What Modern Distributed Runtimes Add Beyond MPI implementations (like MPICH, OpenMPI, Intel MPI, etc.)

Feature	Why It Matters	Used In distributed runtimes like
Elastic jobs	Restart failed workers, scale up/down	TorchElastic, Ray, TorchTitan
High-level APIs	train() , map(), spawn_actor() instead of MPI_Send ( )	Ray, **PyTorch** DDP, TF
Dynamic scheduling / DAGs	Assign tasks on-demand, based on load	Dask, Ray
Orchestration tools	Launch workers across clusters with minimal config	Monarch, TorchX
GPU-aware backends	**Use NCCL/OneCCL for GPU fast paths**	**PyTorch, DeepSpeed, Horovod**
Cloud & K8s native	Pods, service discovery, autoscaling	Ray, TensorFlow, TorchElastic
Multirole workflows	E.g. actor/learner/logger, not just "ranks"	TorchTitan, Ray RLLib

More focus on pytorch – even though we support many runtimes

Key Orchestration Models in distributed runtimes

Orchestrator Type	Description	Examples
Decentralized (MPI)	Each process started independently; rank configured manually	MPI, **PyTorch DDP**, Horovod
Single Controller	One driver spawns/monitors all remote workers	Monarch, TorchTitan, Ray
Elastic Controller	Launches dynamic workers, restarts failed ones	TorchElastic, Torchx
Central Scheduler	Maintains task queues, dependencies, assigns tasks dynamically	Ray, Dask, Spark
Launcher-Integrated	Orchestration delegated to batch systems like SLURM or Kubernetes	Horovod (on SLURM), TorchElastic
Compiler-based	Program is compiled with parallelism injected by the compiler/runtime	XLA (JAX), Alpa, SYCL

Here "Decentralized (MPI)" means processes don't rely on a central launcher.
Each process is launched independently (e.g., via `mpirun, mpiexec`), and they must be told how to communicate `(eg. MPI.COMM_WORLD)` with the others using environment variables or configs.

Collective Communication Libraries (CCLs)

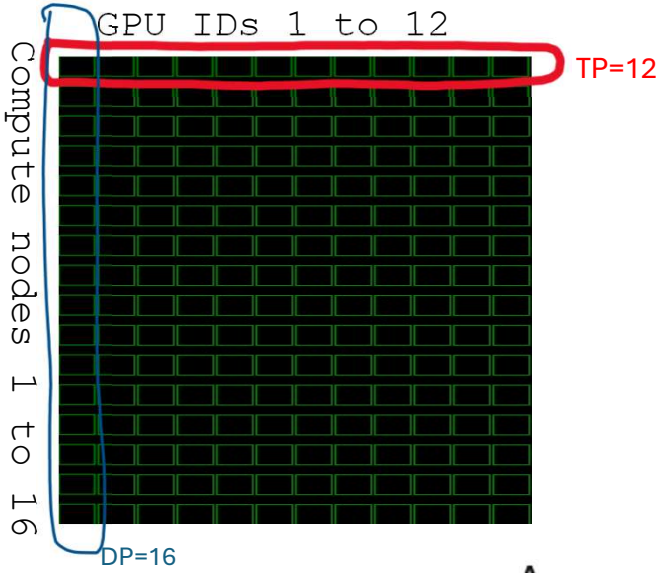
```
distributed.init_process_group( backend = ”xccl",
                                init_method = 'env://',
                                world_size = mpi_world_size,
                                rank = mpi_my_rank,
                                 timeout = datetime.timedelta(seconds=3600))
```

Intel – XCCL backend [Aurora]
Nvidia – NCCL backend [Polaris]
AMD – RCCL backend [Frontier]
Mpi – “mpi” backend - not recommended – need XPU support.

