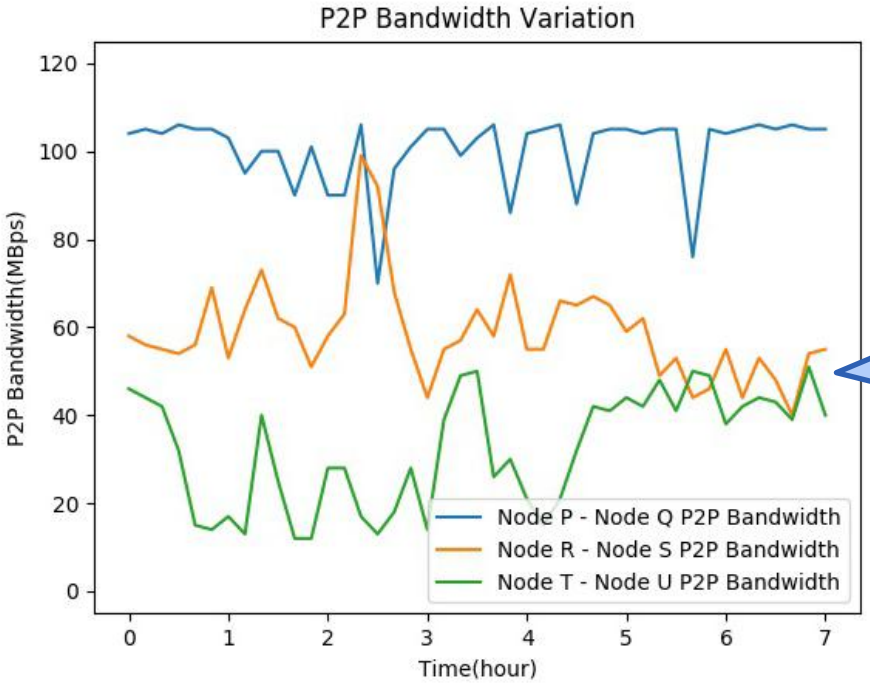
Impact of Job Interference



Variation in Job1 due to Job2

Mishra et al., “Communication-aware Job Scheduling using SLURM”, ICPPW 2020

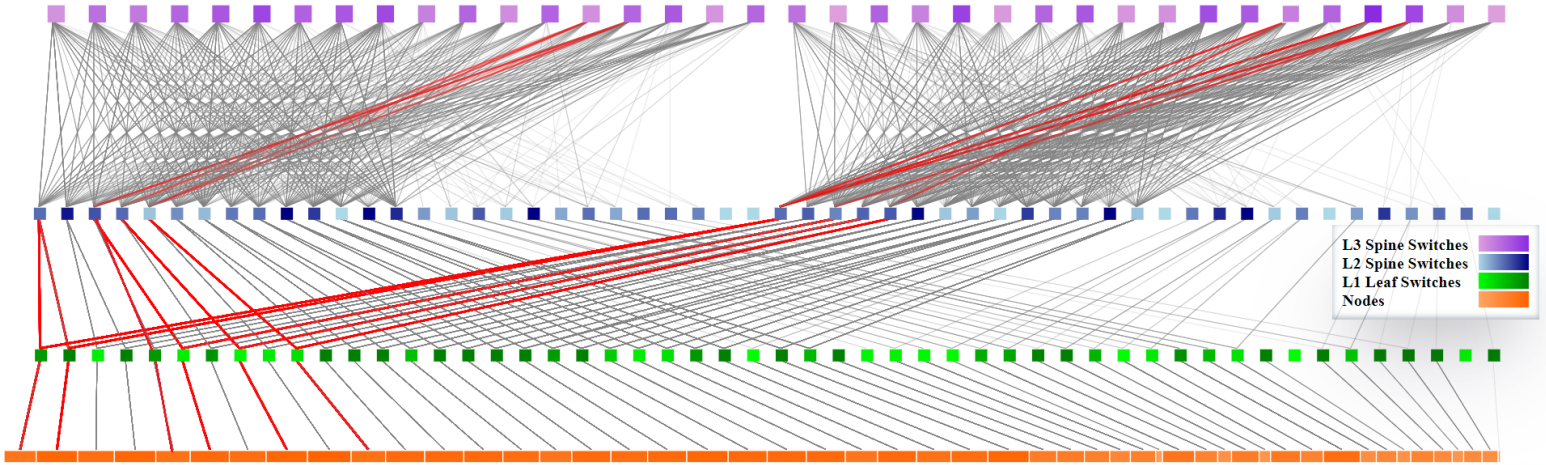
Variation in Network Bandwidth



Variation in pairwise effective bandwidth between nodes connected in a tree topology

Kumar et al., “Network and Load-Aware Resource Manager for MPI Programs”, ICPPW 2020

Shared Communication Paths

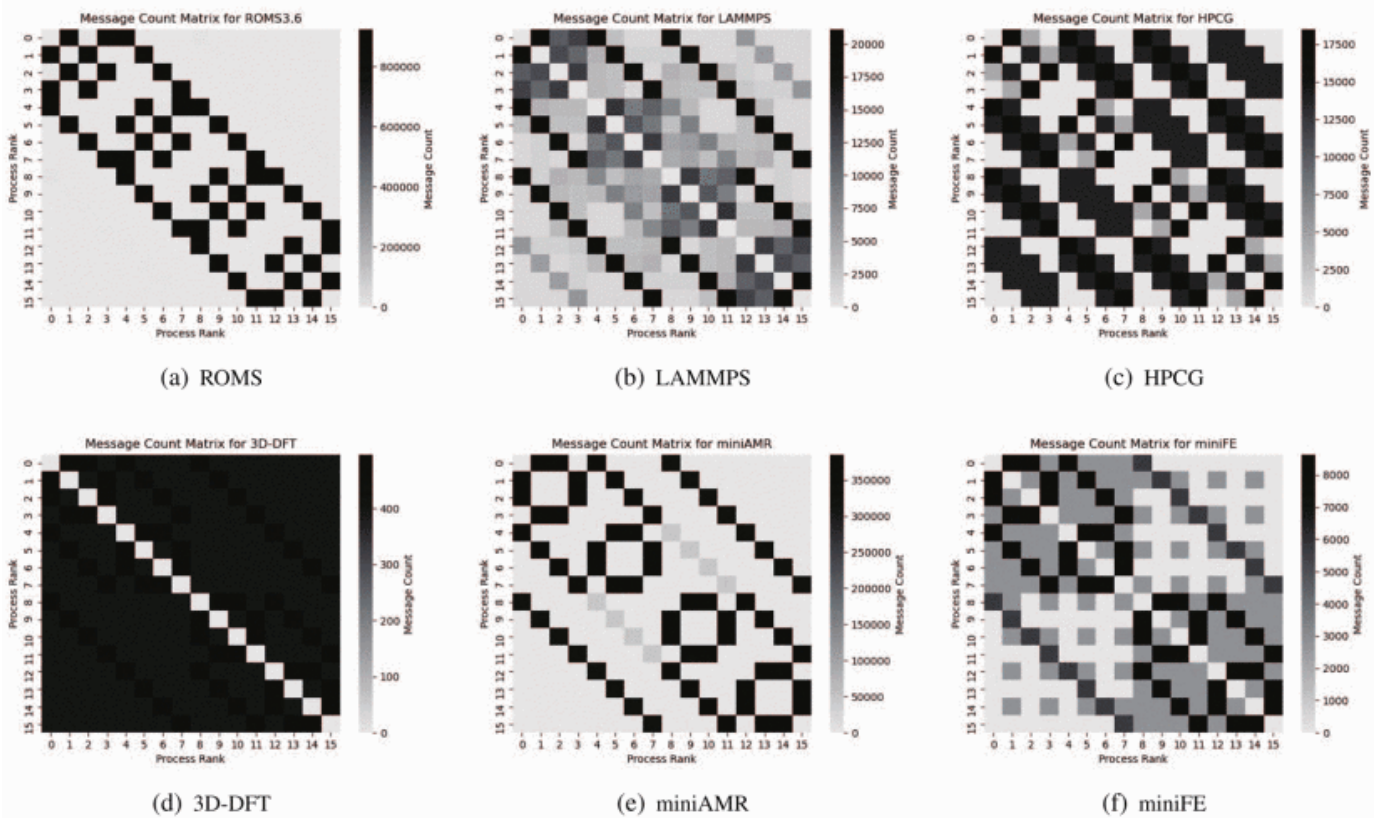


	JobID	UserName	QueueName	TotalNodes	TotalCores	RequiredTime	JobState	ElaspedTime
⊕	1004182.un05	[redacted]	medium	4	80	96:00	R	44:53
⊕	1004941.un05	[redacted]	medium	4	80	96:00	R	76:53
⊖	1004949.un05	[redacted]	large	6	120	96:00	R	49:32
	NodeList: hpc013,hpc018,hpc027,hpc104,hpc139,hpc169							
⊕	1005316.un05	[redacted]	small	2	40	120:0	R	48:22

