MPI Collective Algorithm Selection in the Presence of Process Arrival Patterns

Majid Salimi Beni[1], Biagio Cosenza[1], Sascha Hunold[2]

[1] Department of Computer Science
University of Salerno, Salerno, Italy

[2] Faculty of Informatics, TU Wien
Vienna, Austria

2024 IEEE International Conference on Cluster Computing (CLUSTER)
September 2024
Kobe, Japan

UNIVERSITÀ DEGLI STUDI DI SALERNO
TECHNISCHE UNIVERSITÄT WIEN Vienna | Austria
FWF Austrian Science Fund
IEEE CLUSTER 2024 Kobe, Japan • 24–27 September

Outline

Background

- MPI, Collective Operations and Algorithms
- MPI Collective Algorithm Selection

Motivation

- Process Arrival Patterns and Algorithm Selection

Methodology

- Micro-benchmarking technique

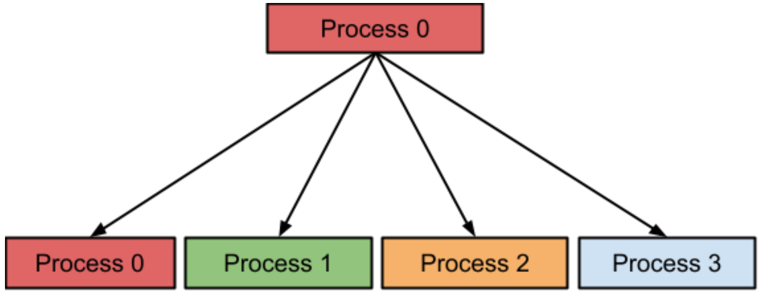
Experimental results

- Simulation study
- Real-world experiments
- Arrival patterns in the applications

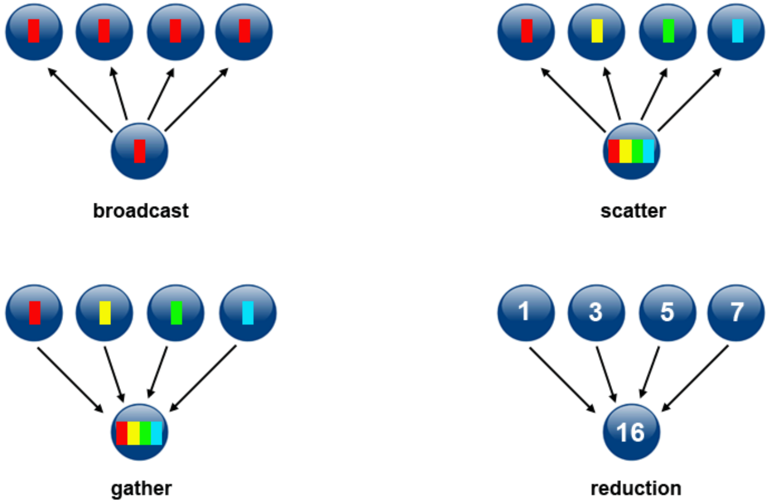
Conclusion and future work

Background: MPI, Collective Operations, and Algorithms

- MPI (Message Passing Interface)
  - A standard message-passing library designed to function on parallel computing architectures
- MPI collectives
  - Time-consuming: Big share of HPC applications' runtime is spent while performing collective communications
  - Efficient implementation of collective operations
    - Optimal performance
    - Scalability
    - Communication overhead
    - Resource utilization



https://hpc.nmsu.edu/discovery/mpi/introduction/



https://hpc-tutorials.llnl.gov/mpi/collective_communication_routines/

Background: MPI Collective Algorithm Selection

- MPI standard defines the **semantics** of collective operations
- Leaves their **algorithmic implementations** to MPI libraries
- MPI libraries provide several algorithms for each collective operation
- A decision logic selects one of these algorithms
- Algorithm selection of MPI collectives
  - Message size, process count, network topology, available hardware resources, network utilization
- Based on the scenario, one algorithm may outperform the others

MPI

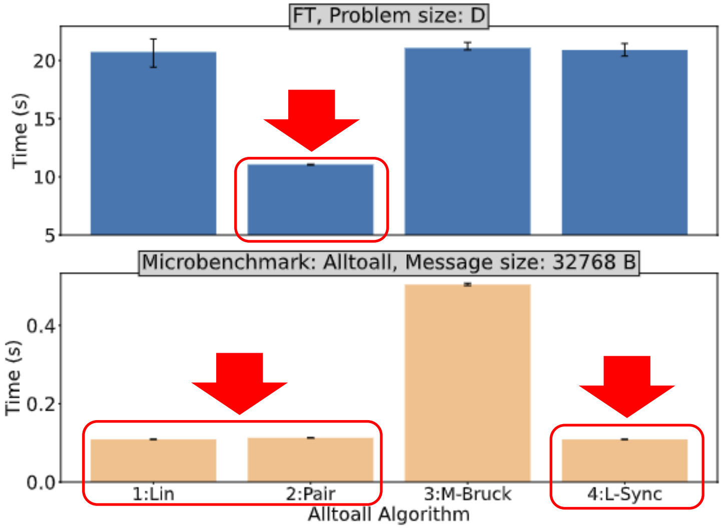
OPEN MPI MPICH MVAPICH

Intel® MPI Library

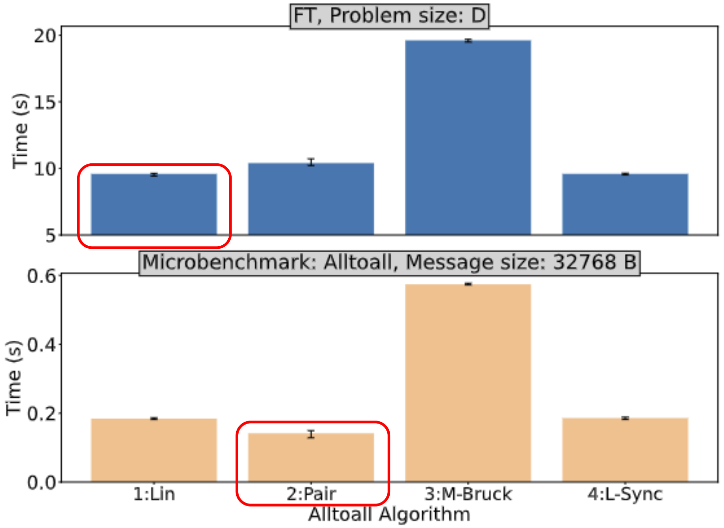
Motivation

- FT (problem size D) from NAS Parallel Benchmark
  - Communication-intensive
  - Profiling: MPI_Alltoall with a specific message size takes 50–70% of the total runtime
  - Application vs Micro-benchmark (with the message size found in the application)

