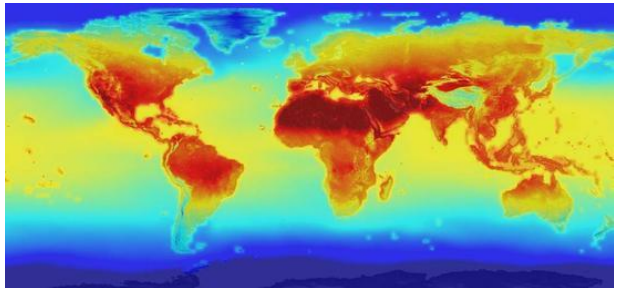
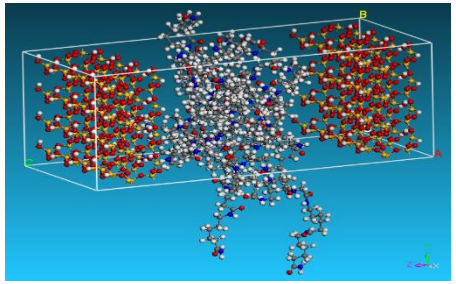
Automatic Application-Aware Forwarding Resource Allocation

Xu Ji, Bin Yang, Tianyu Zhang, Xiaosong Ma, Xiupeng Zhu, Xiyang Wang, Nosayba El-sayed, Jidong Zhai, Weiguo Liu, **Wei Xue**

Storage Crucial for Supercomputing

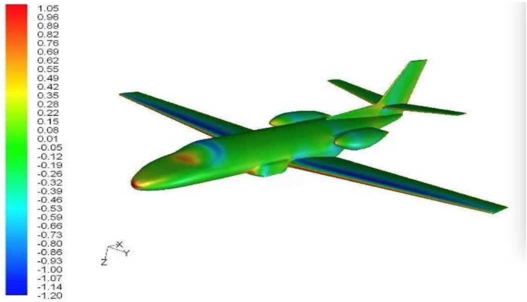


Climate simulation



Molecular dynamics

Sunway TaihuLight

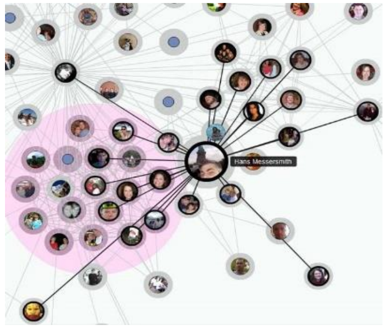


Fluid dynamics

Summit

Sierra

Bioscience



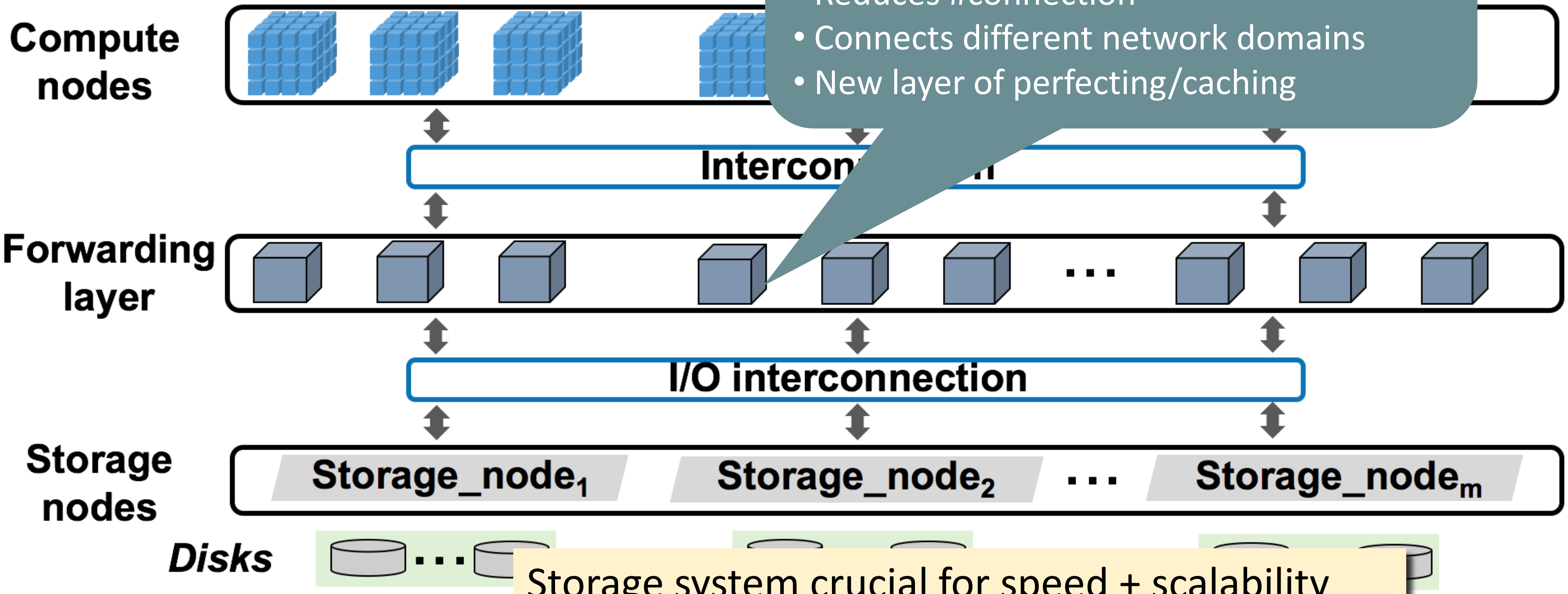
Graph computing

Neural network

I/O Forwarding Layer

I/O Forwarding architecture

- Enables lean OS on compute nodes
- Reduces #connection
- Connects different network domains
- New layer of perfecting/caching

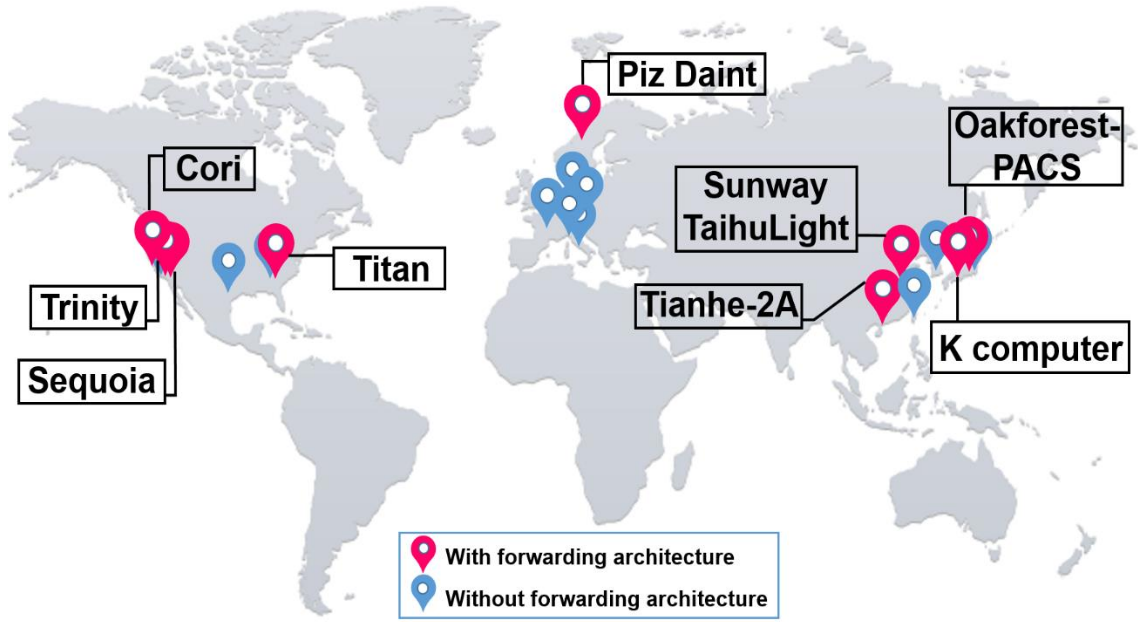


Storage system crucial for speed + scalability

- Performance bottleneck
- Contention point => inter-job performance interference