Showing 1 to 150 of 150 entries

Sharma et al., “Visual Analysis of Congestion and Interference in Supercomputers”, HiPC SRS 2023

Communication Patterns



(a) ROMS (b) LAMMPS (c) HPCG

(d) 3D-DFT (e) miniAMR (f) miniFE

Agrawal et al., “IPMPI: Improved MPI Communication Logger. ”, ExaMPI@SC 2022

Communication Challenges Summary

- Communication performance variation
- Application bandwidth requirements vary
- Distinct communication patterns
- Job interference

Communication plays a huge role in real-time performance

Job Request Parameters

```
#SBATCH –N 2
#SBATCH --ntasks-per-node=2
#SBATCH --error=job.%J.err
#SBATCH --output=job.%J.out
#SBATCH --time=00:10:00
#SBATCH --partition=standard mpirun –np 4 ./exe
```

- Number of processes
- Wall-clock time
- Communication metric

Related Work

- Jokanovic et al. [IPDPS'15] proposed to eliminate job interactions in the neighborhood to minimize contention and load on switches
- Jeannot et al. [EuroPar'10] proposed TopoMatch that maps processes to resources in order to reduce the communication cost of the application
- Pollard et al. [SC'18] propose a fat-tree network topology-aware node allocation policy in the Flux resource management that allocates isolated partitions to jobs for eliminating inter-job interference
- Mishra et al. [ICPPW'20] proposed algorithms based on the cluster state and the communication patterns of MPI collective calls
- In contrast, we allocate resources based on the communication matrix and current switch load to reduce the overall cost of the job

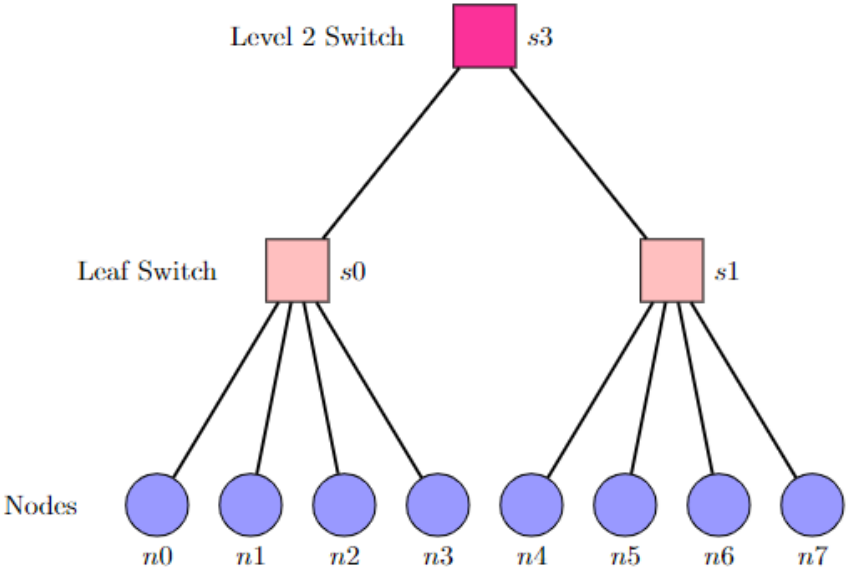
Communication-balanced Allocation

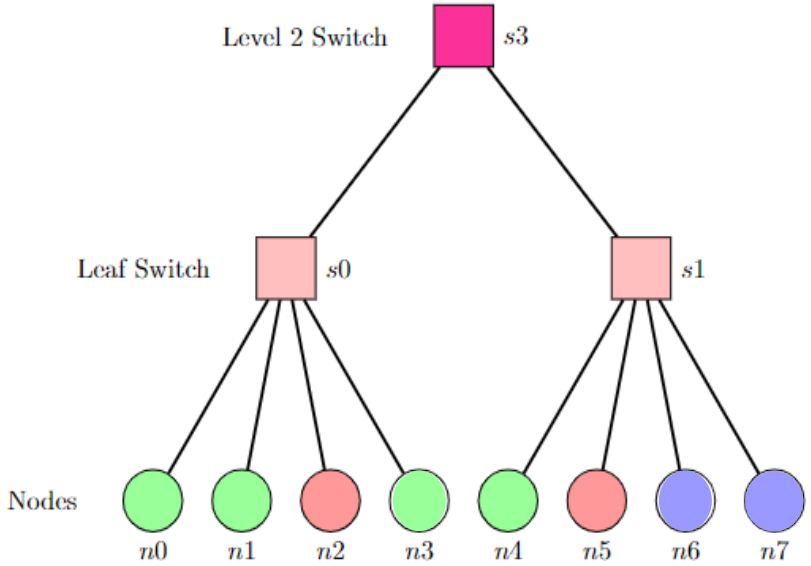
Objectives

- Consider communication as an additional parameter while allocating jobs
- Partition node request to groups of processes such that intra-group communication volume is high and inter-group communication volume is low
- Allocate groups of processes that do not incur high communication volume on separate nodes

Allocate groups of processes to leaf switches of a fat-tree as per the communication characteristics of the application such that most of the communications within an application are contained within a switch.