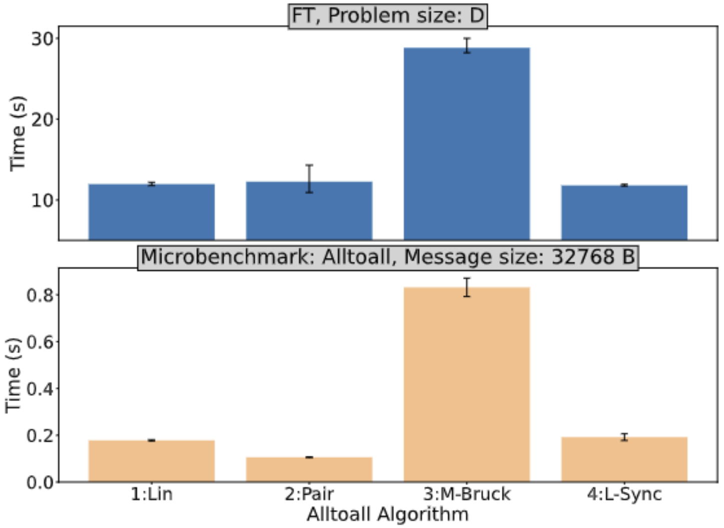
Observation: Choosing the fastest algorithm in the micro-benchmark, doesn't lead to the best performance in the application



(a) *Hydra*



(b) *Galileo100*



(c) *Discoverer*

Motivation: Process Arrival Patterns and Algorithm Selection

- ❑ In MPI applications, processes typically don’t enter collective operations simultaneously
  - ❑ System noise, performance variability, etc.
- ❑ **Process Arrival Patterns**
- ❑ **Hypothesis:** Collective algorithms may perform differently when there is process arrival pattern
  - ❑ Well-performing collective algorithm under a balanced process arrival pattern may show poor performance under an imbalanced process arrival pattern
- ❑ **Proposed solution**: Micro-benchmarking and exposing collective algorithms to different arrival patterns
  - ❑ Simulation (SimGrid toolkit)
  - ❑ Real-world experiments on production machines (Hydra, Galileo100, Discoverer)

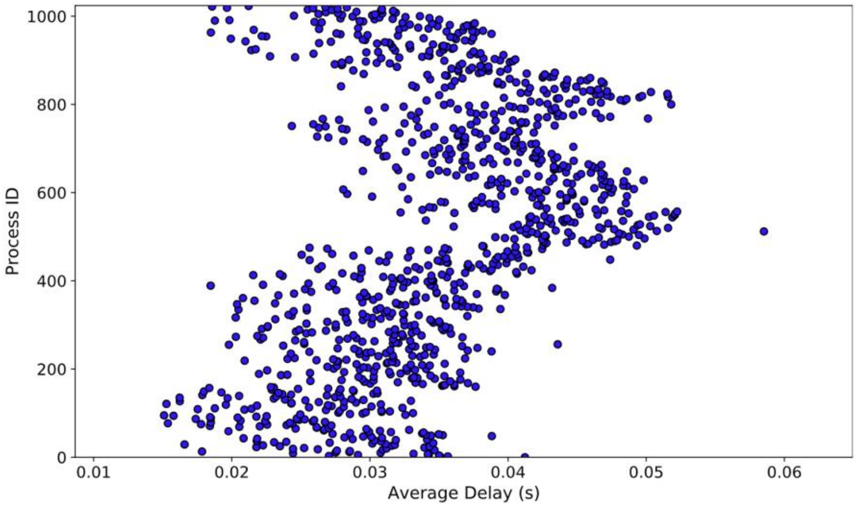
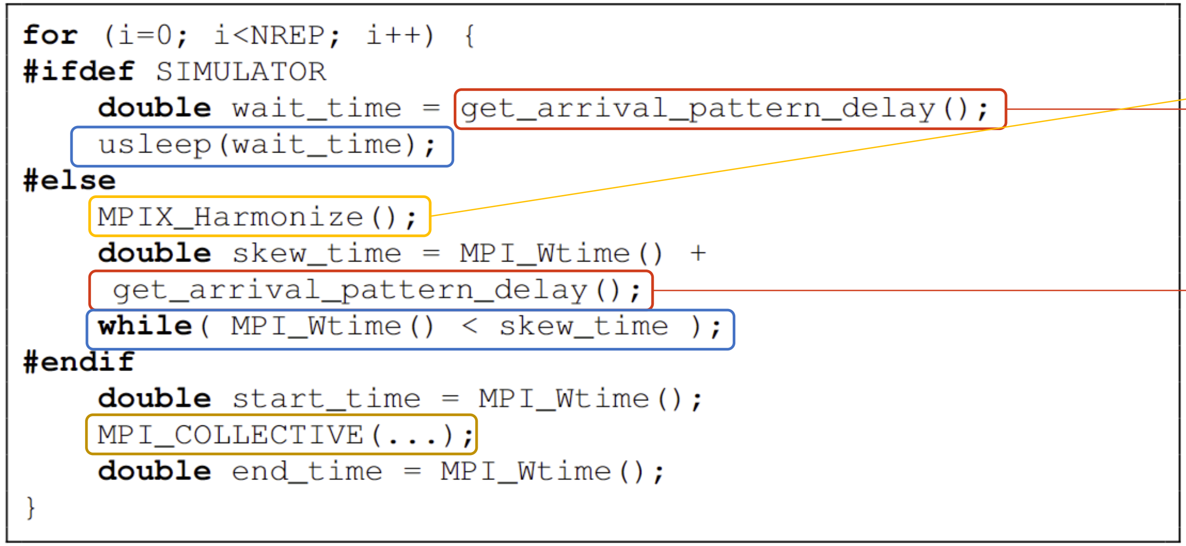


Fig: Avg. process delay (skew) across all MPI_Alltoall calls in FT (NAS parallel benchmarks) on Galileo100 with 32 × 32 processes.

