Missouri University of Science and Technology

Modular Architecture for High-Performance and Low Overhead Data Transfers

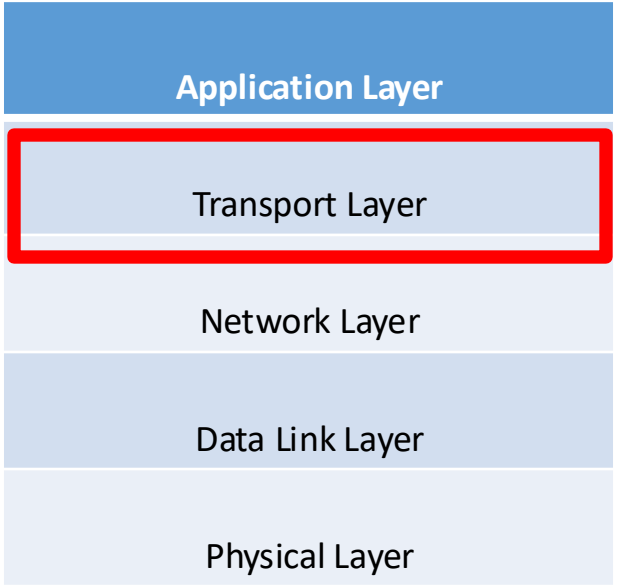
Rasman Mubtasim Swargo, Engin Arslan, and Md Arifuzzaman

The Data Deluge vs. Reality

- Modern scientific workflows are increasingly data-intensive
  - Example: Genome sequencing output grew from 5 MB (2006) → 150 GB (2023)
- The generated data must be moved across wide-area networks for analysis and archival
- Research backbones such as ESnet and Internet2 now deliver more than 400 Gbps connectivity
- However, legacy transfer tools often fail to reach high throughput

Needs for Application Layer Optimization

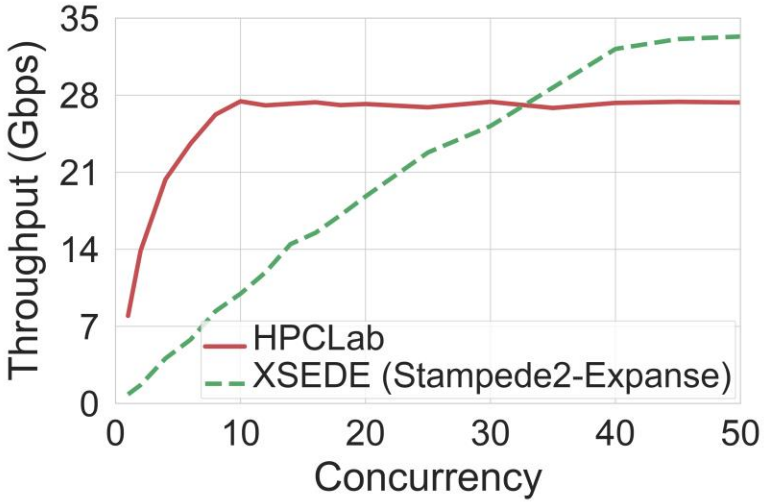
- Single file read/write operations cannot yield more than 20 Gbps throughput
- Single TCP connection is limited to around 30 Gbps throughput
- For production workloads, performance is significantly lower



TCP/IP Model

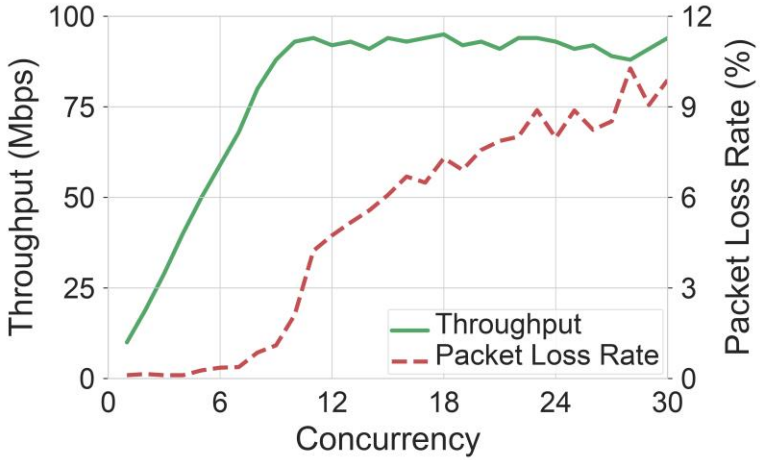
Transfer Parallelism (i.e., Concurrency)

- Concurrency is the key to increase system utilization.
- The optimal level of concurrency is not the same in all networks.