Job Allocation on Fat-tree Topology

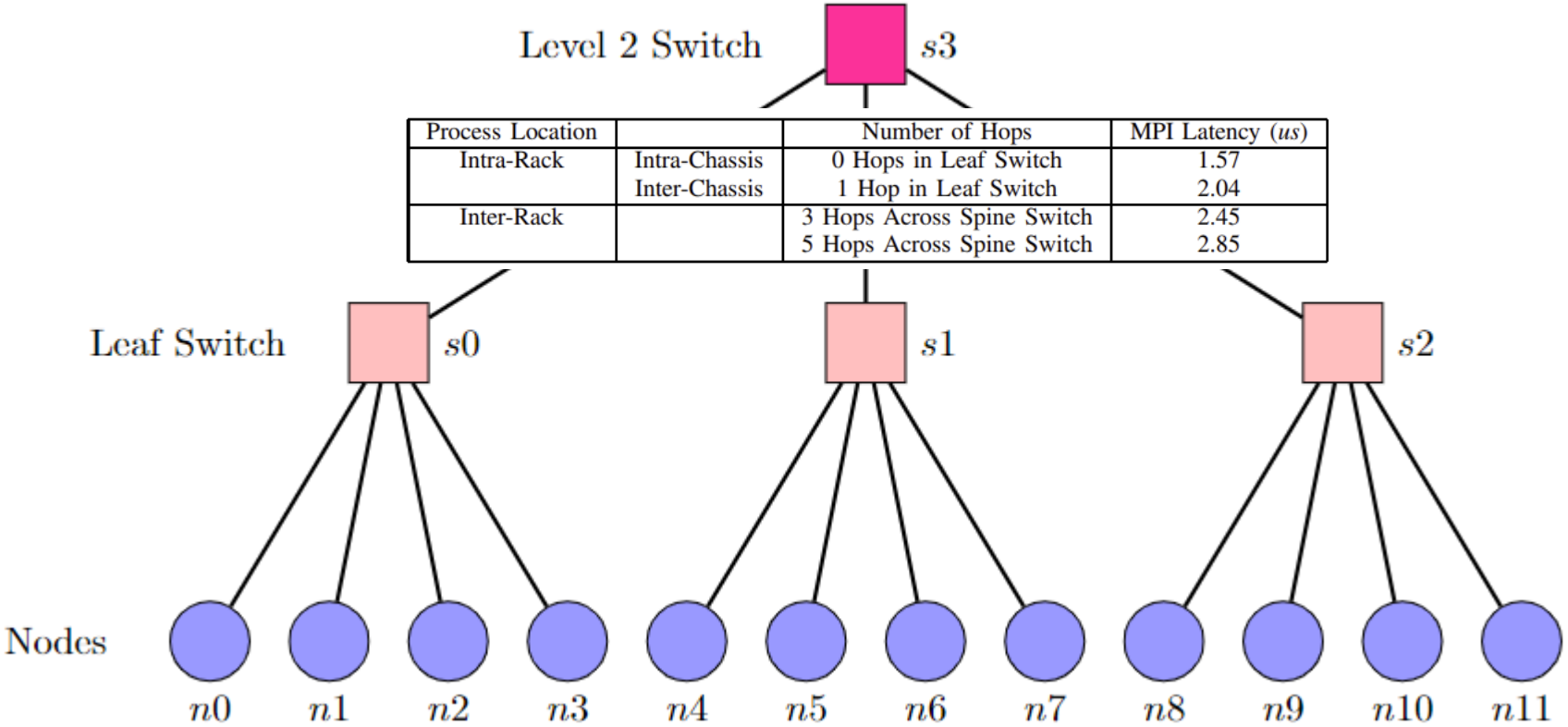




0 1 0 2 3 1

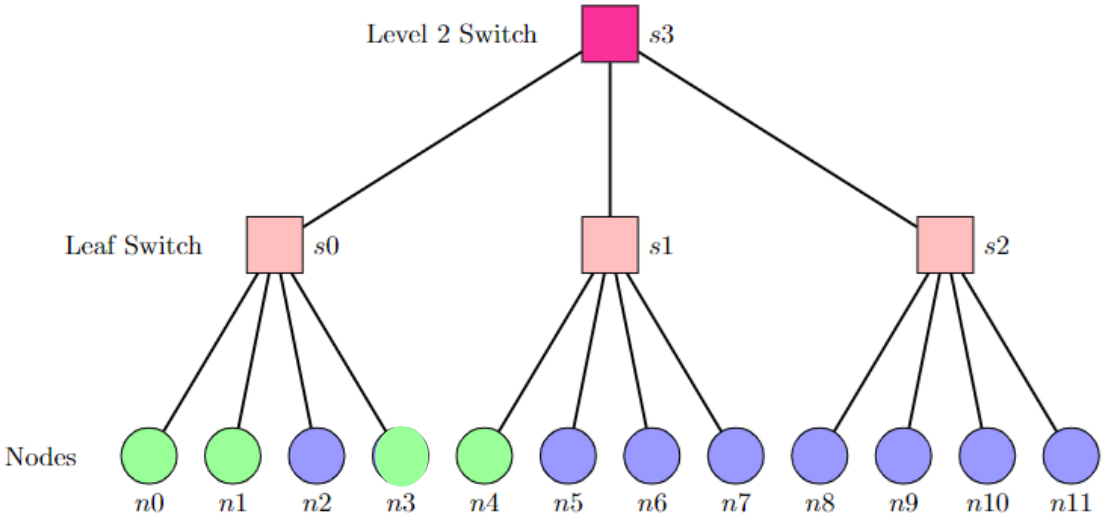
0 1 0 3 2 1

Intra-switch vs. Inter-switch



Process Location		Number of Hops	MPI Latency (*us*)
Intra-Rack	Intra-Chassis	0 Hops in Leaf Switch	1.57
	Inter-Chassis	1 Hop in Leaf Switch	2.04
Inter-Rack		3 Hops Across Spine Switch	2.45
		5 Hops Across Spine Switch	2.85

Communication Matrix



An example current system state when a job J will be allocated nodes

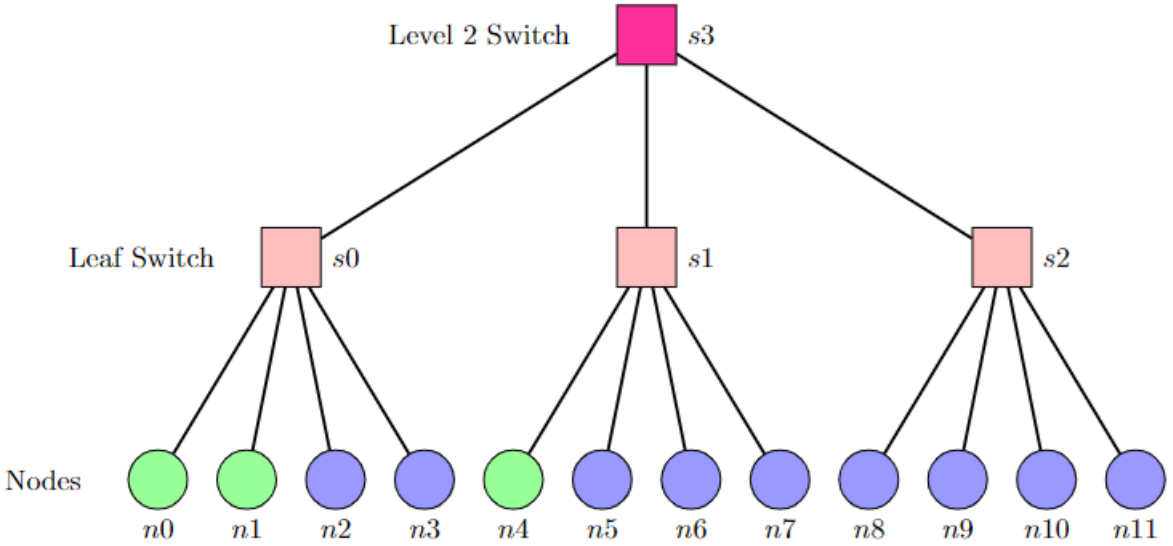
	0	1	2	3
0	0	100	20	180
1	100	0	70	250
2	20	70	0	80
3	180	250	80	0

An example communication matrix

Communication-balanced (Combal)

I. Find the optimal lowest level switch and sort the leaf switches in decreasing order of number of free nodes (L). Find the median (L[m]).

Example: Sorted order: s_2:4, s_1:3, s_0: 2

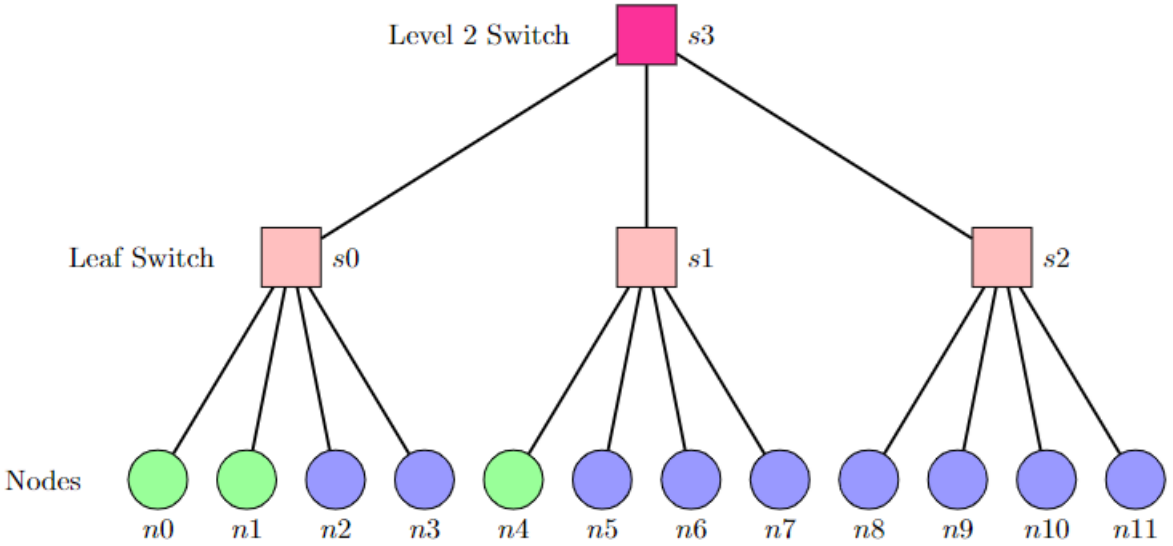


Communication-balanced (Combal)

II-a. Partition the requested #nodes (R) using a graph partitioner (METIS) to reduce inter-partition communication volume

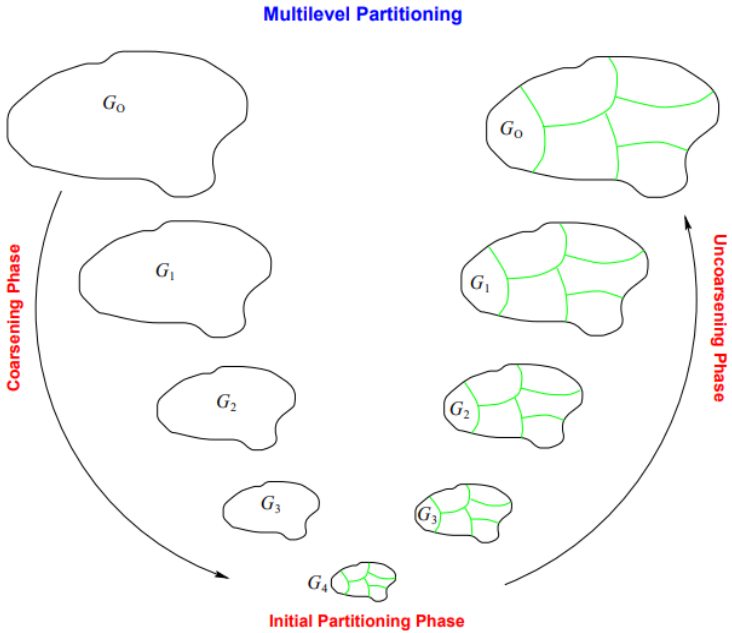
Required number of partitions (R / free(L[m]))