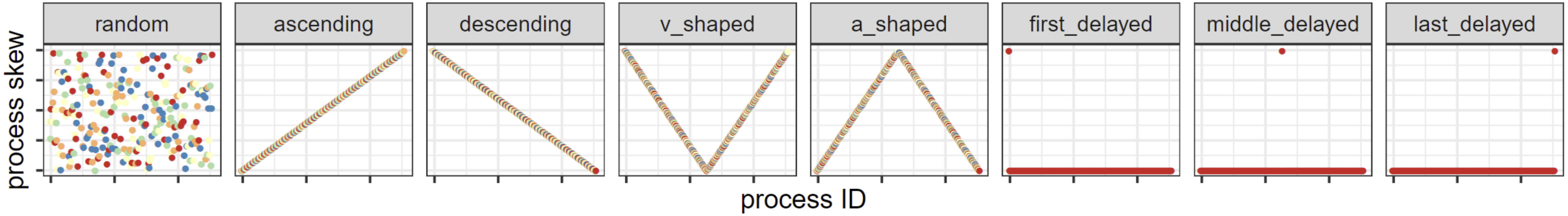
Methodology



```
for (i=0; i<NREP; i++) {
#ifdef SIMULATOR
    double wait_time = get_arrival_pattern_delay();
    usleep(wait_time);
#else
    MPIX_Harmonize();
    double skew_time = MPI_Wtime() +
     get_arrival_pattern_delay();
    while( MPI_Wtime() < skew_time );
#endif
    double start_time = MPI_Wtime();
    MPI_COLLECTIVE(...);
    double end_time = MPI_Wtime();
}
```

Joseph Schuchart, Sascha Hunold, George Bosilca: Synchronizing MPI Processes in Space and Time. EuroMPI 2023: 7:1-7:11

Exposing different algorithms to different (artificial) arrival patterns

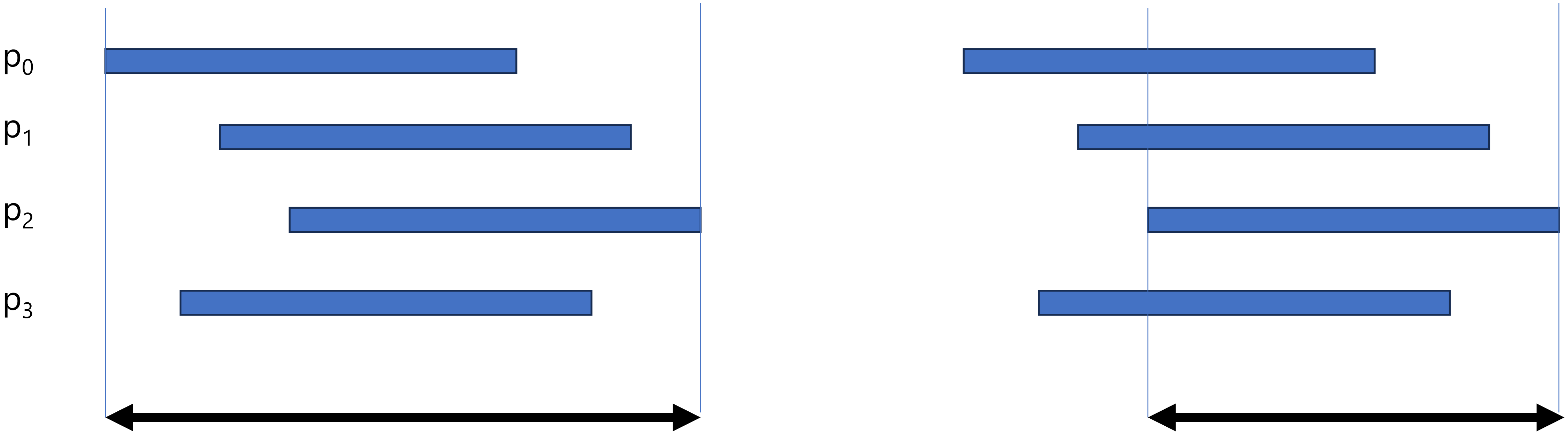


Artificial process arrival patterns

Last Delay Metric

Total Delay
(absolute makespan)

Last Delay
(more meaningful in case of load imbalance)



Since it’s a collective call: it matters most how fast we can complete it when the last process has arrived!

Simulation results

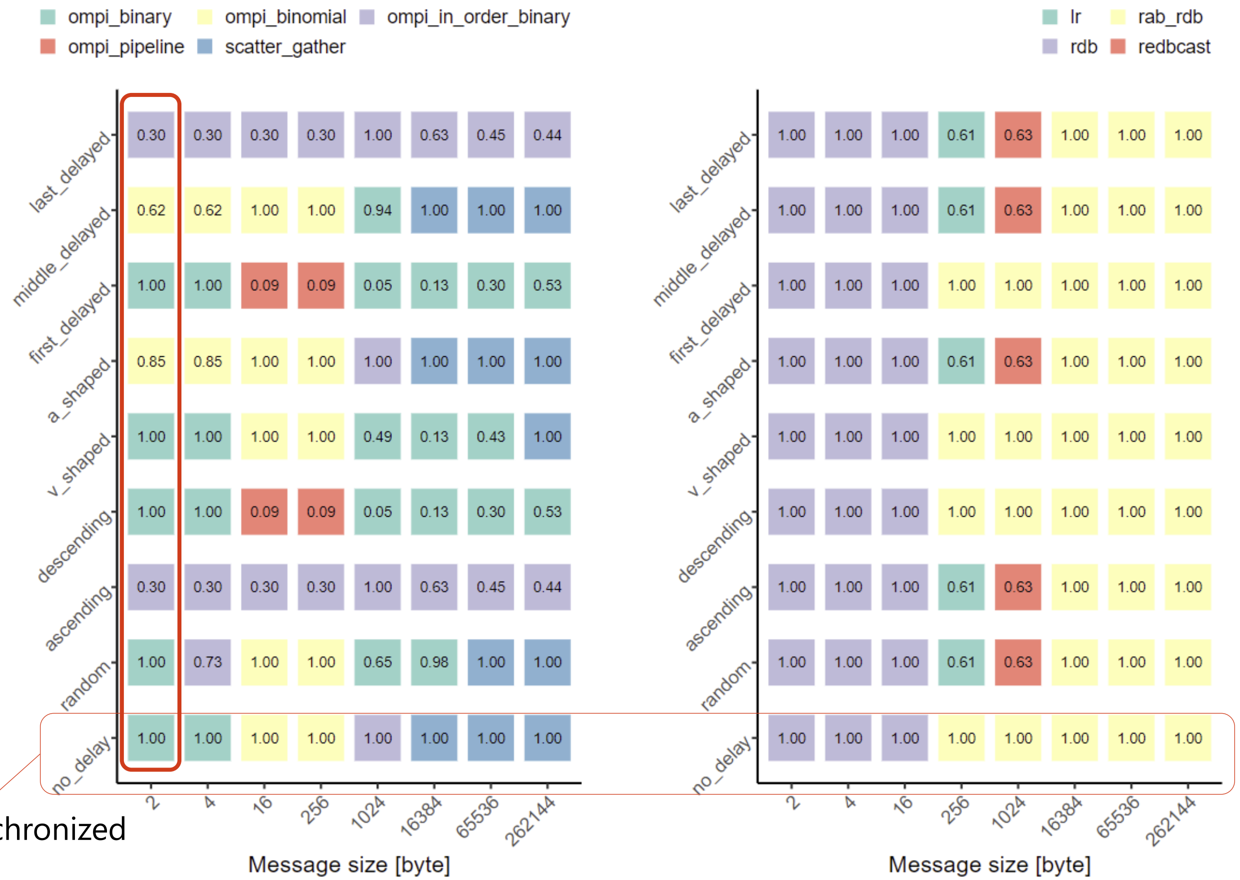
- ❑ 1024 processes (32 x 32)