I/O Forwarding Widely Adopted

- 9 out of top 20 supercomputers use I/O forwarding (Nov 2018)



Rank	Machine	Vendor	# Compute nodes	# Forwarding nodes	File system
3	TaihuLight	NRCPC	40,960	240	Lustre
4	Tianhe-2A	NUDT	16,000	256	Lustre+H2FS
5	Piz Daint	Cray	5,320	54	Lustre+GPFS
6	Trinity	Cray	14,436	576	Lustre
9	Titan	Cray	18,688	432	Lustre
10	Sequoia	BlueGene	98,304	768	Lustre
12	Cori	Cray	12,076	130	Lustre+GPFS
14	Oakforest-PACS	Fujitsu	8,208	50	Lustre
18	K computer	Fujitsu	82,944	5,184	FEFS

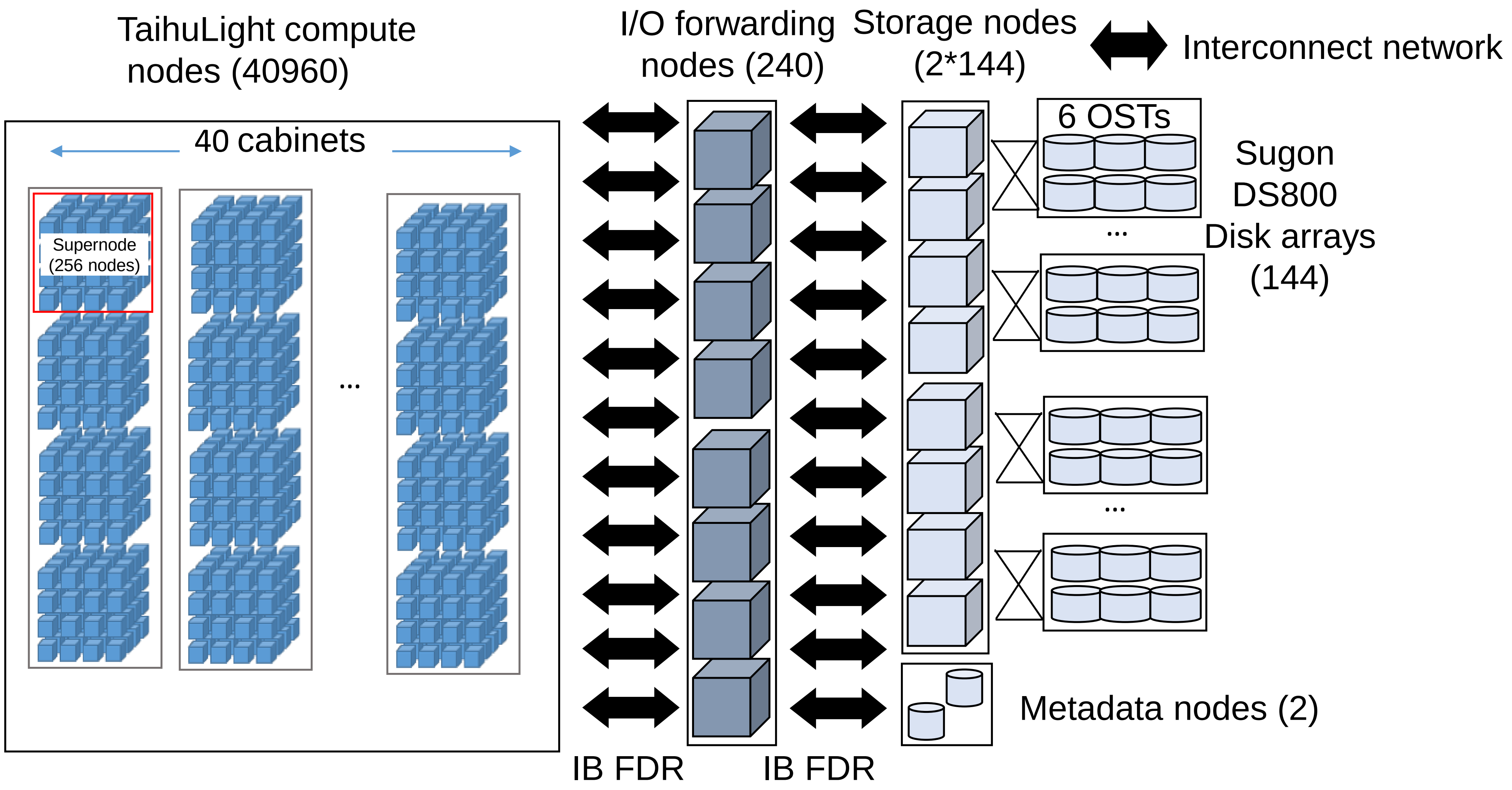
The ratio between compute and forwarding nodes range between 170 to 1 and 16 to 1

Challenges with I/O Forwarding

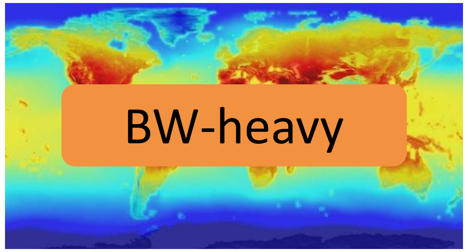
- Forwarding resource ***provisioning***
  - How many forwarding nodes and how much forwarding capacity?
    - Need to consider application I/O demands, machine utilization, budget constraint
    - Design and procurement often finish years before application test runs
- Forwarding resource ***management***
  - How many forwarding nodes to allocate to each job?
  - Which forwarding nodes to assign to each job?