Example: 6 / 3 = 2



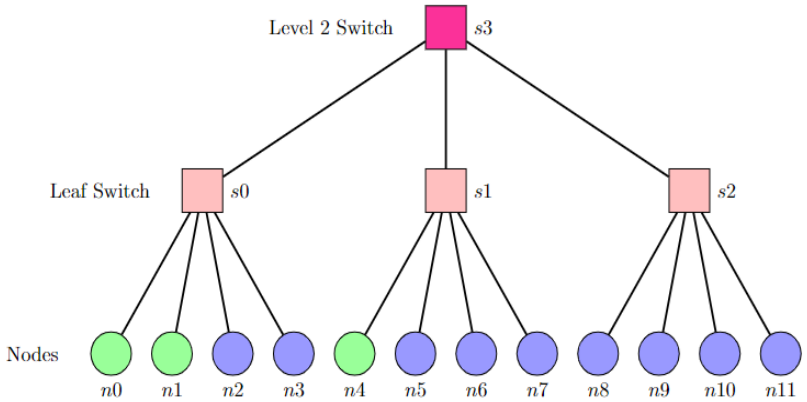
METIS Graph Partitioner

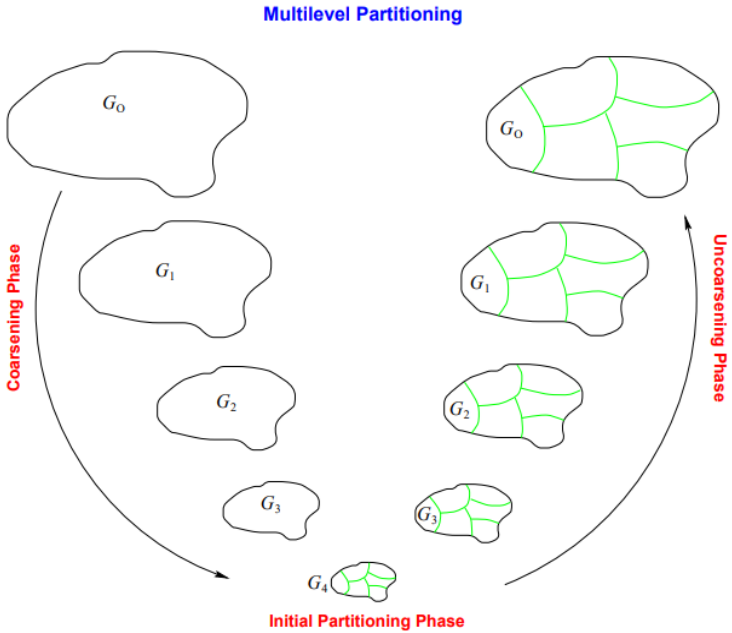
- Multilevel k-way partitioning
- Three stages: Coarsening, Partitioning, Uncoarsening
- Resultant partitions have high intra-partition volume and low inter-partition volume



Devine K, Boman EG, Karypis G (2006) Partitioning and load balancing for emerging parallel applications and architectures, Parallel Processing for Scientific Computing, SIAM.

METIS Graph Partitioner





options[METIS OPTION UFACTOR]

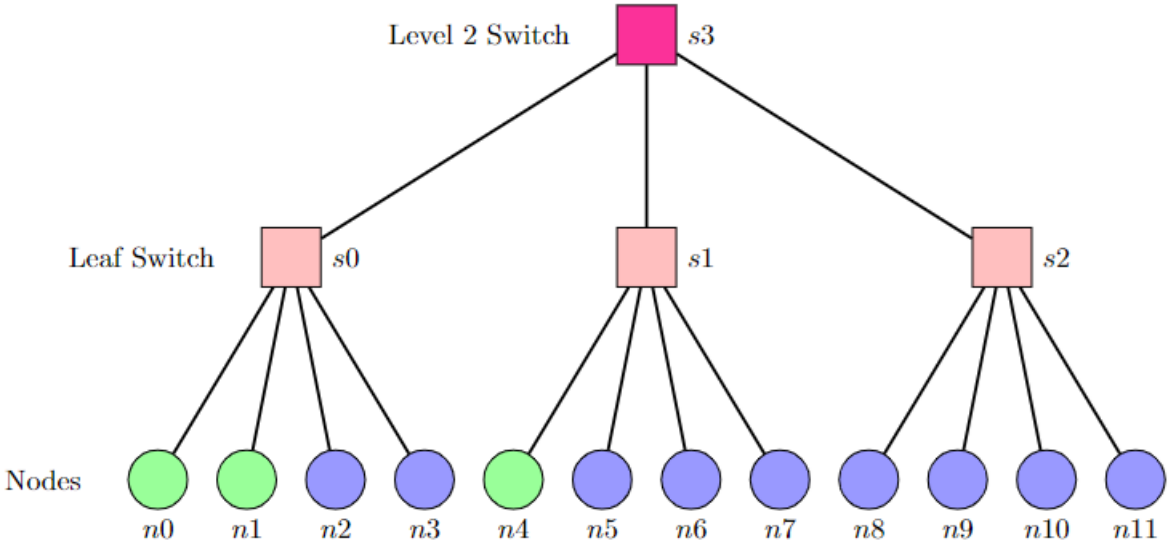
Specifies the maximum allowed load imbalance among the partitions

Devine K, Boman EG, Karypis G (2006) Partitioning and load balancing for emerging parallel applications and architectures, Parallel Processing for Scientific Computing, SIAM.

Size of Partitions (uCombal)

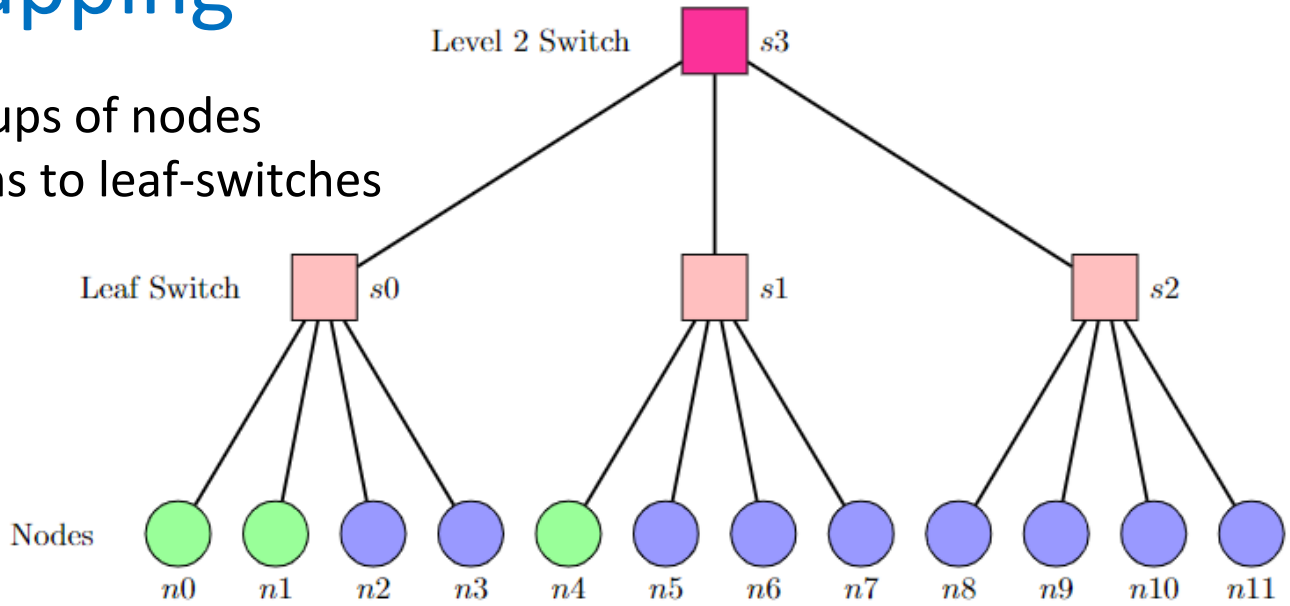
II-b. Control the size of partitions using unbalanced factor (free(L[0] / free(L[m]))