🟩🟥🟨🟦🟪 The **color:** indicates the best algorithm found for a specific message size

- ❑ The **value**: denotes the relative performance of this algorithm compared to the best algorithm from the no_delay case

- ❑ **MPI_Reduce**
  - ❑ The optimal algorithm for MPI_Reduce varies with different message sizes and process arrival patterns

- ❑ **MPI_Allreduce**
  - ❑ The reduction step in an Allreduce is a strongly synchronizing sub-task



(a) MPI_Reduce

	2	4	16	256	1024	16384	65536	262144
last_delayed	🟪 0.30	🟪 0.30	🟪 0.30	🟪 0.30	🟪 1.00	🟪 0.63	🟪 0.45	🟪 0.44
middle_delayed	🟨 0.62	🟨 0.62	🟨 1.00	🟨 1.00	🟩 0.94	🟦 1.00	🟦 1.00	🟦 1.00
first_delayed	🟩 1.00	🟩 1.00	🟥 0.09	🟥 0.09	🟩 0.05	🟩 0.13	🟩 0.30	🟩 0.53
a_shaped	🟨 0.85	🟨 0.85	🟨 1.00	🟨 1.00	🟪 1.00	🟦 1.00	🟦 1.00	🟦 1.00
v_shaped	🟩 1.00	🟩 1.00	🟨 1.00	🟨 1.00	🟩 0.49	🟩 0.13	🟩 0.43	🟦 1.00
descending	🟩 1.00	🟩 1.00	🟥 0.09	🟥 0.09	🟩 0.05	🟩 0.13	🟩 0.30	🟩 0.53
ascending	🟪 0.30	🟪 0.30	🟪 0.30	🟪 0.30	🟪 1.00	🟪 0.63	🟪 0.45	🟪 0.44
random	🟩 1.00	🟪 0.73	🟨 1.00	🟨 1.00	🟩 0.65	🟩 0.98	🟦 1.00	🟦 1.00
no_delay	🟩 1.00	🟩 1.00	🟨 1.00	🟨 1.00	🟪 1.00	🟦 1.00	🟦 1.00	🟦 1.00

(b) MPI_Allreduce

	2	4	16	256	1024	16384	65536	262144
last_delayed	🟪 1.00	🟪 1.00	🟪 1.00	🟩 0.61	🟥 0.63	🟨 1.00	🟨 1.00	🟨 1.00
middle_delayed	🟪 1.00	🟪 1.00	🟪 1.00	🟩 0.61	🟥 0.63	🟨 1.00	🟨 1.00	🟨 1.00
first_delayed	🟪 1.00	🟪 1.00	🟪 1.00	🟨 1.00	🟨 1.00	🟨 1.00	🟨 1.00	🟨 1.00
a_shaped	🟪 1.00	🟪 1.00	🟪 1.00	🟩 0.61	🟥 0.63	🟨 1.00	🟨 1.00	🟨 1.00
v_shaped	🟪 1.00	🟪 1.00	🟪 1.00	🟨 1.00	🟨 1.00	🟨 1.00	🟨 1.00	🟨 1.00
descending	🟪 1.00	🟪 1.00	🟪 1.00	🟨 1.00	🟨 1.00	🟨 1.00	🟨 1.00	🟨 1.00
ascending	🟪 1.00	🟪 1.00	🟪 1.00	🟩 0.61	🟥 0.63	🟨 1.00	🟨 1.00	🟨 1.00
random	🟪 1.00	🟪 1.00	🟪 1.00	🟩 0.61	🟥 0.63	🟨 1.00	🟨 1.00	🟨 1.00
no_delay	🟪 1.00	🟪 1.00	🟪 1.00	🟨 1.00	🟨 1.00	🟨 1.00	🟨 1.00	🟨 1.00

All processes are perfectly synchronized

Arrival patterns impact the collective algorithms

Hardware Overview

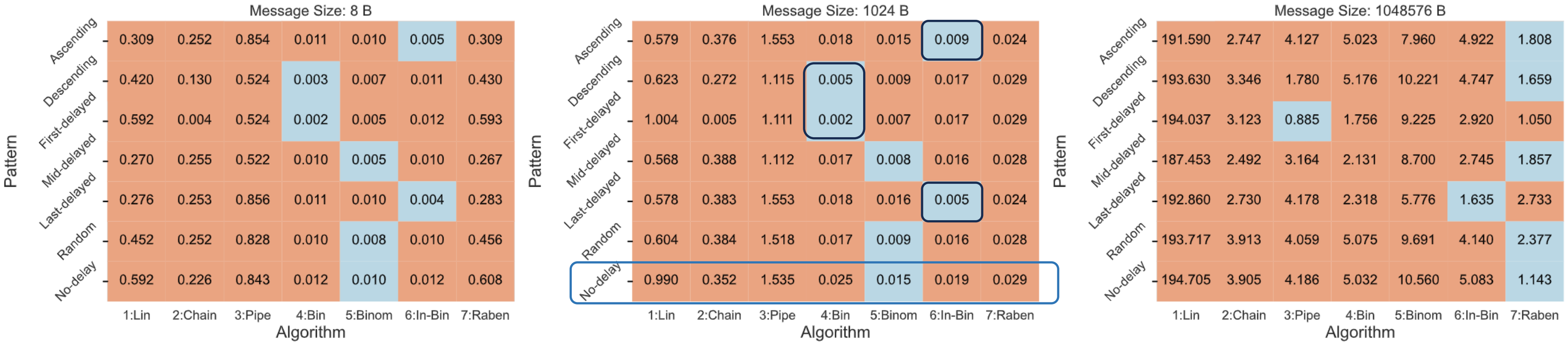
Hydra (TU Wien)	**Galileo 100 (Cineca)**	**Discover (Sofia Tech Park)**
36 nodes, 2x16-core Intel Xeon 2.1GHz	512 nodes, 2x24-core Intel CascadeLake 8260	1128 nodes, 2x64-core AMD Epyc 7H12
Dual-rail Intel **Omni-Path** (100 Gbit/s)	Mellanox **Infiniband** HDR100	**Infiniband** HDR (Dragonfly+)
Open MPI 4.1.5	Open MPI 4.1.1	Open MPI 4.1.4