- Current common practice: ***Fixed Forwarding Mapping (FFM)***
  - Static compute-to-forwarding node mapping

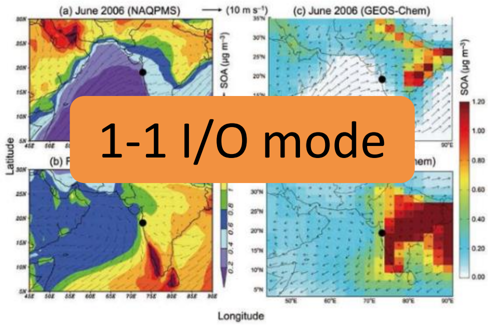
Sample Forwarding Configuration: TaihuLight



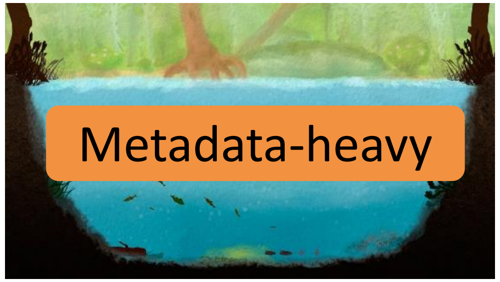
FFM Problem 1: Resource Misallocation



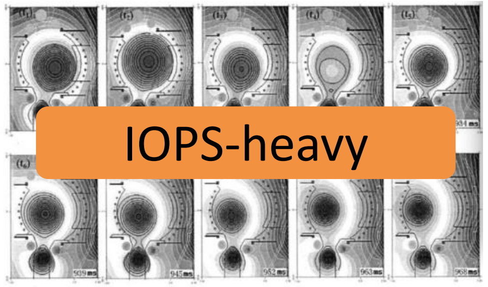
CESM climate simulator



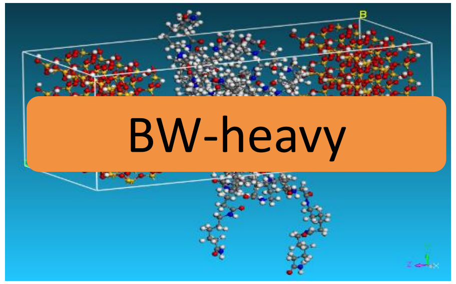
CAM (2017 GB finalist) atmospheric model



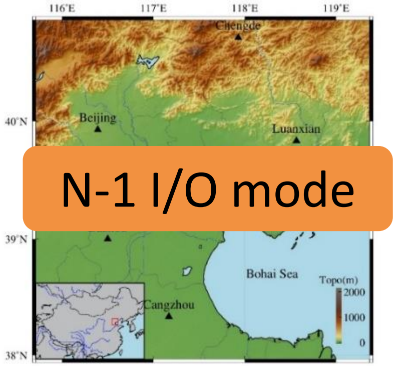
DNDC agro-ecosystems



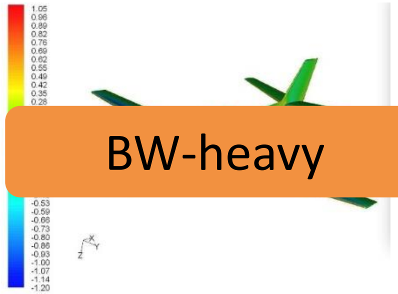
APT particle dynamics



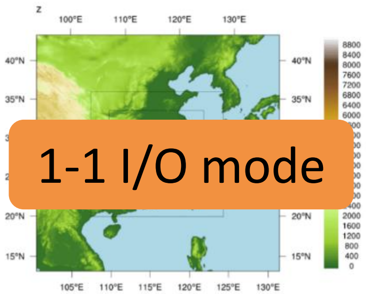
LAMMPS molecular dynamics



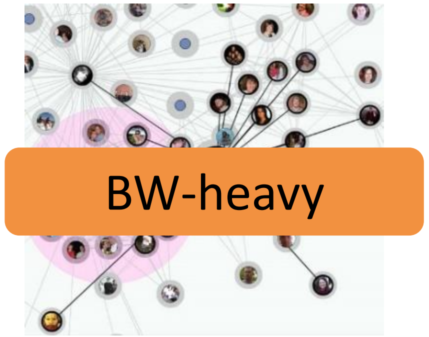
AWP (2017 GB prize) earthquake simulation



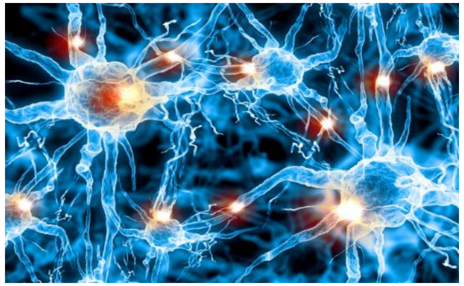
XCFD fluid dynamics



WRF weather forecasting



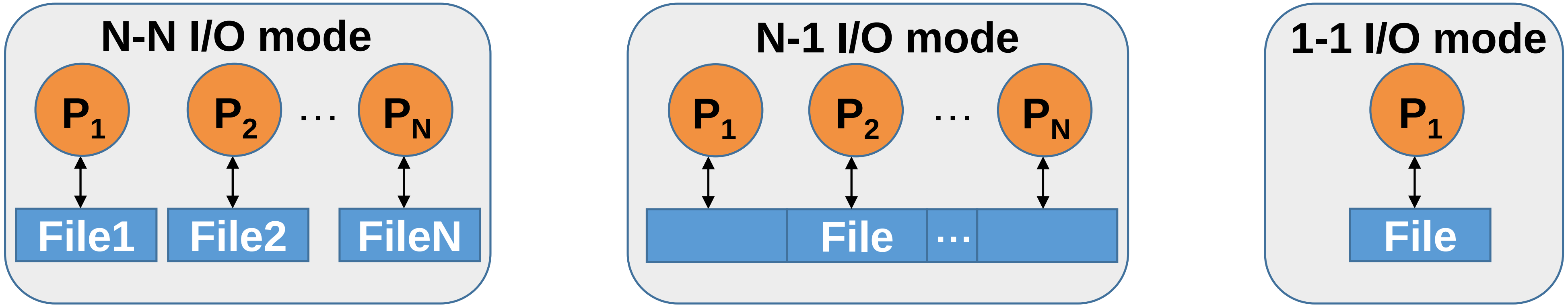
Shentu (2018 GB finalist) graph engine

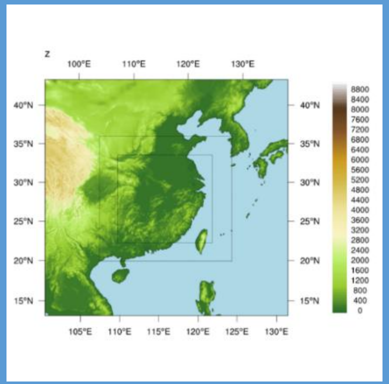


swDNN neuronal network

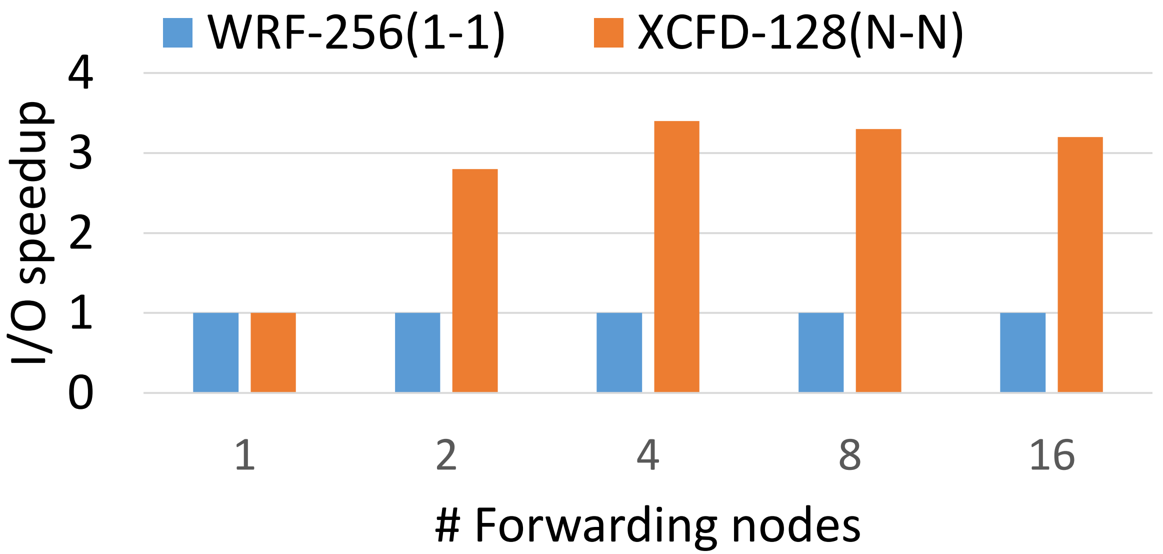
Apps have different I/O behaviors!