Unbalanced factor = 4 / 3 = 2



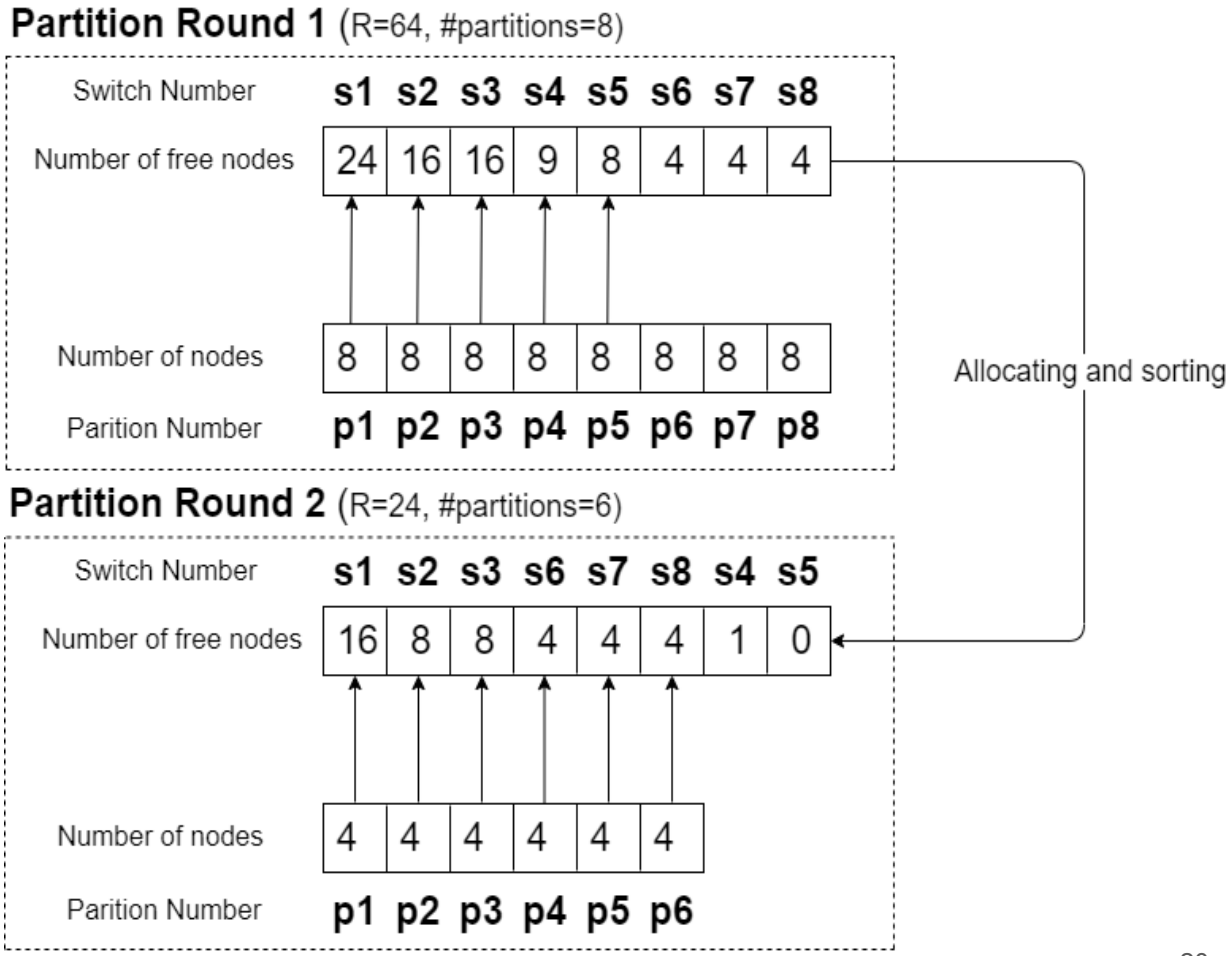
Partition Mapping

- SLURM outputs groups of nodes
- Map entire partitions to leaf-switches

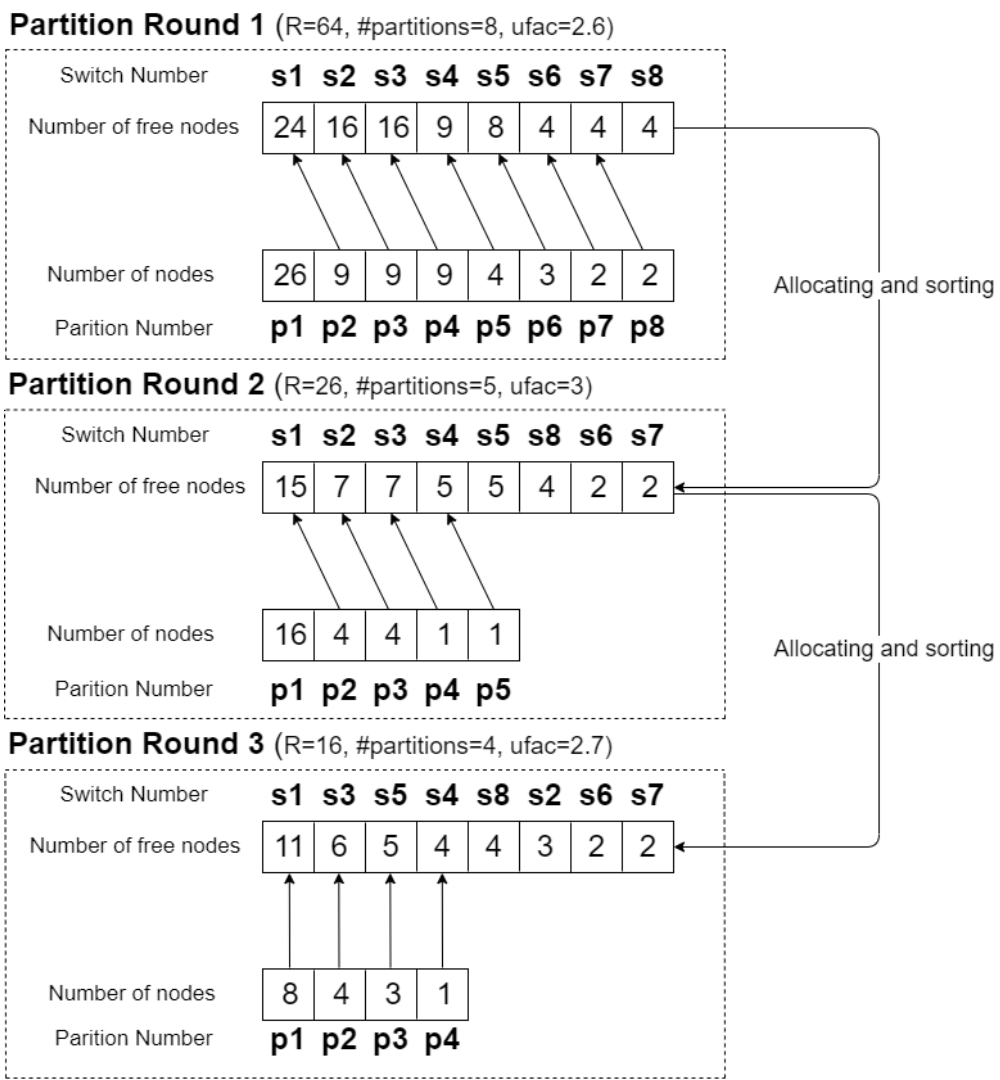


Default	**0**	**1**		**2**	**3**	**4**	**5**				
Combal				**1**	**2**	**5**	**0**	**3**	**4**		
uCombal				**1**	**2**		**0**	**3**	**4**	**5**	

Partitioning using Combal



Partitioning using uCombal



SLURM Code Modifications

```
struct node_record {
  char *name; /* name of the node */
  uint16_t cpus; /* count of processors on the node */
  uint16_t cores; /* number of cores per socket */
  uint16_t threads; /* number of threads per core */
  struct node_record *node_next; /* next entry with same hash index */
  int leaf_switch; /* index of the leaf switch connected to the node */
};
```

SLURM Code Modifications

```
struct switch_record {
  int level; /* level in hierarchy, leaf=0 */
  char *name; /* switch name */
  bitstr_t *node_bitmap; /* bitmap of all nodes descended from this switch */
  uint16_t parent; /* index of parent switch */
  int comm_jobs; /* number of communication-intensive jobs on this switch */
  int num_nodes; /* number of direct descendant nodes*/
};
```

The selected nodes on a leaf switch are stored in the node bitmap field of the switch record.