Mpi here means mpi collective communications

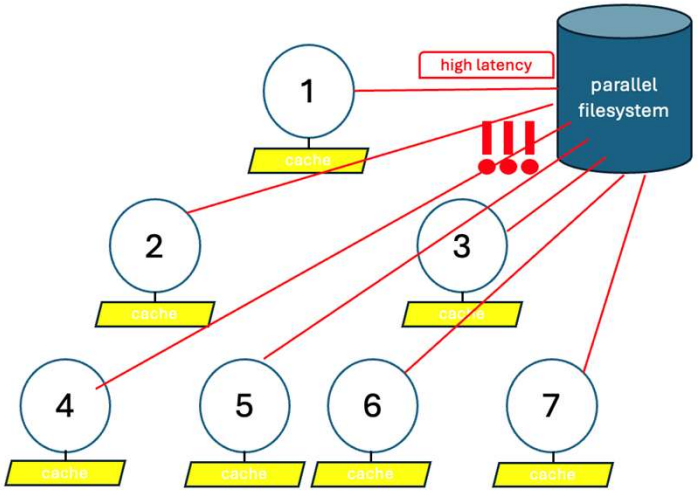
Example

```
torch.distributed.new_group(ranks)
distributed.all_reduce(x, op=dist.ReduceOp.SUM)

TP=12 is only within nodes – intra node communication
DP=16 is only across nodes.- inter node communication
```