Real-world Experiments

- ❑ 1024 processes (32 × 32) processes
- ❑ For each arrival pattern, algorithms within 5% of the fastest are in blue
- ❑ Knowing the arrival patterns, we can accurately select the best algorithm
- ❑ Detecting arrival patterns is time-consuming /infeasible in real-world

Key Idea:

Selecting a **robust** algorithm for MPI collectives, capable of performing well when facing various arrival patterns



Message Size: 8 B

Pattern	1:Lin	2:Chain	3:Pipe	4:Bin	5:Binom	6:In-Bin	7:Raben
Ascending	0.309	0.252	0.854	0.011	0.010	0.005	0.309
Descending	0.420	0.130	0.524	0.003	0.007	0.011	0.430
First-delayed	0.592	0.004	0.524	0.002	0.005	0.012	0.593
Mid-delayed	0.270	0.255	0.522	0.010	0.005	0.010	0.267
Last-delayed	0.276	0.253	0.856	0.011	0.010	0.004	0.283
Random	0.452	0.252	0.828	0.010	0.008	0.010	0.456
No-delay	0.592	0.226	0.843	0.012	0.010	0.012	0.608

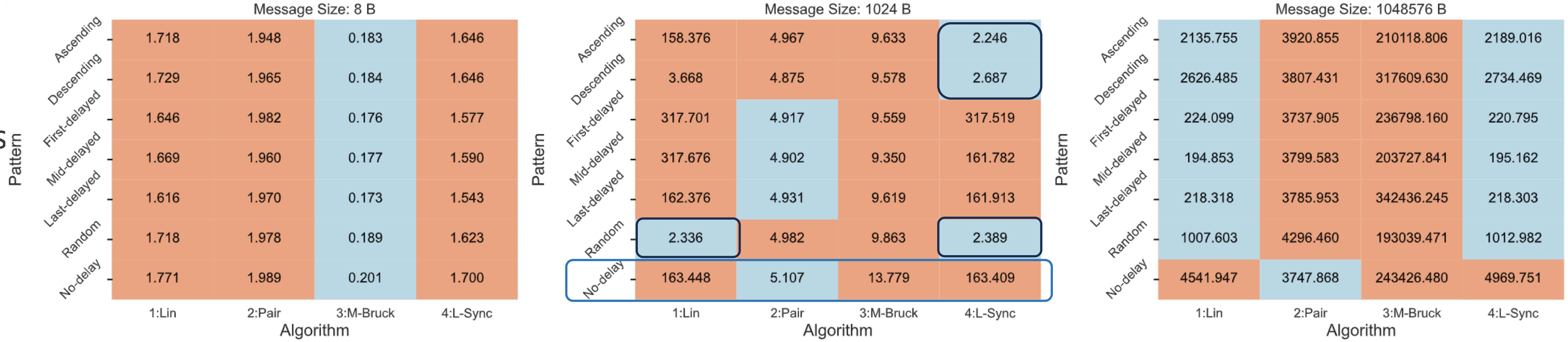
Message Size: 1024 B

Pattern	1:Lin	2:Chain	3:Pipe	4:Bin	5:Binom	6:In-Bin	7:Raben
Ascending	0.579	0.376	1.553	0.018	0.015	0.009	0.024
Descending	0.623	0.272	1.115	0.005	0.009	0.017	0.029
First-delayed	1.004	0.005	1.111	0.002	0.007	0.017	0.029
Mid-delayed	0.568	0.388	1.112	0.017	0.008	0.016	0.028
Last-delayed	0.578	0.383	1.553	0.018	0.016	0.005	0.024
Random	0.604	0.384	1.518	0.017	0.009	0.016	0.028
No-delay	0.990	0.352	1.535	0.025	0.015	0.019	0.029

Message Size: 1048576 B

Pattern	1:Lin	2:Chain	3:Pipe	4:Bin	5:Binom	6:In-Bin	7:Raben
Ascending	191.590	2.747	4.127	5.023	7.960	4.922	1.808
Descending	193.630	3.346	1.780	5.176	10.221	4.747	1.659
First-delayed	194.037	3.123	0.885	1.756	9.225	2.920	1.050
Mid-delayed	187.453	2.492	3.164	2.131	8.700	2.745	1.857
Last-delayed	192.860	2.730	4.178	2.318	5.776	1.635	2.733
Random	193.717	3.913	4.059	5.075	9.691	4.140	2.377
No-delay	194.705	3.905	4.186	5.032	10.560	5.083	1.143

MPI_Reduce



Message Size: 8 B

Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	1.718	1.948	0.183	1.646
Descending	1.729	1.965	0.184	1.646
First-delayed	1.646	1.982	0.176	1.577
Mid-delayed	1.669	1.960	0.177	1.590
Last-delayed	1.616	1.970	0.173	1.543
Random	1.718	1.978	0.189	1.623
No-delay	1.771	1.989	0.201	1.700

Message Size: 1024 B

Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	158.376	4.967	9.633	2.246
Descending	3.668	4.875	9.578	2.687
First-delayed	317.701	4.917	9.559	317.519
Mid-delayed	317.676	4.902	9.350	161.782
Last-delayed	162.376	4.931	9.619	161.913
Random	2.336	4.982	9.863	2.389
No-delay	163.448	5.107	13.779	163.409

Message Size: 1048576 B

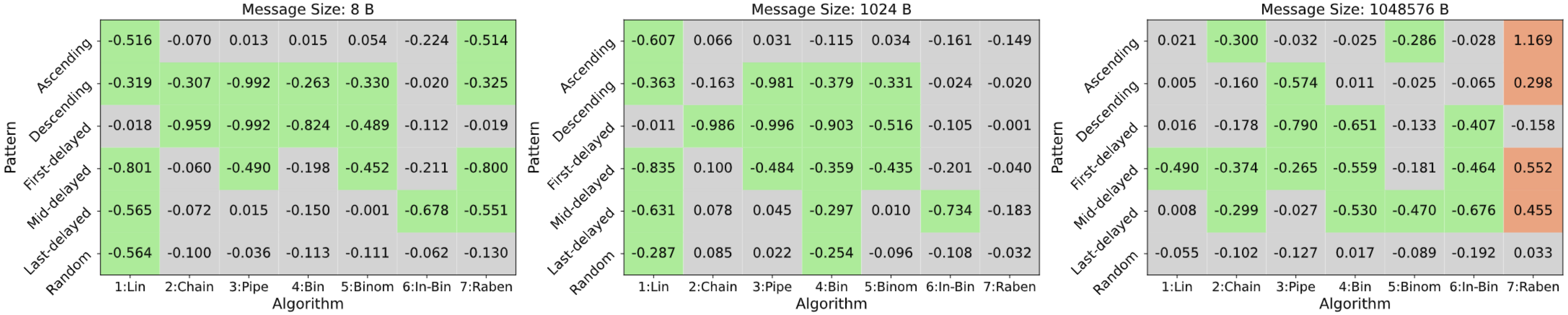
Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	2135.755	3920.855	210118.806	2189.016
Descending	2626.485	3807.431	317609.630	2734.469
First-delayed	224.099	3737.905	236798.160	220.795
Mid-delayed	194.853	3799.583	203727.841	195.162
Last-delayed	218.318	3785.953	342436.245	218.303
Random	1007.603	4296.460	193039.471	1012.982
No-delay	4541.947	3747.868	243426.480	4969.751