SLURM Code Modifications

- (u)Combal is invoked whenever a new job is scheduled by SLURM
- Number of free nodes on every switch is the input
- METIS API is invoked from SLURM
- Partitions output by METIS are used in SLURM to map the processes to nodes in each leaf switch

Experimental Evaluations

Simulation Based Evaluations Using Real Job Logs

- SLURM simulator [1]
- Real job logs: Intrepid, Mira, and Theta supercomputers [2]

```
User JobID Timelimit Submit Eligible Start End Elapsed NNodes NCPUS NodeList Wait Turnaround

gagandeep 17344 UNLIMITED "2022-06-15 11:48:24" "2022-06-15 11:48:24" "2022-06-15 11:48:24" "2022-06-15
12:10:24" 00:22:00 128 128 cn[209-224,241-256,289-304,321-336,401-464] 0 26400

gagandeep 17345 UNLIMITED "2022-06-15 11:48:24" "2022-06-15 11:48:24"
"2022-06-15 11:48:24" "2022-06-15 12:02:41" 00:14:17 16 16 cn[385-400] 0 17140

gagandeep 17346 365-00:00:00 "2022-06-15 11:48:24" "2022-06-15 11:49:37"
"2022-06-15 11:49:49" "2022-06-15 12:04:22" 00:14:33 16 16 cn[369-384] 240 17700
```

[1] https://github.com/SchedMD/slurm

[2] https://reports.alcf.anl.gov/data/

Add Communication Data

User JobID Timelimit Submit Start End Elapsed NNodes NCPUS NodeList Wait Turnaround

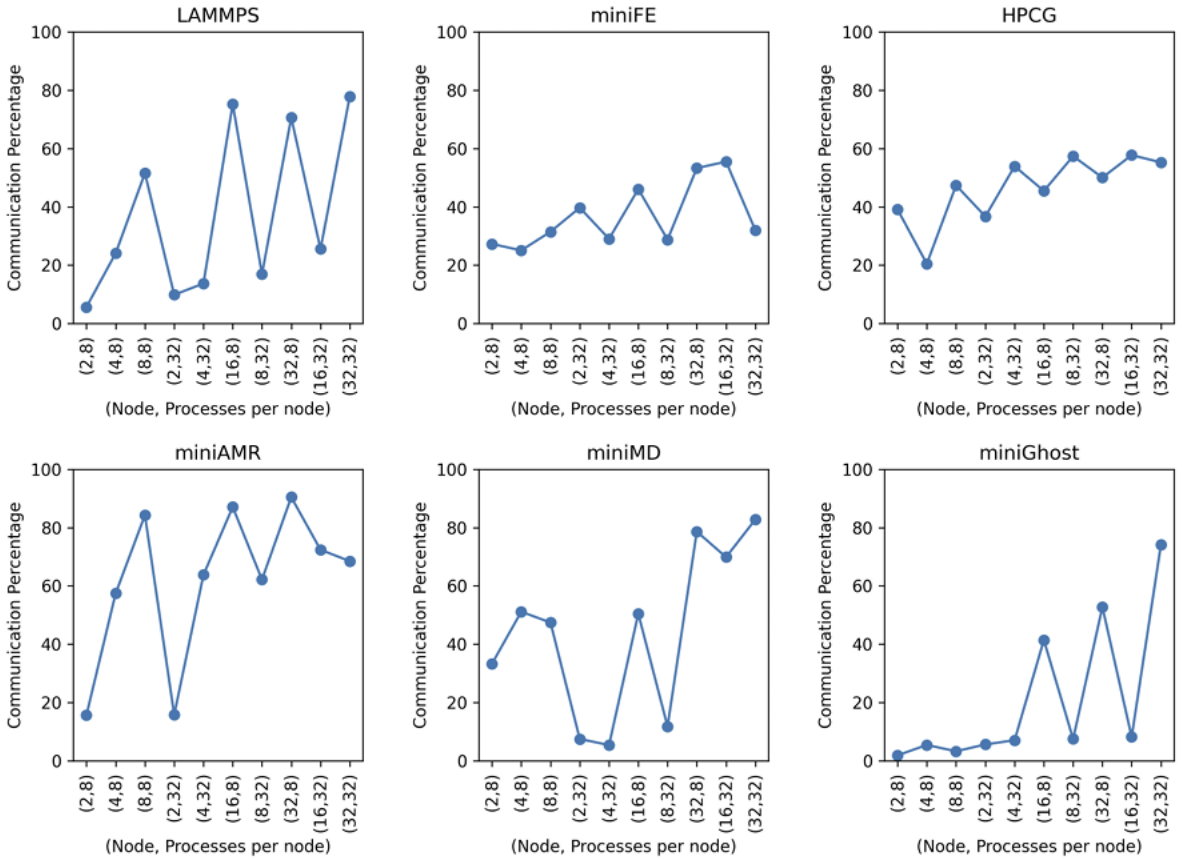
User JobID Timelimit Submit Start End Elapsed NNodes NCPUS NodeList Wait Turnaround Communication Matrix

$$\begin{bmatrix} 0 & 364350000 & 400824000 & 2905760 \\ 364350000 & 0 & 2905760 & 400824000 \\ 400824000 & 2905760 & 0 & 364350000 \\ 2905760 & 400824000 & 364350000 & 0 \end{bmatrix}$$

4-process HPCG Communication Matrix obtained from IPMPI

Agrawal and Malakar, “IPMPI: Improved MPI Communication Logger.”, ExaMPI@SC 2022

Challenge 1: Applications



- Identify six top unique jobs using *user_id* and *project_id*
- Extracted equal number of entries (167*6)
- Map six known applications to these jobs (using same number of processes)

Modified Job Log

JobID SubmitTime Runtime Nodes Comment(Communication matrix) Percentage
1 0 1320 128 1:/home/gagandeep/matrices/comm_ipmpi_miniAMR_32_128_4_2.mat 90.17
2 0 857 16 1:/home/gagandeep/matrices/comm_ipmpi_miniAMR_4_16_4_1.mat 70.99
3 73 873 16 1:/home/gagandeep/matrices/comm_ipmpi_miniAMR_4_16_4_1.mat 70.99
4 73 866 16 1:/home/gagandeep/matrices/comm_ipmpi_miniAMR_4_16_4_1.mat 70.99
5 103 858 16 1:/home/gagandeep/matrices/comm_ipmpi_miniAMR_4_16_4_1.mat 70.99
6 103 882 16 1:/home/gagandeep/matrices/comm_ipmpi_miniAMR_4_16_4_1.mat 70.99
7 281 746 64 1:/home/gagandeep/matrices/comm_ipmpi_lammps_16_64_4_0.mat 57.31
8 282 749 64 1:/home/gagandeep/matrices/comm_ipmpi_lammps_16_64_4_0.mat 57.31
9 295 405 16 0:/home/gagandeep/matrices/comm_ipmpi_MiniGhost_4_16_4_0.mat 5.47
10 295 401 16 0:/home/gagandeep/matrices/comm_ipmpi_MiniGhost_4_16_4_0.mat 5.47

Communication-intensive applications categorization (user-defined): >40% communication time

Challenges 2 and 3: Topology and Simulation

- SLURM requires the network configuration file (topology.conf)
- Used supercomputer at IIT Kanpur to retrieve the network topology and application communication matrices
- Capped at 512 cores of PARAM Sanganak, IIT Kanpur
  - avoid high queue waiting times to collect the real communication matrices

Evaluation Methodology

- 1002 job logs from each of the three job logs (four runs)
- Run the jobs using default SLURM allocation algorithm (FCFS with backfilling)
- Run the jobs using TopoMatch algorithm process mapping
- Run the jobs using SLURM + (u)Combal allocation algorithm to obtain the process and node allocation

Challenge 4: Runtime Estimation

$$T_{runtime} = T_{compute} + T_{comm}$$

$$T_{comm'} = T_{comm} * \frac{New\ \ communication\ \ cost}{Old\ \ communication\ \ cost}$$

$$T_{runtime} = T_{compute} + T_{comm'}$$

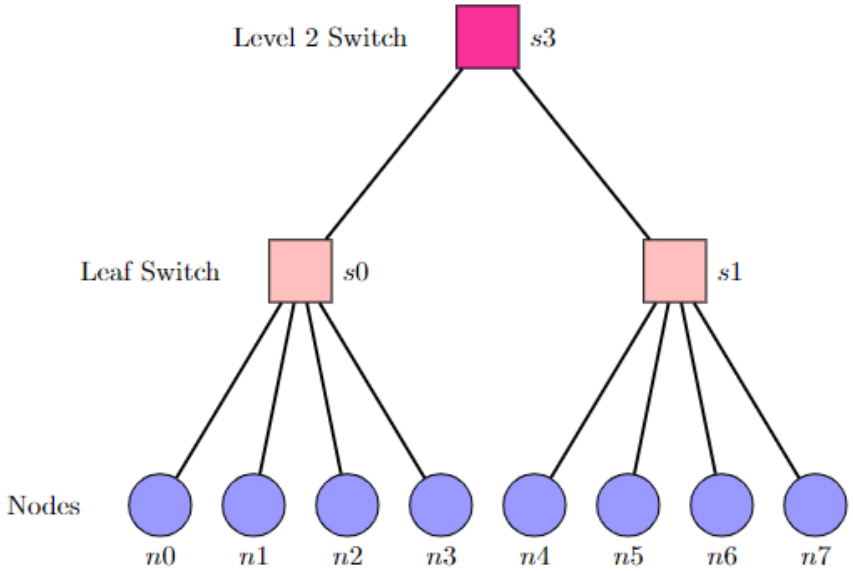
Modified runtime obtained using our communication cost model

Communication Cost Model

$$\text{hop-bytes}(i,j) = comm(n_i, n_j) * d(n_i, n_j)$$

$$\text{Cost} = \sum_{\substack{(i,j) \\ \in (NxN)}} \text{hop-bytes}(i,j)$$

$d(n_i, n_j)$ = 2 * lowest level of common switch



Agarwal et al., Topology-aware task mapping for reducing communication contention on large parallel machines, IPDPS 2006