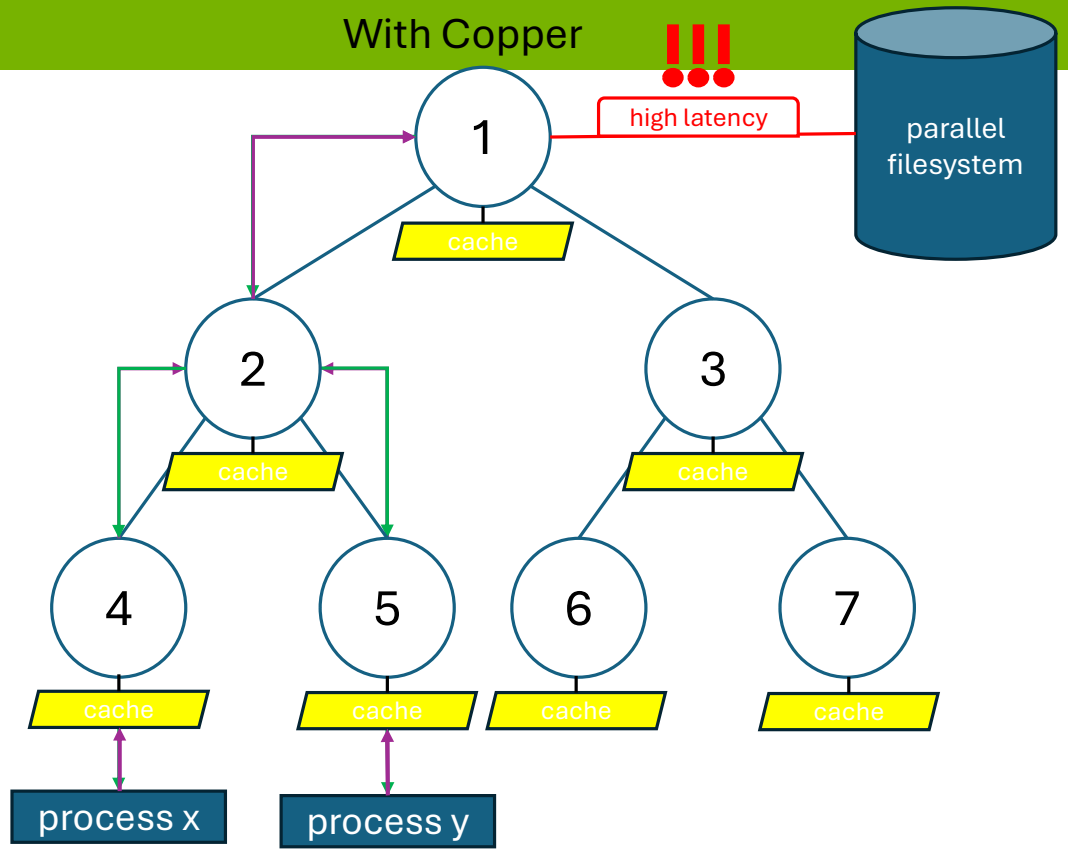


Copper – Data mover

Without Copper



With Copper

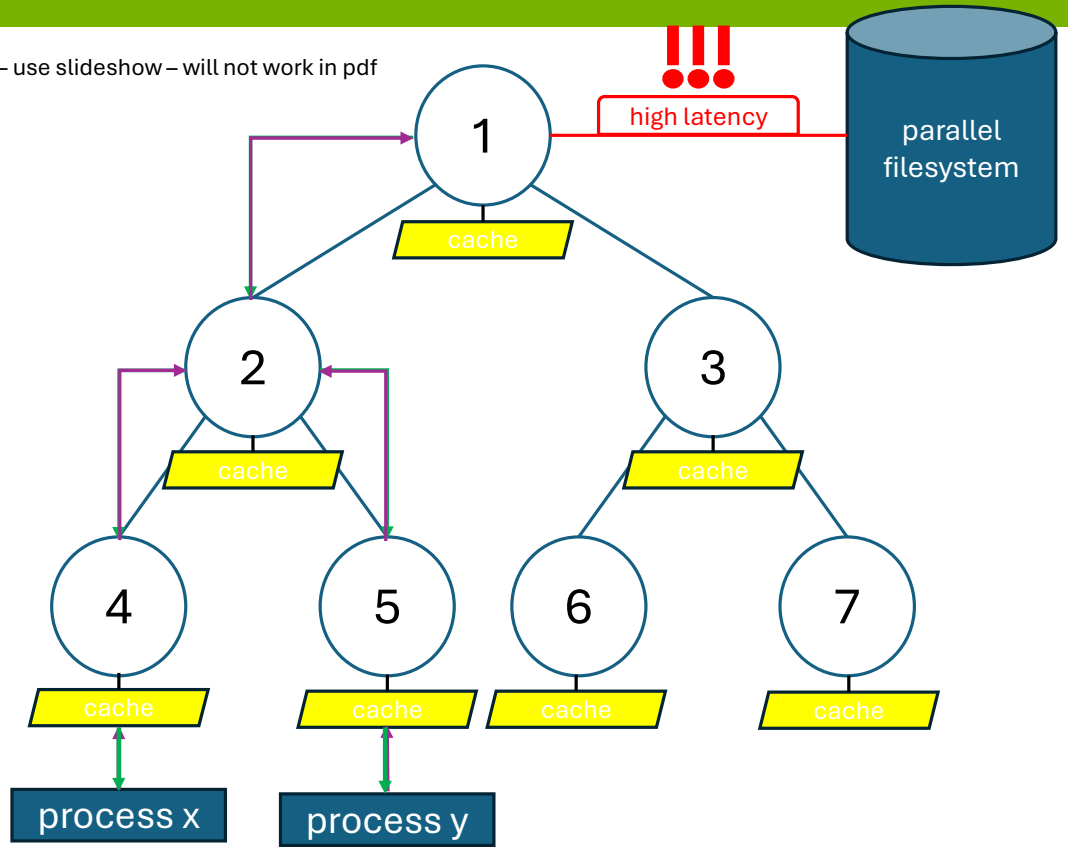


Animation here – use slideshow – will not work in PDF

Copper – Data mover Additional optimizations

num nodes	num ranks per node	without copper import torch time from lustre	with copper import torch time
64	12	105 seconds	50 seconds
128	12	110 seconds	52 seconds
256	12	110 seconds	54 seconds
512	12	115 seconds	56 seconds
1024	12	4 minutes	58 seconds
2048	12	7 minutes	58 seconds
4096	12	8 minutes	59 seconds
8192	12	17 minutes	60 seconds

Animation here – use slideshow – will not work in pdf



Copper Example

Without copper **=/lus/flare/**projects
With Copper **=/tmp/${USER}/copper/**lus/flare/projects