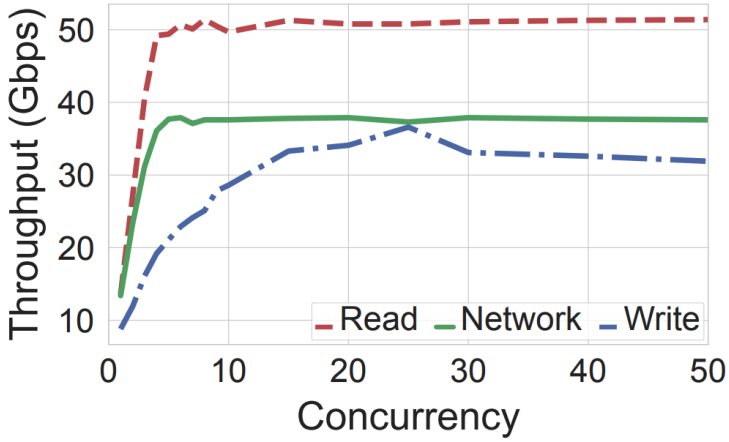
Adverse impacts of concurrency

- High concurrency has adverse impacts on performance and resource usage
  - Increase network congestion
  - Increase CPU usage
  - Increased I/O contention



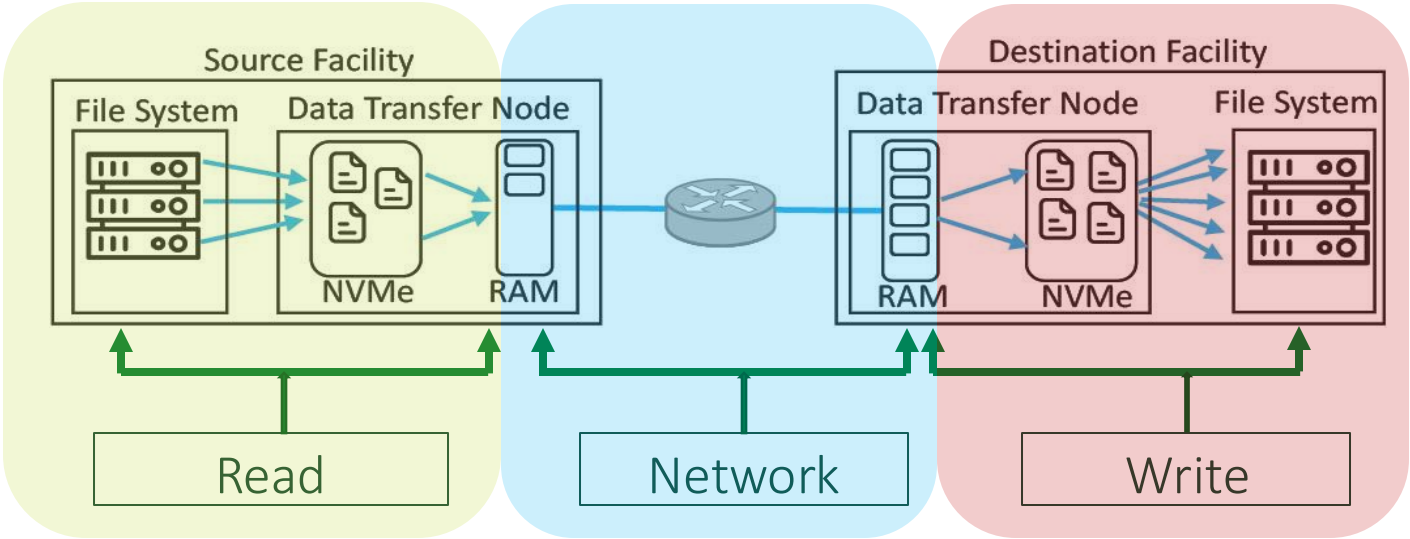
The Problem with Monolithic Optimization

- Core Problem: Optimal concurrency is not the same for read, network, and write operations in the same network.
- Example:
  - Network needs 5 threads for 38 Gbps.
  - File write needs 25 threads.
  - Monolithic tools force 25 threads for network, causing massive overhead.