Contention Factor

$$C(i,j) = \begin{cases} \dfrac{L^i{}_{comm}}{L^i{}_{nodes}} & L^i = L^j \\ \dfrac{L^i{}_{comm}}{L^i{}_{nodes}} + \dfrac{L^j{}_{comm}}{L^j{}_{nodes}} + \dfrac{1}{2}\dfrac{L^i{}_{comm} + L^j{}_{comm}}{L^i{}_{nodes} + L^j{}_{nodes}} & L^i \neq L^j \end{cases}$$

Evaluation Methodology

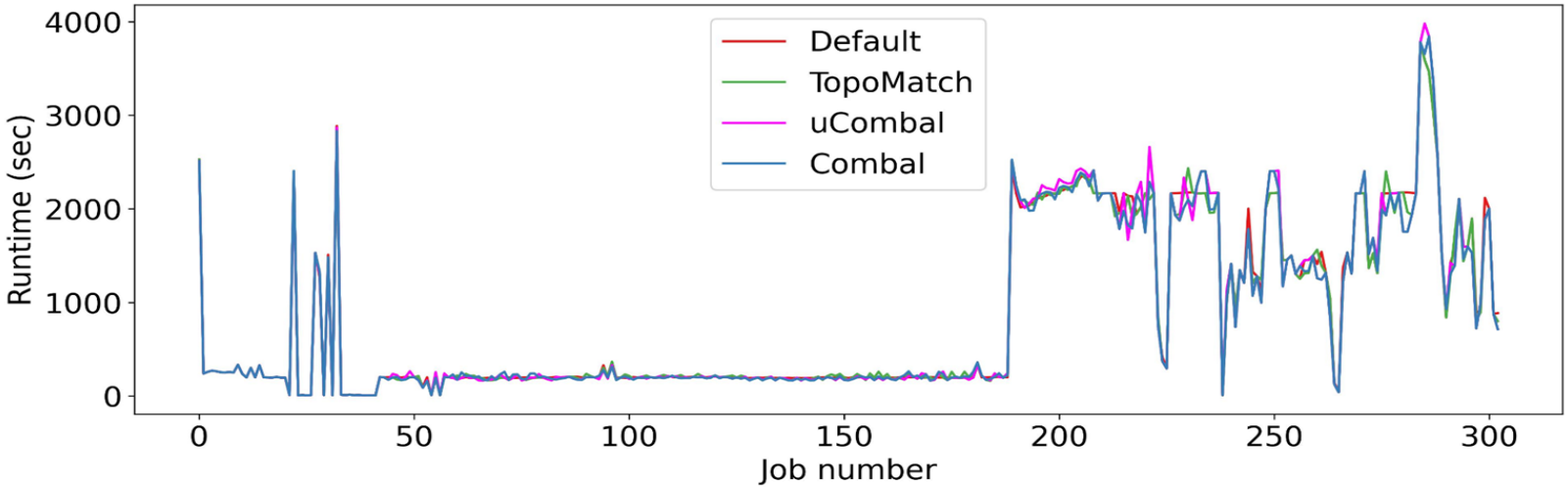
- 1002 job logs from each of the three job logs (four runs)
- Run the jobs using default SLURM allocation algorithm (FCFS with backfilling)
- Run the jobs using TopoMatch algorithm process mapping
- Run the jobs using SLURM + (u)Combal allocation algorithm to obtain the process and node allocation
- Run the jobs using SLURM + (u)Combal allocation algorithm using the modified runtimes

Results

Evaluation Metrics

1. Execution times (runtimes)
2. Wait times
3. Hop bytes
4. Total and average inter-switch and intra-switch communications

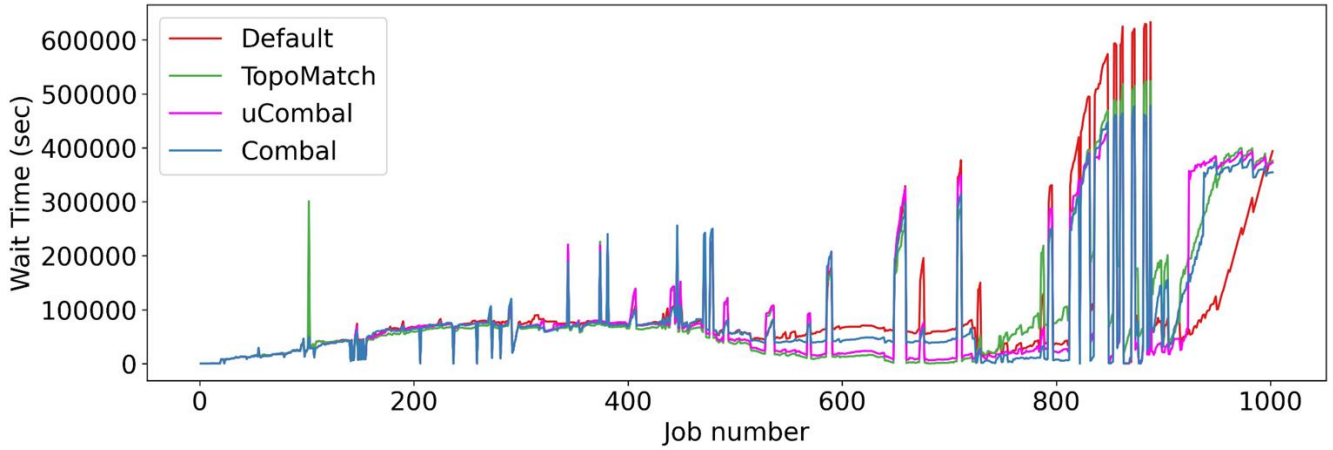
Execution Time (hours) for Intrepid Job Log



# Jobs	Default	TopoMatch	uCombal	Combal
303	1644	1637	1641	1631

- Reduced for all three algorithms in comparison to default algorithm
- Reduction in time mostly for miniAMR 32 nodes and miniFE 16 nodes

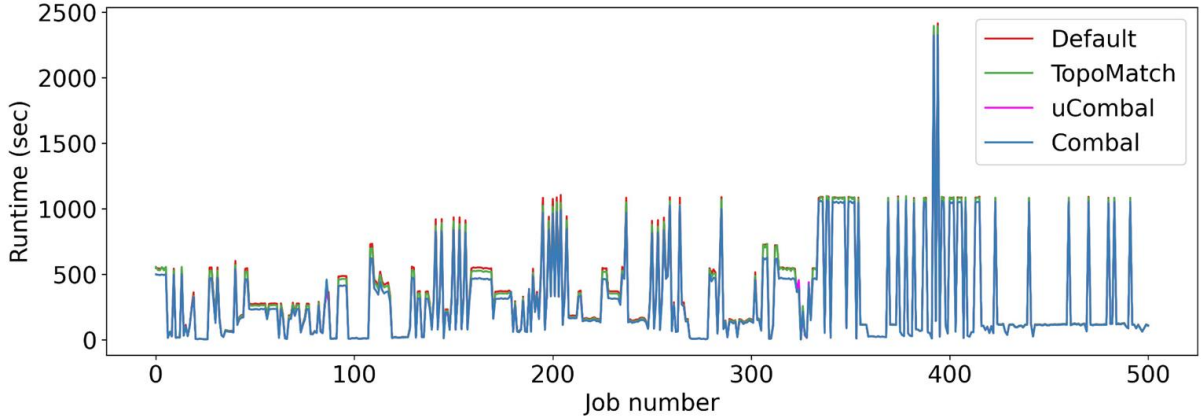
Wait Time (hours) for Mira Job Log



# Jobs	Default	TopoMatch	uCombal	Combal
410	25807	26110	25049	25453

Decreased by 2.9% (758 hours) for uCombal algorithm and 1.36% (354 hours) for Combal algorithm.