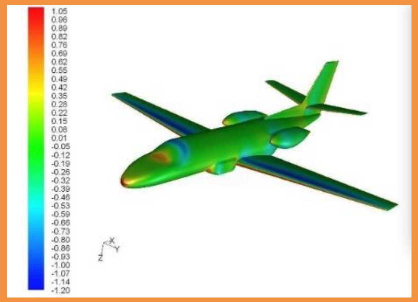
FFM Problem 1: Resource Misallocation





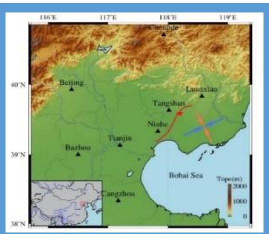
WRF weather forecasting





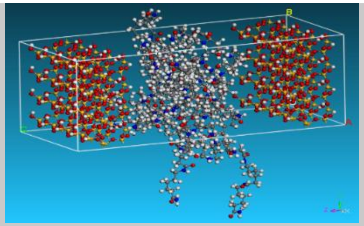
XCFD fluid dynamics

FFM Problem 2: I/O Interference



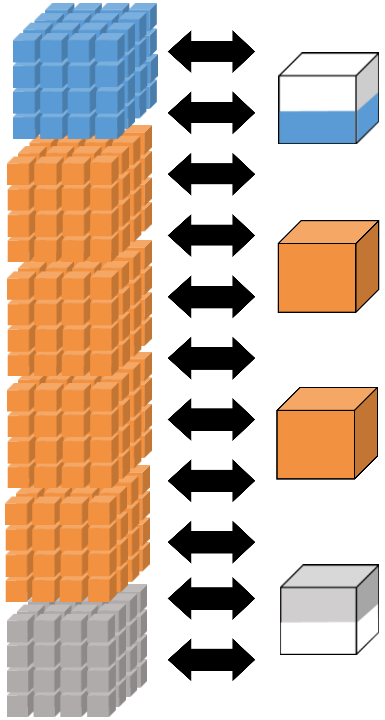
AWP earthquake simulation

Shentu graph engine

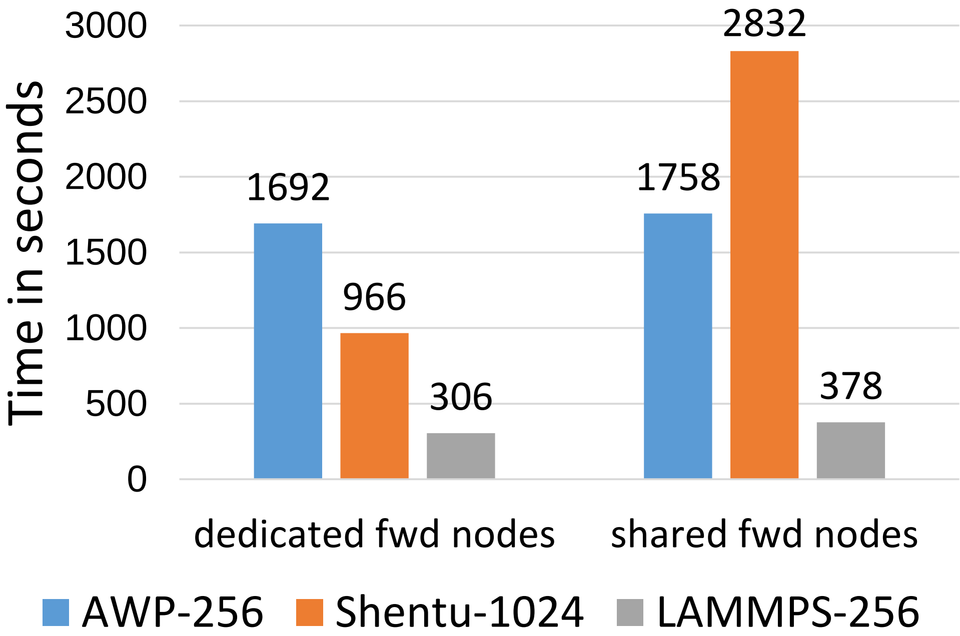


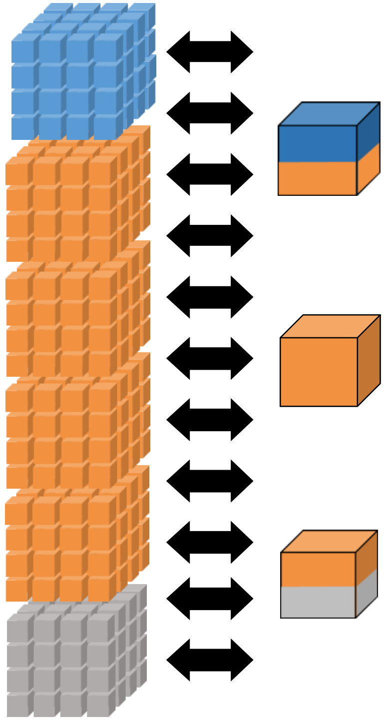
LAMMPS molecular dynamics

I/O interference brings I/O performance inconsistency and degradation



dedicated fwd nodes



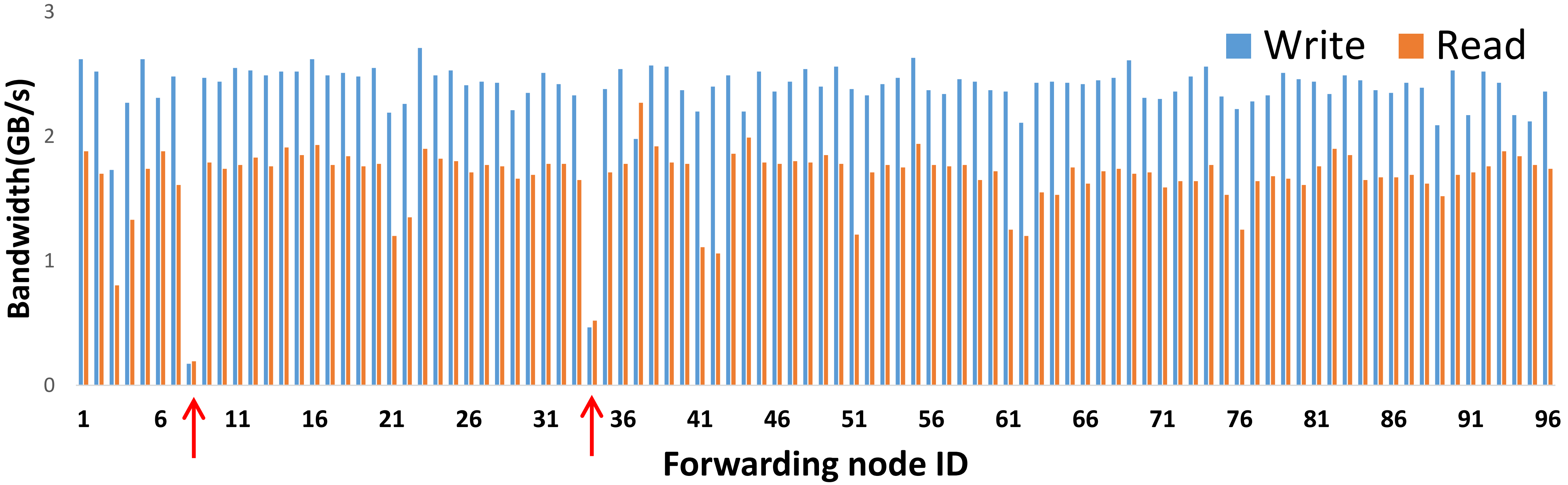


shared fwd nodes

Applications may suffer from I/O interference at I/O forwarding layer!