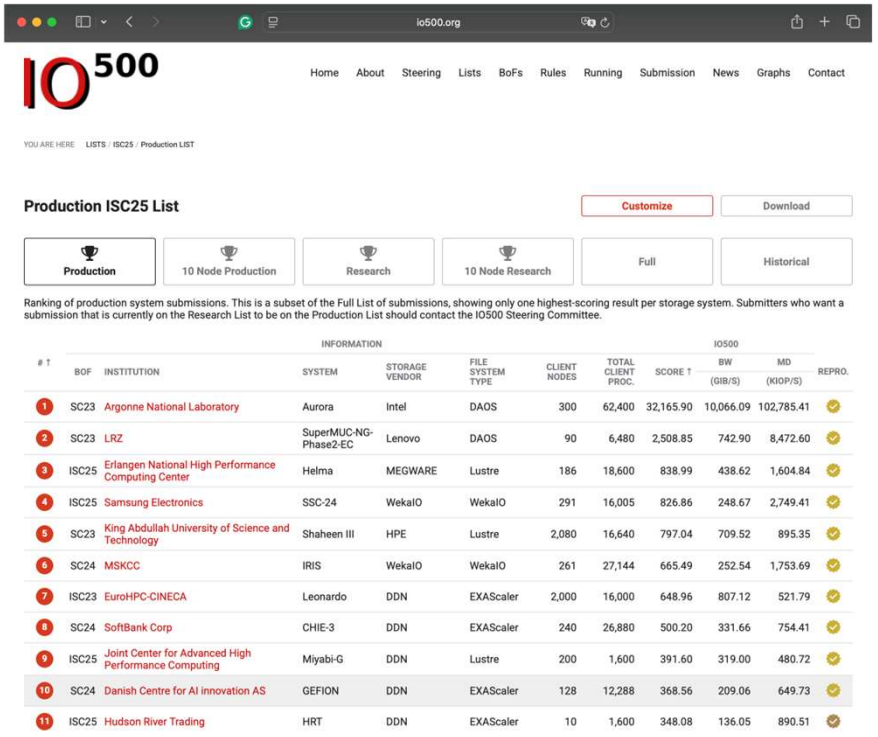
```
module load frameworks
module load copper
launch_copper.sh

# python -m pip install  --target=/lus/flare/projects/datascience/kaushik/copper-test/lus_custom_pip_env/ dragonhpc



time mpirun --np ${NRANKS} --ppn ${RANKS_PER_NODE} \
            --cpu-bind=list:4:56:9:61:14:66:19:71:20:74:25:79 --genvall \
            --genv=PYTHONPATH=/tmp/${USER}/copper/lus/flare/projects/datascience/kaushik/copper-
test/lus_custom_pip_env/:$PYTHONPATH \
             python3 -c "import dragon; print(dragon.__file__)"



stop_copper.sh # optional - enabled by default on the PBS epilog during cleanup.
```

Example and Exercise : https://github.com/argonne-lcf/copper/tree/main/example

DAOS – object Stores

Thursday — August 7, 2025 - Track 7 — I/O Day – DAOS session

- DAOS is a major file system in Aurora
- 1024 DAOS Nodes, 230 PB, >**25 TB/s** **(Lustre 0.6 TB/s)**
- Open-source software-defined **Distributed Asynchrnous Object Store (DAOS)**
- Designed for massively **distributed** Non Volatile **Memory** (NVM) and NVMe **SSD**
- Store your datasets/ python packages in daos container.
- Delivers high performance data loading throughput when training at scale (~60K GPUs)