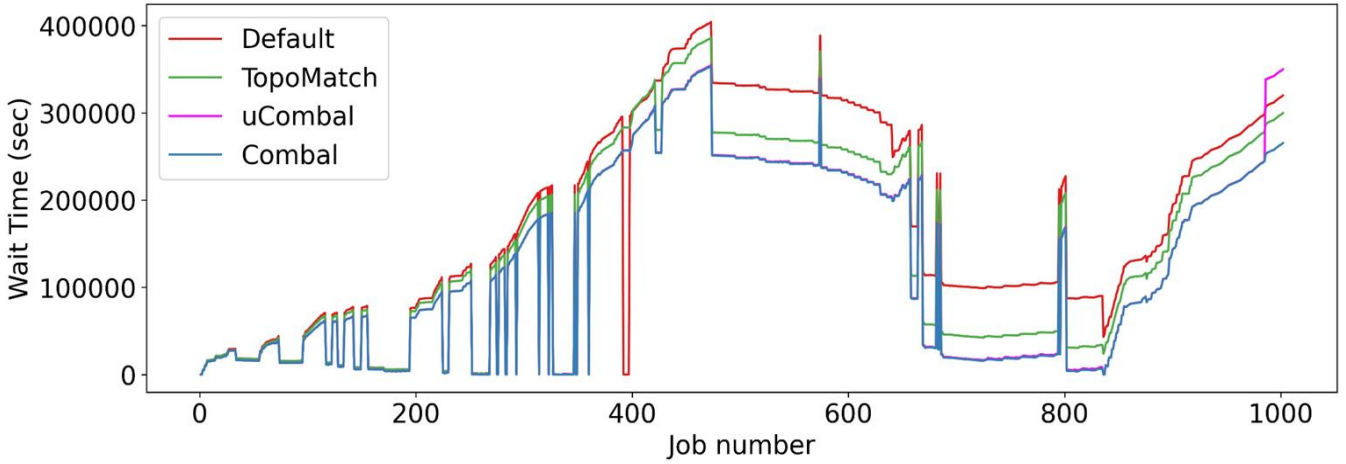
Execution Time (hours) for Theta Job Log



# Jobs	Default	TopoMatch	uCombal	Combal
501	675	664	637	637

- Decrease of 5.8% with Combal and uCombal
- Majorly reduced for miniAMR 256-node and miniMD 256-node jobs

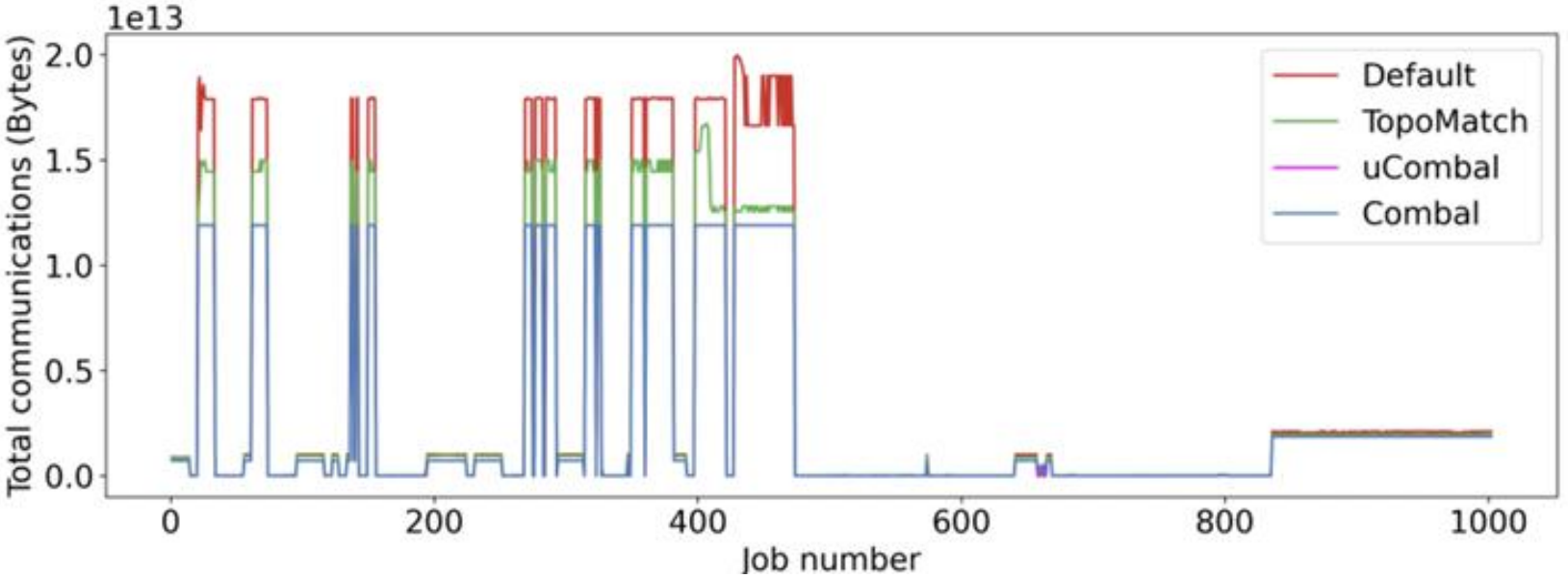
Wait Time (hours) for Theta Job Log



# Jobs	Default	TopoMatch	uCombal	Combal
501	47493	40987	35821	35291

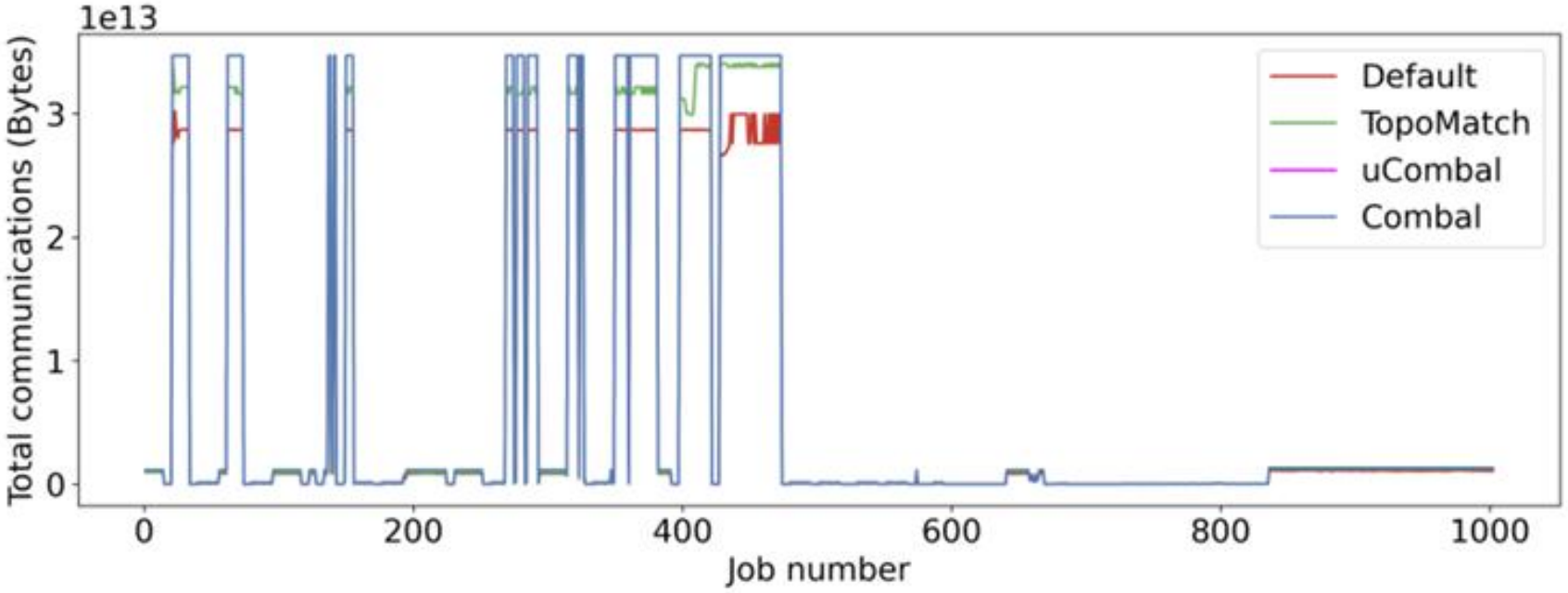
25.6% improvement with uCombal algorithm

Total Inter-Switch Communications for Theta Job Log



Lesser for Combal/uCombal in comparison to Default allocation which resulted in better runtime/wait times.

Total Intra-Switch Communications for Theta Job Log



More for Combal/uCombal in comparison to Default allocation which resulted in better runtime/wait times.

Cost Comparison of A Single Job

- HPCG 64-node job
- Default allocation cost: 7.27e11
- Combal allocation cost: 5.48e11
- 24.6% improvement

Switch Number	Selected Nodes	Default Allocation (Node numbers)	Combal Allocation (Node numbers)
s17	cn[161-176]	0 – 15	0, 4, 8, .., 60
s18	cn[177-192]	16 – 31	1, 5, 9, .., 61
s19	cn[193-208]	32 – 47	2, 6, 10, .., 62
s20	cn[225-240]	48 – 63	3, 7, 11, .., 63

Conclusions

- Proposed the Combal and uCombal algorithms to reduce the overall execution and wait time for a job by using its communication pattern.
- Compared with the default SLURM algorithm and the TopoMatch algorithm.
- Obtained a maximum improvement of 5.6% in the execution time and a maximum improvement of 25.6% in the wait time using the Theta job log.

Future Work

- Extend algorithm to other cluster topologies.
  - Redesign using other plugins in SLURM
  - Redefine cost model
- Use other graph partitioning libraries such as Scotch.
- Design I/O aware scheduling algorithms.
- Experiment with more applications and jobs.

Thank You

Questions?

pmalakar@iitk.ac.in