FFM Problem 3: Forwarding Node Anomaly

Fail-slow is also a significant problem to FFM



App performance severely hurt when assigned fail-slow forwarding nodes

Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems, Gunawi, FAST'18

Our Solution: *DFRA (Dynamic Forwarding Resource Allocation)*

Main idea: to ***upgrade*** forwarding node allocation on demand

Normal - default policy

Disruptive – private allocation

Demanding - more allocation

Per-App I/O Profiling : *Beacon End-to-end I/O Monitor*

- Automatic, transparent multi-level I/O monitoring
  - Allows DFRA to look up node status and per-application I/O profiles
  - Intuition: HPC applications have ***consistent I/O behavior*** across runs (jobs)

- Deployed at TaihuLight since April, 2017
- Paper at NSDI ’19
  - “End-to-end I/O Monitoring on a Leading Supercomputer”, by Yang et al.
- Code and I/O monitoring data released
  - https://github.com/Beaconsys/Beacon

Automatic Forwarding Node Scaling

- Identifying jobs **needing more than default FFM allocation**
- Target job eligible for consideration if application
  - has enough I/O volume (> V_{min}), and
  - has enough nodes performing I/O (> N_{min}), and
  - is ***not*** metadata-bound (***avarage metadata queue length < $W_{metadata}$***)

- Estimating adjusted forwarding node number ***S***
  - $S = N_{I/O} * B_c / B_f$

    B_f : I/O bandwidth per forwarding node

    B_c : I/O bandwidth per compute node

    $N_{I/O}$: #Nodes performance I/O
  - Upgrade if $S > N_f$
    - Do nothing otherwise

I/O Interference Avoidance

- Identifying jobs **requiring dedicated forwarding nodes**
  - Based on comprehensive inter-application I/O interference study
  - Pair-wise interference measurement on 8 apps with representative behaviors
    - 256-node runs, two apps sharing one forwarding node

Apps	MPI-IO$_N$	APT	DNDC	WRF$_1$	WRF$_N$	Shentu	CAM	AWP
MPI-IO$_N$	*(2.1,2.1)	**(1.1,9.3)**	**(4.8,1.1)**	(1.0,1.0)	*(2.1,2.0)	**(1.3,4.5)**	(1.0,1.0)	**(3.3,1.1)**
APT	-	*(2.0,2.1)	**(33.3,1.0)**	(1.0,1.0)	**(4.3,1.4)**	**(6.3,1.3)**	(1.0,1.0)	**(50.0,1.1)**
DNDC	-	-	*(2.0,2.0)	**(1.0,25.0)**	**(1.0,11.1)**	**(1.1,16.7)**	**(1.0,33.3)**	*(2.2,2.4)
WRF$_1$	-	-	-	(1.0,1.0)	(1.0,1.0)	(1.0,1.0)	(1.0,1.0)	**(50.0,1.0)**
WRF$_N$	-	-	-	-	*(2.1,2.1)	*(2.0,2.3)	(1.0,1.0)	**(12.5,1.3)**
Shentu	-	-	-	-	-	*(2.0,2.0)	(1.0,1.0)	**(12.5,1.1)**
CAM	-	-	-	-	-	-	(1.0,1.0)	**(100.0,1.0)**
AWP	-	-	-	-	-	-	-	*(2.0,2.0)

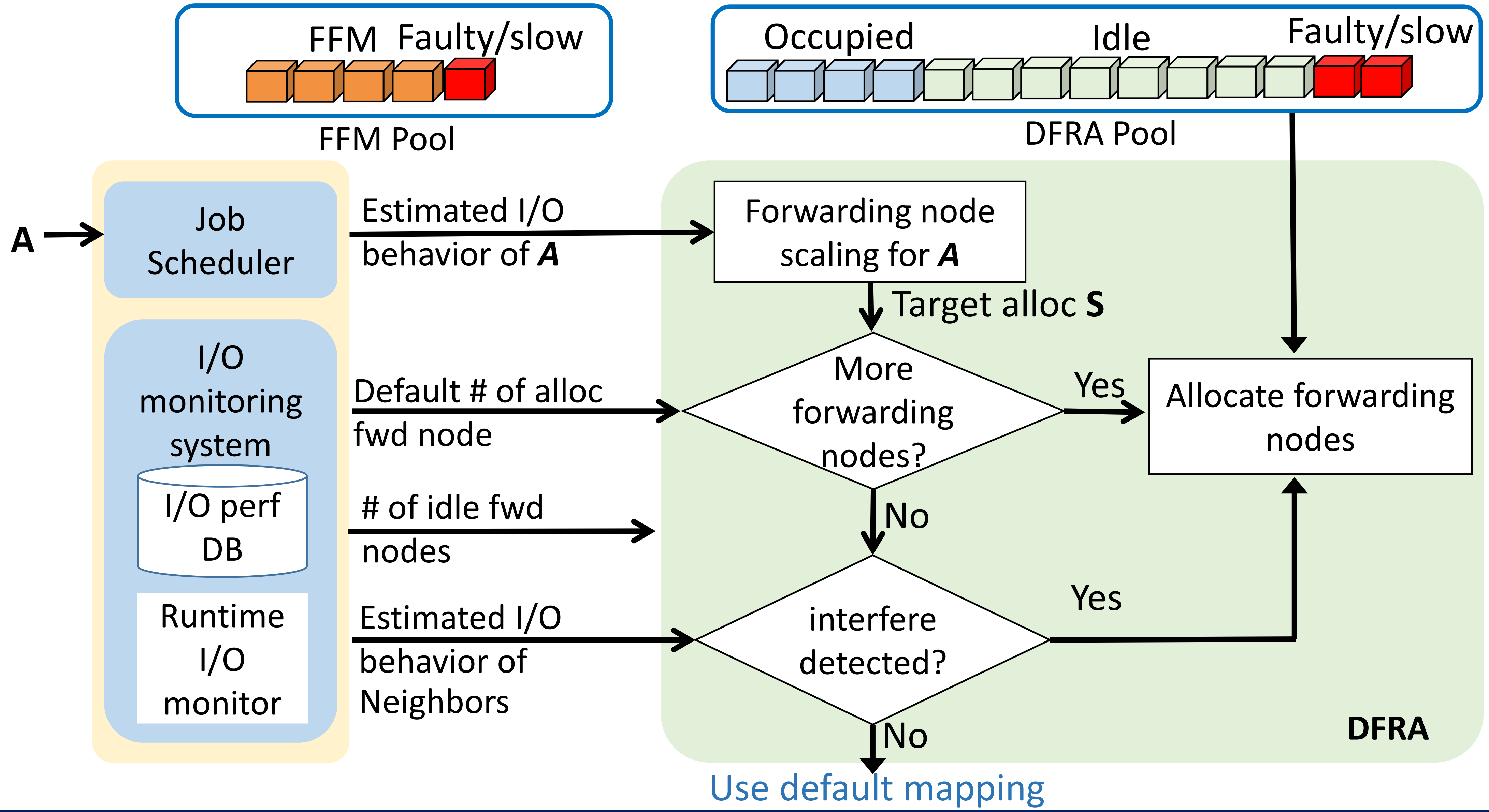
I/O slowdown factor pairs

I/O Interference Study: Summary

- Identified I/O interference ***root causes***:
  - N-1 I/O mode, metadata-heavy, and high-bandwidth workloads;
- High-IOPS workloads more vulnerable