MPI_Alltoall

Fig: Runtimes of MPI collectives for various message sizes on Hydra

Real-world Experiments – Robustness

- 1024 processes (32 × 32) processes
- Normalized runtimes to No-delay
- Green rectangles: at least 25% faster than No-delay; Red rectangles: at least 25% slower than No-delay
- For MPI_Reduce: most algorithms are sensitive to process arrival patterns
- **Selection strategy**: Algorithms with more green/grey areas can be good choices



Message Size: 8 B

Pattern	1:Lin	2:Chain	3:Pipe	4:Bin	5:Binom	6:In-Bin	7:Raben
Ascending	-0.516	-0.070	0.013	0.015	0.054	-0.224	-0.514
Descending	-0.319	-0.307	-0.992	-0.263	-0.330	-0.020	-0.325
First-delayed	-0.018	-0.959	-0.992	-0.824	-0.489	-0.112	-0.019
Mid-delayed	-0.801	-0.060	-0.490	-0.198	-0.452	-0.211	-0.800
Last-delayed	-0.565	-0.072	0.015	-0.150	-0.001	-0.678	-0.551
Random	-0.564	-0.100	-0.036	-0.113	-0.111	-0.062	-0.130

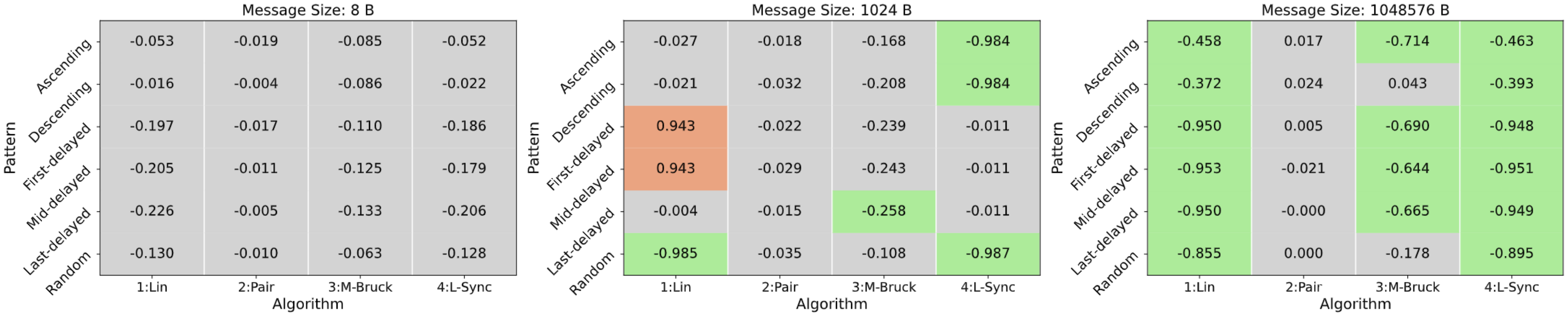
Message Size: 1024 B

Pattern	1:Lin	2:Chain	3:Pipe	4:Bin	5:Binom	6:In-Bin	7:Raben
Ascending	-0.607	0.066	0.031	-0.115	0.034	-0.161	-0.149
Descending	-0.363	-0.163	-0.981	-0.379	-0.331	-0.024	-0.020
First-delayed	-0.011	-0.986	-0.996	-0.903	-0.516	-0.105	-0.001
Mid-delayed	-0.835	0.100	-0.484	-0.359	-0.435	-0.201	-0.040
Last-delayed	-0.631	0.078	0.045	-0.297	0.010	-0.734	-0.183
Random	-0.287	0.085	0.022	-0.254	-0.096	-0.108	-0.032

Message Size: 1048576 B

Pattern	1:Lin	2:Chain	3:Pipe	4:Bin	5:Binom	6:In-Bin	7:Raben
Ascending	0.021	-0.300	-0.032	-0.025	-0.286	-0.028	1.169
Descending	0.005	-0.160	-0.574	0.011	-0.025	-0.065	0.298
First-delayed	0.016	-0.178	-0.790	-0.651	-0.133	-0.407	-0.158
Mid-delayed	-0.490	-0.374	-0.265	-0.559	-0.181	-0.464	0.552
Last-delayed	0.008	-0.299	-0.027	-0.530	-0.470	-0.676	0.455
Random	-0.055	-0.102	-0.127	0.017	-0.089	-0.192	0.033

MPI_Reduce



Message Size: 8 B

Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	-0.053	-0.019	-0.085	-0.052
Descending	-0.016	-0.004	-0.086	-0.022
First-delayed	-0.197	-0.017	-0.110	-0.186
Mid-delayed	-0.205	-0.011	-0.125	-0.179
Last-delayed	-0.226	-0.005	-0.133	-0.206
Random	-0.130	-0.010	-0.063	-0.128

Message Size: 1024 B

Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	-0.027	-0.018	-0.168	-0.984
Descending	-0.021	-0.032	-0.208	-0.984
First-delayed	0.943	-0.022	-0.239	-0.011
Mid-delayed	0.943	-0.029	-0.243	-0.011
Last-delayed	-0.004	-0.015	-0.258	-0.011
Random	-0.985	-0.035	-0.108	-0.987