#	BOF	INSTITUTION	SYSTEM	STORAGE VENDOR	FILE SYSTEM TYPE	CLIENT NODES	TOTAL CLIENT PROC.	SCORE	IO500 BW (GIB/S)	IO500 MD (KIOP/S)	REPRO.
1	SC23	Argonne National Laboratory	Aurora	Intel	DAOS	300	62,400	32,165.90	10,066.09	102,785.41	
2	SC23	LRZ	SuperMUC-NG-Phase2-EC	Lenovo	DAOS	90	6,480	2,508.85	742.90	8,472.60	
3	ISC25	Erlangen National High Performance Computing Center	Helma	MEGWARE	Lustre	186	18,600	838.99	438.62	1,604.84	
4	ISC25	Samsung Electronics	SSC-24	WekaIO	WekaIO	291	16,005	826.86	248.67	2,749.41	
5	SC23	King Abdullah University of Science and Technology	Shaheen III	HPE	Lustre	2,080	16,640	797.04	709.52	895.35	
6	SC24	MSKCC	IRIS	WekaIO	WekaIO	261	27,144	665.49	252.54	1,753.69	
7	ISC23	EuroHPC-CINECA	Leonardo	DDN	EXAScaler	2,000	16,000	648.96	807.12	521.79	
8	SC24	SoftBank Corp	CHIE-3	DDN	EXAScaler	240	26,880	500.20	331.66	754.41	
9	ISC25	Joint Center for Advanced High Performance Computing	Miyabi-G	DDN	Lustre	200	1,600	391.60	319.00	480.72	
10	SC24	Danish Centre for AI innovation AS	GEFION	DDN	EXAScaler	128	12,288	368.56	209.06	649.73	
11	ISC25	Hudson River Trading	HRT	DDN	EXAScaler	10	1,600	348.08	136.05	890.51	

CCL Exercise

https://github.com/argonne-lcf/DLcomm_benchmark/tree/master/tools/examples_dl_scaling

within.py
across.py
combined.py
k_all.py

```
module load frameworks
echo $CCL_ROOT

export CCL_PROCESS_LAUNCHER=pmix
export CCL_ATL_TRANSPORT=mpi
export CCL_KVS_MODE=mpi
export CCL_CONFIGURATION_PATH=""
export CCL_CONFIGURATION=cpu_gpu_dpcpp
export CCL_KVS_CONNECTION_TIMEOUT=600
export MPI_PROVIDER=$FI_PROVIDER
```

Algorithm selection

```
export CCL_COLLECTIVENAME=topo # SCALE UP ALGO
export CCL_COLLECTIVENAME_SCALEOUT=ALGORITHM_NAME # SCALE OUT ALGO

export CCL_ALLREDUCE=topo
export CCL_ALLREDUCE_SCALEOUT=rabenseifner # or ring
```

https://uxlfoundation.github.io/oneCCL/env-variables.html#ccl-allgather-ccl-allgatherv

Recommended NIC binding for 12 PPN (12 GPUs and 8NICS)

```
CPU_BINDING1=list:4:9:14:19:20:25:56:61:66:71:74:79
```

NIC 0	NIC 1	NIC 2	NIC 3	NIC 4	NIC 5	NIC 6	NIC 7
★	1	2	3	★	53	54	55
4	5	6	7	**56**	57	58	59
8	**9**	10	11	60	**61**	62	63
12	13	**14**	15	64	65	**66**	67
16	17	18	**19**	68	69	70	**71**
20	21	22	23	72	73	**74**	75
24	**25**	26	27	76	77	78	**79**
28	29	30	31	80	81	82	83
32	33	34	35	84	85	86	87
36	37	38	39	88	89	90	91
40	41	42	43	92	93	94	95
44	45	46	47	96	97	98	99
48	49	50	51	100	101	102	103

Binding examples

```
# 12 ppn to 12 cores
export CPU_BINDING1="list:4:9:14:19:20:25:56:61:66:71:74:79”


# 12 ppn with each rank having 4 cores
export CPU_BINDING2="list:4-7:8-11:12-15:16-19:20-23:24-27:56-59:60-63:64-67:68-71:72-75:76-79"


export CCL_WORKER_AFFINITY=42,43,44,45,46,47,94,95,96,97,98,99
## Optional cores for oneCCL workers - default picks last 24 cores.


mpiexec --env CCL_LOG_LEVEL=info  --env CCL_PROCESS_LAUNCHER=pmix  --env CCL_ATL_TRANSPORT=mpi \
        --np ${NRANKS} -ppn ${RANKS_PER_NODE} --cpu-bind  $CPU_BINDING1     $APP1
```

Note : when copper is used . copper runs on [Default: "48-51"] cores - can be changed