- Detect I/O interference by checking both ***target job*** and ***existing neighbors*** on shared forwarding nodes to be allocated via FFM
  - Assign target job dedicated forwarding nodes if either party belongs to above categories

DFRA Workflow



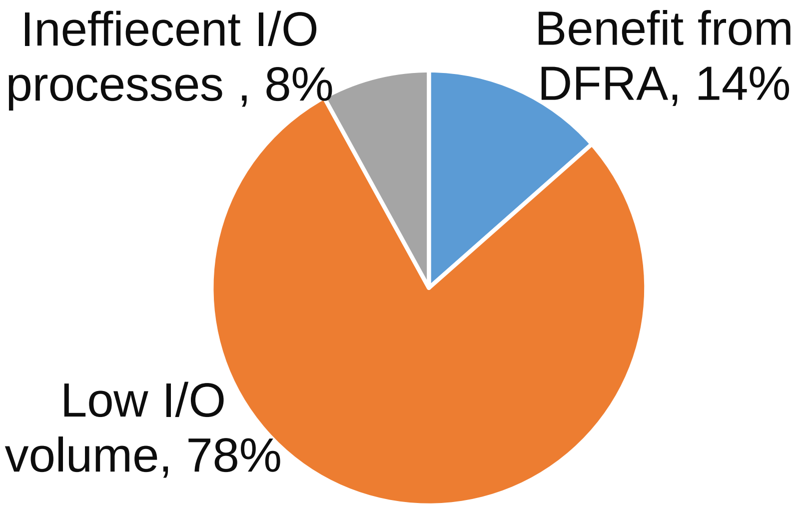
Implementation and Deployment Status

- Implemented to be used with SLURM-based scheduler, in ***C and Python***
- ***Remapping to more/dedicated forwarding*** nodes when job approved for upgrade
    - By relinking RDMA connection
    - ***Per-job*** basis, new mapping removed at end of job run
    - Currently no "downgrading"
- Partially deployed on TaihuLight production system since *Feb 2018*
    - Users "*opt in*" with job submission command
    - Intend to switch to "*opt out*" in the future

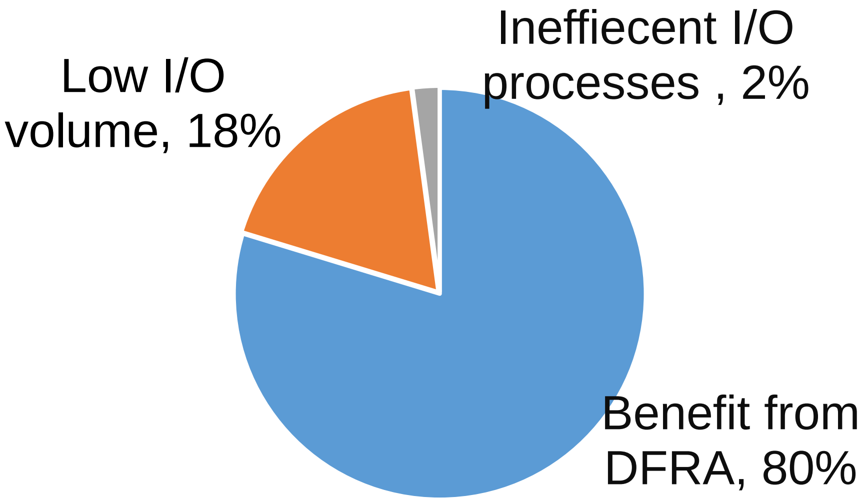
Evaluation 1: Upgrade Eligibility of Historical Jobs

- 18 months I/O history analysis (Apr 2017 – Sep 2018)

```
[2016-08-27 10:14:44] T lwfs-fuse: 69742: OPEN() btio.full.out => 0x200095032d0
[2016-08-27 10:14:44] T lwfs-fuse: 69745: WRITE (0x200095032d0, size=161320, offset=0, count=2)
[2016-08-27 10:14:44] T lwfs-fuse: 69740: RELEASE 0x20009500e40 (FLUSH implied)
[2016-08-27 10:14:49] T lwfs-fuse: 69805: OPEN() btio.full.out => 0x20009504160
[2016-08-27 10:14:49] T lwfs-fuse: 69811: READ (0x20009504160, size=262144, offset=262144, count=4)
```