Message Size: 1048576 B

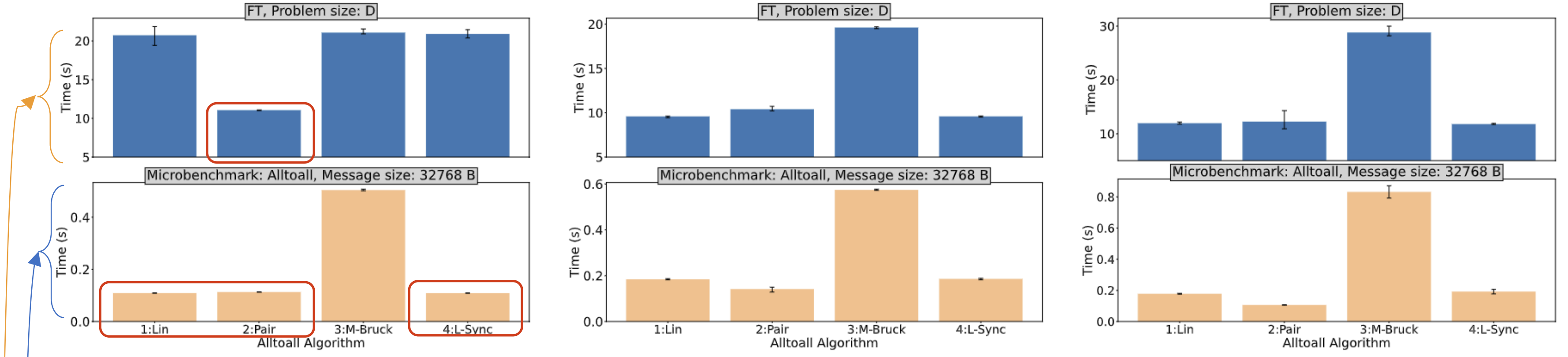
Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	-0.458	0.017	-0.714	-0.463
Descending	-0.372	0.024	0.043	-0.393
First-delayed	-0.950	0.005	-0.690	-0.948
Mid-delayed	-0.953	-0.021	-0.644	-0.951
Last-delayed	-0.950	-0.000	-0.665	-0.949
Random	-0.855	0.000	-0.178	-0.895

MPI_Alltoall

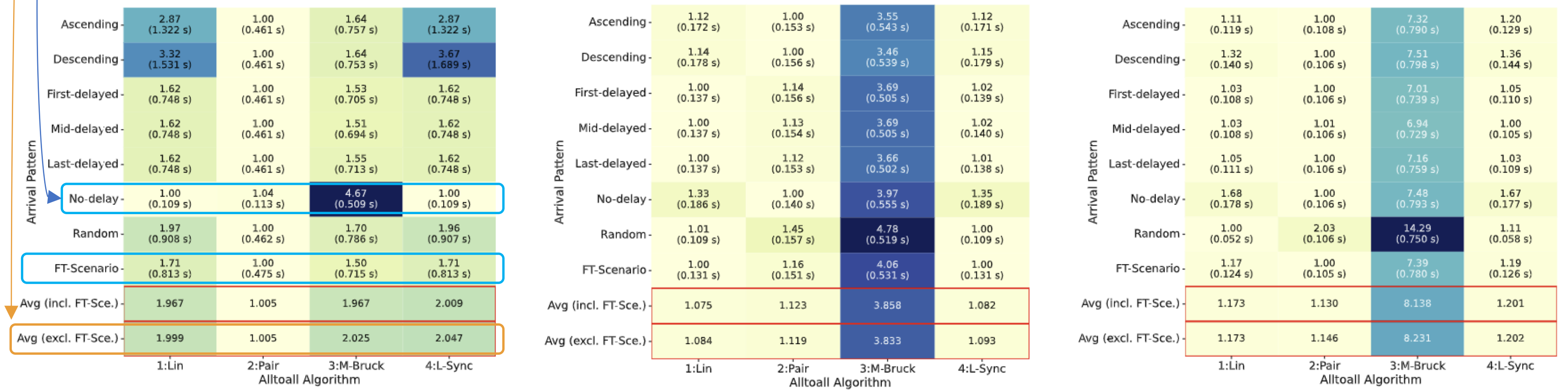
Fig: Normalized runtimes to No-delay case on Hydra

Arrival Patterns in Applications

- ❑ FT (problem size D) from NAS Parallel Benchmark
- ❑ FT-Scenario: Real-world
  - ❑ Enables us to accurately predict the best performing algorithm
- ❑ Selection strategy: average is a good indicator
- ❑ An algorithm that **consistently performs well across multiple arrival patterns** will likely yield satisfactory results across various applications.



(a) *Hydra* (b) *Galileo100* (c) *Discoverer*



(a) *Hydra*

Arrival Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	2.87 (1.322 s)	1.00 (0.461 s)	1.64 (0.757 s)	2.87 (1.322 s)
Descending	3.32 (1.531 s)	1.00 (0.461 s)	1.64 (0.753 s)	3.67 (1.689 s)
First-delayed	1.62 (0.748 s)	1.00 (0.461 s)	1.53 (0.705 s)	1.62 (0.748 s)
Mid-delayed	1.62 (0.748 s)	1.00 (0.461 s)	1.51 (0.694 s)	1.62 (0.748 s)
Last-delayed	1.62 (0.748 s)	1.00 (0.461 s)	1.55 (0.713 s)	1.62 (0.748 s)
No-delay	1.00 (0.109 s)	1.04 (0.113 s)	4.67 (0.509 s)	1.00 (0.109 s)
Random	1.97 (0.908 s)	1.00 (0.462 s)	1.70 (0.786 s)	1.96 (0.907 s)
FT-Scenario	1.71 (0.813 s)	1.00 (0.475 s)	1.50 (0.715 s)	1.71 (0.813 s)
Avg (incl. FT-Sce.)	1.967	1.005	1.967	2.009
Avg (excl. FT-Sce.)	1.999	1.005	2.025	2.047

Alltoall Algorithm

(b) *Galileo100*

Arrival Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	1.12 (0.172 s)	1.00 (0.153 s)	3.55 (0.543 s)	1.12 (0.171 s)
Descending	1.14 (0.178 s)	1.00 (0.156 s)	3.46 (0.539 s)	1.15 (0.179 s)
First-delayed	1.00 (0.137 s)	1.14 (0.156 s)	3.69 (0.505 s)	1.02 (0.139 s)
Mid-delayed	1.00 (0.137 s)	1.13 (0.154 s)	3.69 (0.505 s)	1.02 (0.140 s)
Last-delayed	1.00 (0.137 s)	1.12 (0.153 s)	3.66 (0.502 s)	1.01 (0.138 s)
No-delay	1.33 (0.186 s)	1.00 (0.140 s)	3.97 (0.555 s)	1.35 (0.189 s)
Random	1.01 (0.109 s)	1.45 (0.157 s)	4.78 (0.519 s)	1.00 (0.109 s)
FT-Scenario	1.00 (0.131 s)	1.16 (0.151 s)	4.06 (0.531 s)	1.00 (0.131 s)
Avg (incl. FT-Sce.)	1.075	1.123	3.858	1.082
Avg (excl. FT-Sce.)	1.084	1.119	3.833	1.093