By job count

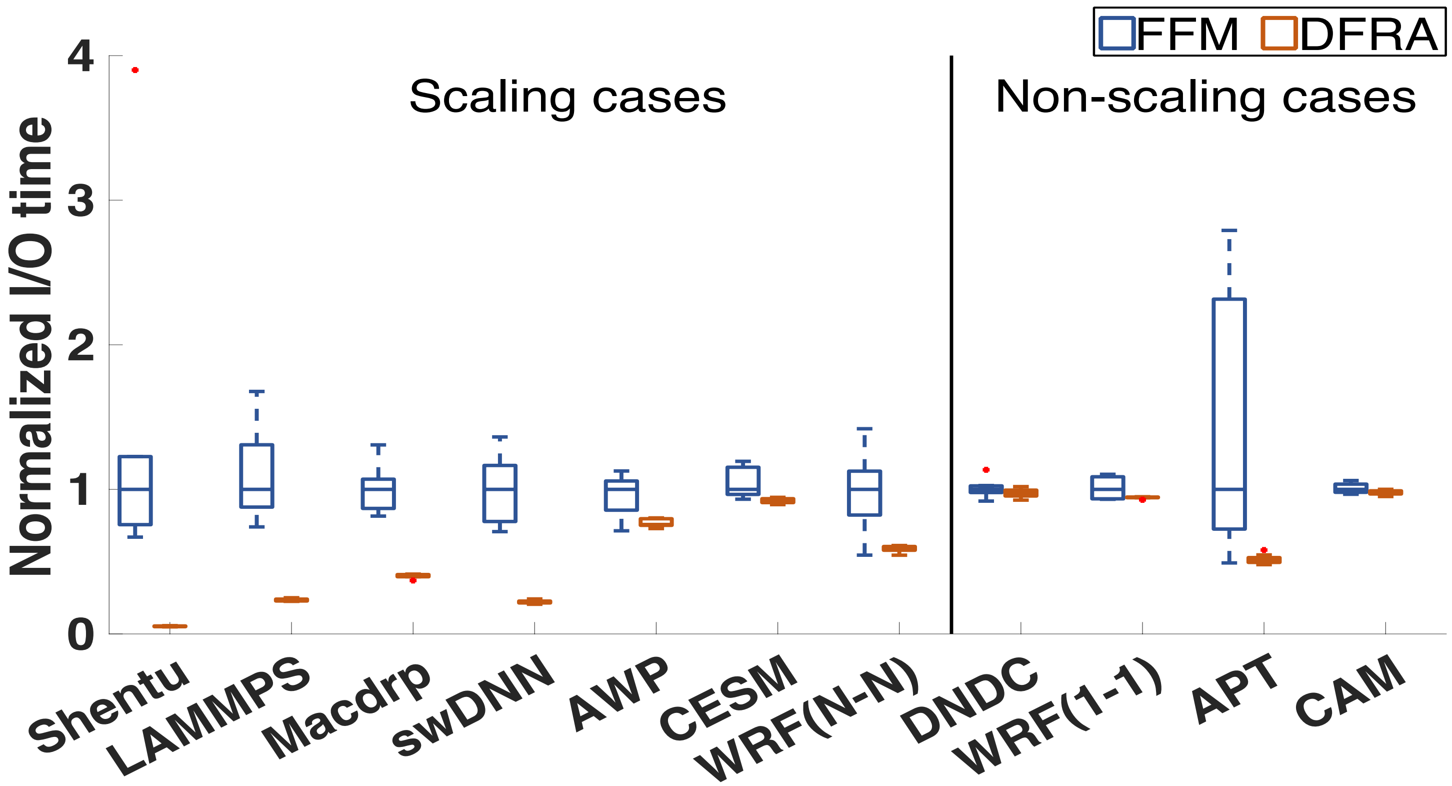


By core-hours consumed

Few jobs but consuming considerable core-hours are benefiting from DFRA

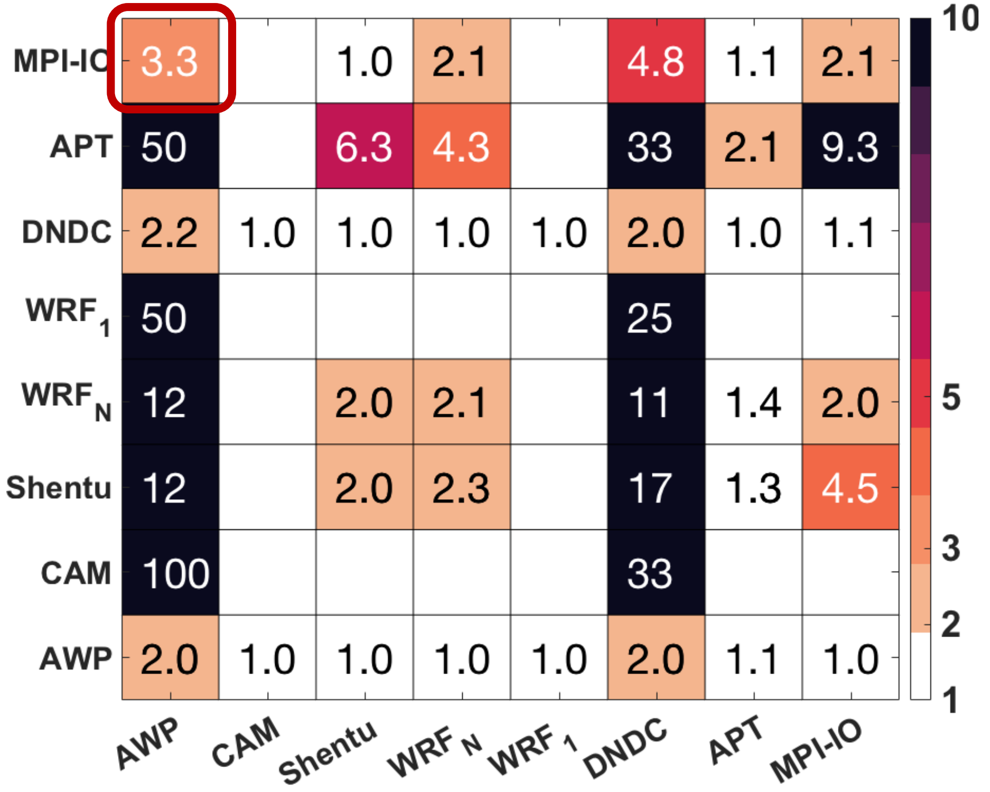
Evaluation 2: DFRA Effectiveness

Overall performance from 10 runs (each box) in production environment

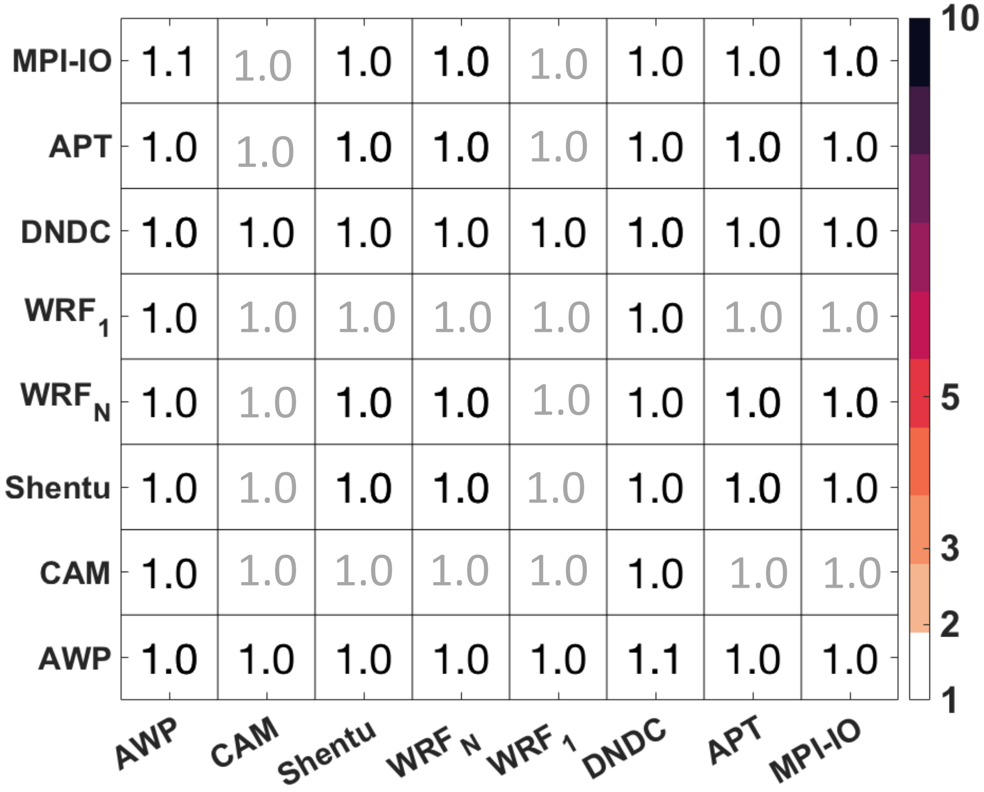


DFRA reduces I/O variation and improves I/O performance at I/O forwarding layer

Evaluation 3: I/O Interference Reduction



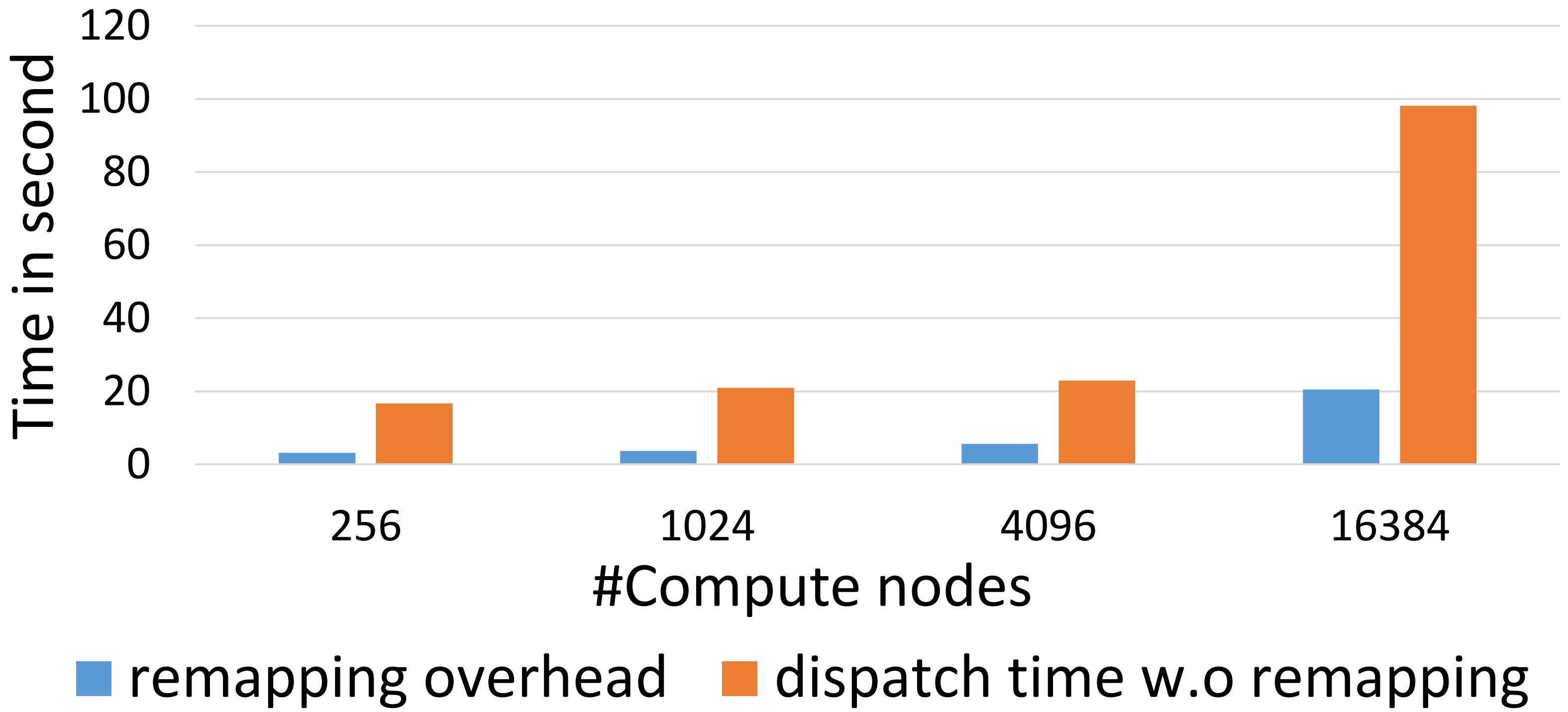
FFM



DFRA

DFRA reduce variation and improve I/O performance at I/O forwarding layer