Alltoall Algorithm

(c) *Discoverer*

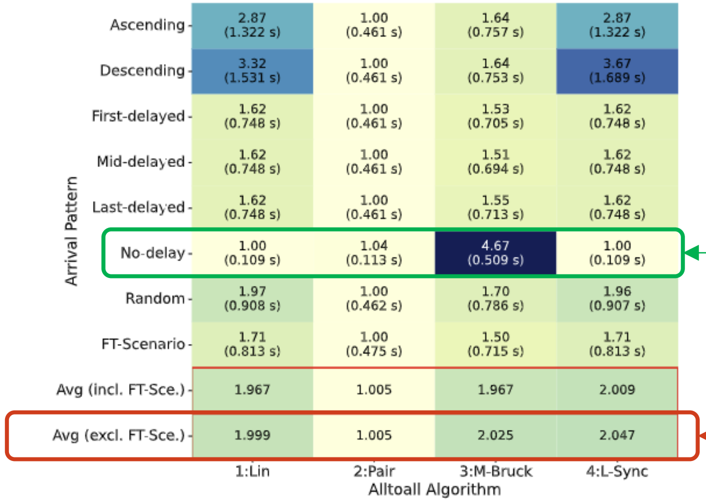
Arrival Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	1.11 (0.119 s)	1.00 (0.108 s)	7.32 (0.790 s)	1.20 (0.129 s)
Descending	1.32 (0.140 s)	1.00 (0.106 s)	7.51 (0.798 s)	1.36 (0.144 s)
First-delayed	1.03 (0.108 s)	1.00 (0.106 s)	7.01 (0.739 s)	1.05 (0.110 s)
Mid-delayed	1.03 (0.108 s)	1.01 (0.106 s)	6.94 (0.729 s)	1.00 (0.105 s)
Last-delayed	1.05 (0.111 s)	1.00 (0.106 s)	7.16 (0.759 s)	1.03 (0.109 s)
No-delay	1.68 (0.178 s)	1.00 (0.106 s)	7.48 (0.793 s)	1.67 (0.177 s)
Random	1.00 (0.052 s)	2.03 (0.106 s)	14.29 (0.750 s)	1.11 (0.058 s)
FT-Scenario	1.17 (0.124 s)	1.00 (0.105 s)	7.39 (0.780 s)	1.19 (0.126 s)
Avg (incl. FT-Sce.)	1.173	1.130	8.138	1.201
Avg (excl. FT-Sce.)	1.173	1.146	8.231	1.202

Alltoall Algorithm

(a) *Hydra* (b) *Galileo100* (c) *Discoverer*

Arrival Patterns in Applications (Cont'd)

- ❑ Expected FT Runtime, based on the **No-delay case**, does not align with the Actual FT Runtime.
- ❑ Expected FT Runtime, based on the **Average case**, aligns well with the Actual FT Runtime.
- ❑ The behavior of the collective algorithm in the application can be predicted!



Arrival Pattern	1:Lin	2:Pair	3:M-Bruck	4:L-Sync
Ascending	2.87 (1.322 s)	1.00 (0.461 s)	1.64 (0.757 s)	2.87 (1.322 s)
Descending	3.32 (1.531 s)	1.00 (0.461 s)	1.64 (0.753 s)	3.67 (1.689 s)
First-delayed	1.62 (0.748 s)	1.00 (0.461 s)	1.53 (0.705 s)	1.62 (0.748 s)
Mid-delayed	1.62 (0.748 s)	1.00 (0.461 s)	1.51 (0.694 s)	1.62 (0.748 s)
Last-delayed	1.62 (0.748 s)	1.00 (0.461 s)	1.55 (0.713 s)	1.62 (0.748 s)
No-delay	1.00 (0.109 s)	1.04 (0.113 s)	4.67 (0.509 s)	1.00 (0.109 s)
Random	1.97 (0.908 s)	1.00 (0.462 s)	1.70 (0.786 s)	1.96 (0.907 s)
FT-Scenario	1.71 (0.813 s)	1.00 (0.475 s)	1.50 (0.715 s)	1.71 (0.813 s)
Avg (incl. FT-Sce.)	1.967	1.005	1.967	2.009
Avg (excl. FT-Sce.)	1.999	1.005	2.025	2.047

(a) *Hydra*

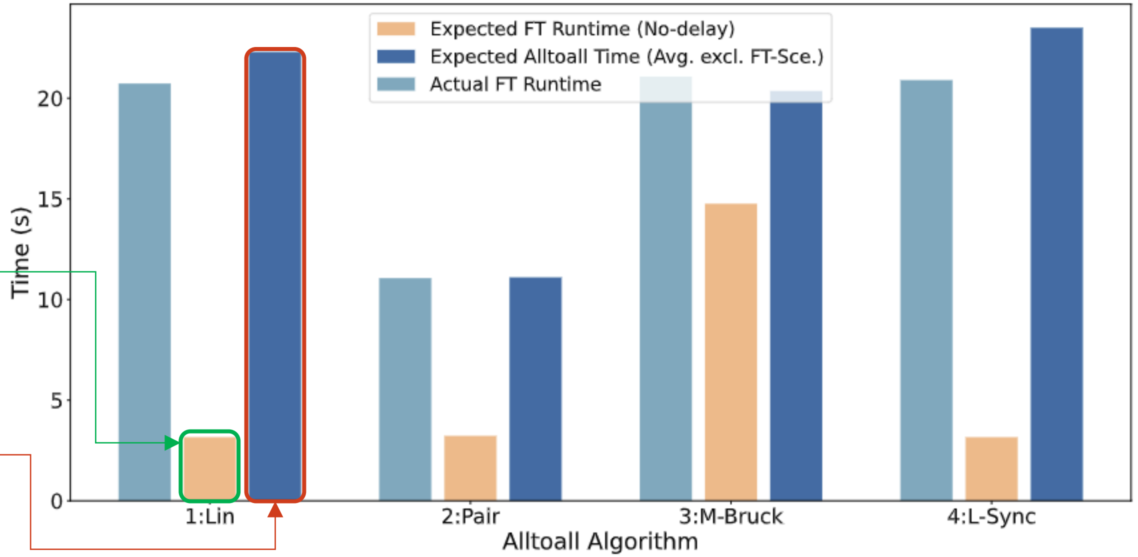


Fig: The actual runtime of FT versus its projected runtimes (when processes enter collectives simultaneously, the No-delay case, and the average case) on Hydra with 32 × 32 processes.

Conclusion and Future Work

- MPI collective algorithm selection problem
- Impact of arrival patterns on collective algorithms
- Micro-benchmarking strategy
  - Simulation study
  - 3 real-world production machines
- * Rooted collectives, such as MPI_Reduce, are more influenced
- * Algorithm selection without considering the process imbalance may lead to an inefficient choice
- * Considering robustness
- Future Directions
  - Studying more complicated applications
  - Studying arrival patterns on GPU clusters

MPI Collective Algorithm Selection
in the Presence of Process Arrival Patterns

Majid Salimi Beni, Biagio Cosenza, Sascha Hunold

2024 IEEE International Conference on Cluster Computing (CLUSTER)

Kobe, Japan
September 24-27, 2024

✉ Reach me at:
msalimibeni@unisa.it