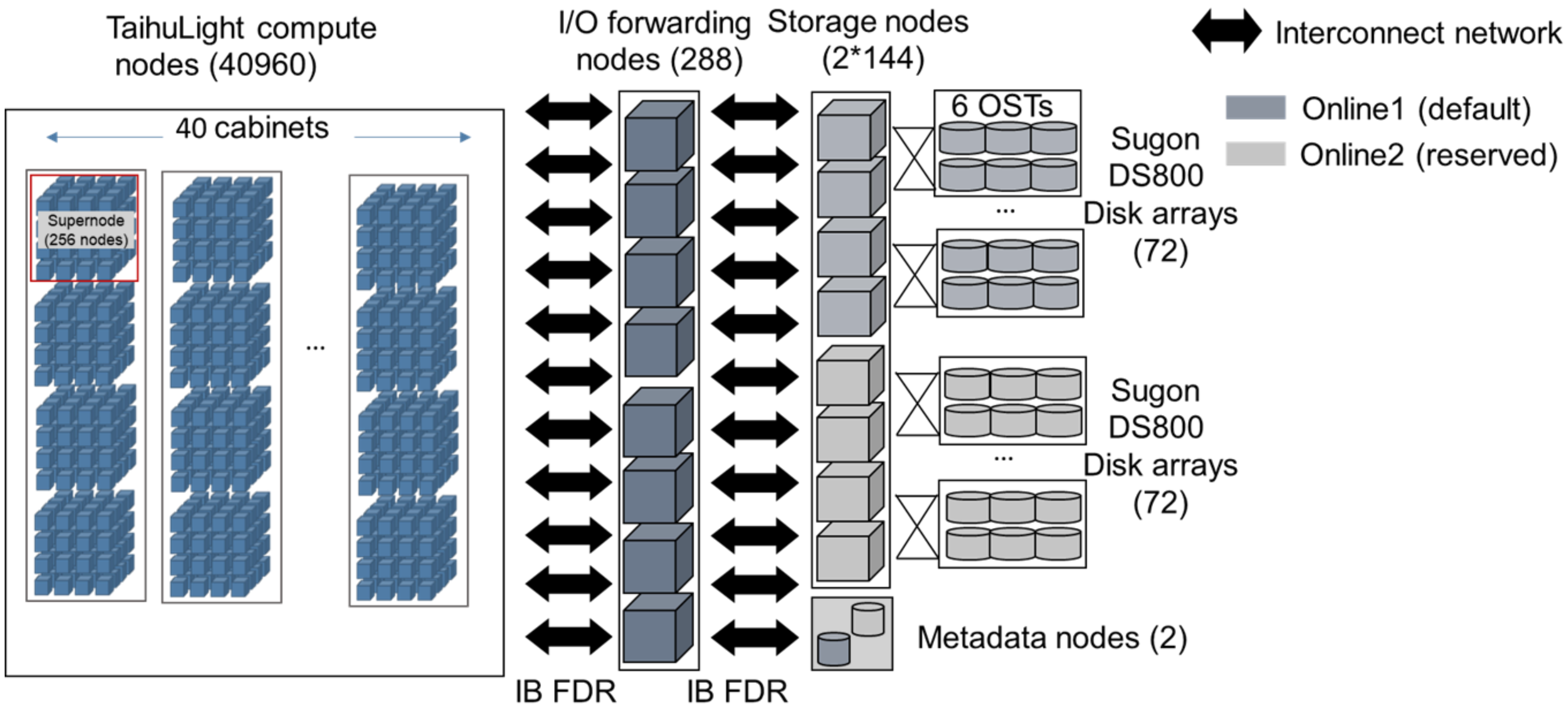
Evaluation 4: Remapping Overhead vs. Saving



Overhead acceptable considering benefit

- Average I/O time saving of 6 minutes for I/O-intensive jobs
- Estimated saving of 200 million core-hours in past 8 months

Conclusion



Take away points:

- Don't guess future user I/O demands
  - Over-provision, give low "basic plan", then upgrade when needed
- DFRA applicable to shared burst buffer management too

Q&A

Thank you!

Partial I/O monitoring data released at
https://github.com/Beaconsys/Beacon