Marlin: Modular File Transfer Architecture

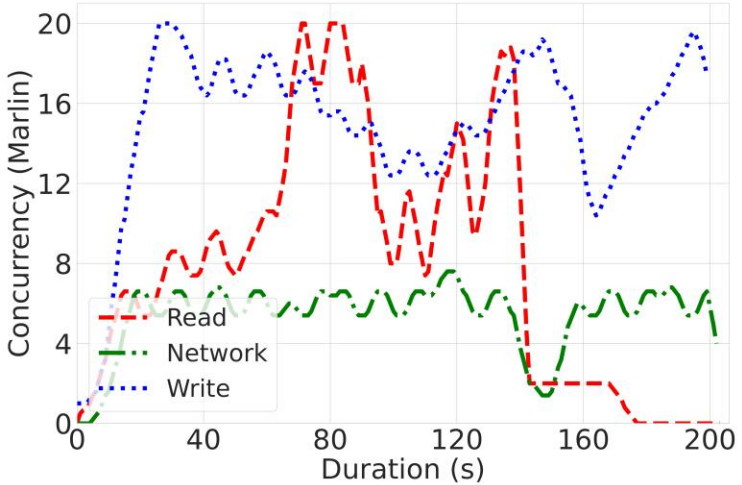
Marlin **introduces modular file transfer architecture to separate and read, network, and write operations and tune them independently.**



==> Arifuzzaman, Md, and Engin Arslan. "*Use only what you need: Judicious parallelism for file transfers in high performance networks.*" *Proceedings of the ACM 37th International Conference on Supercomputing*. 2023.

Marlin's Limitation: Uncoordinated Optimization

- Marlin used three *independent* optimizers (Gradient Descent) that do not coordinate.
- This approach takes longer time to converge.
- Optimizers may get misdirected as one of them may mistakenly believe the throughput increase happened due to its action.

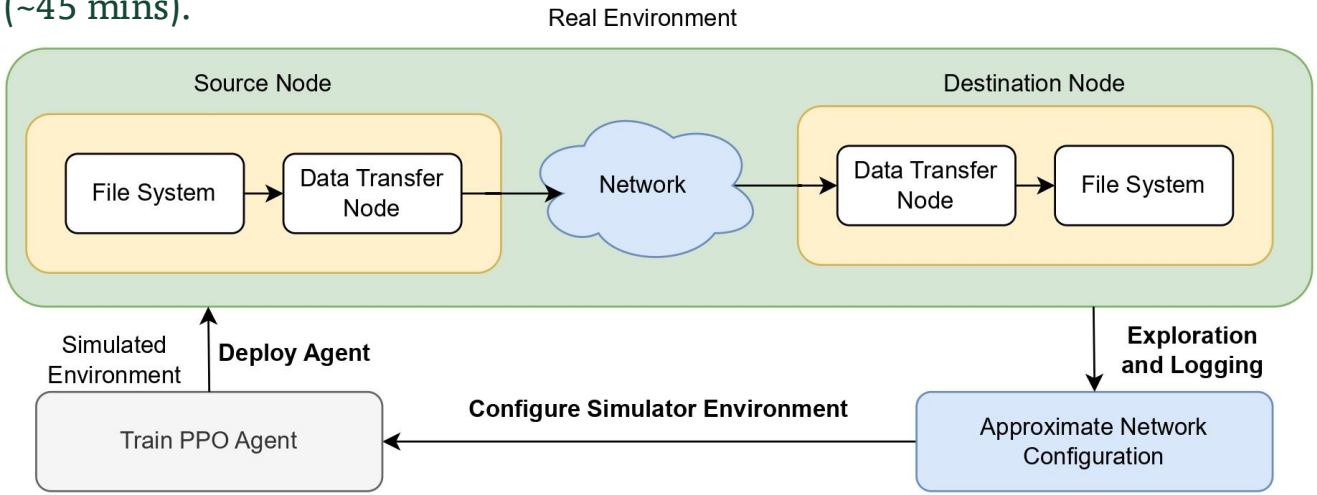


Proposed Solution: AutoMDT

- Modular Design: Build on the modular idea, decoupling Read, Network, and Write.
- Joint Optimization: A single Deep Reinforcement Learning (DRL) agent *jointly* optimizes all three variables, learning their complex interactions.
- Fast Offline Training: A lightweight simulator solves DRL's "long training time" problem.
  - Trains in ~45 minutes (vs. ~7 days online).

AutoMDT System Architecture

- Phase 1-2 (Training):
  - Briefly "Explore" the real network and configure Simulator.
  - Train PPO Agent offline (~45 mins).
- Phase 3 (Inference):
  - Deploy the fully-trained agent for real, high-performance transfers.



Phase 1: Learning the Environment

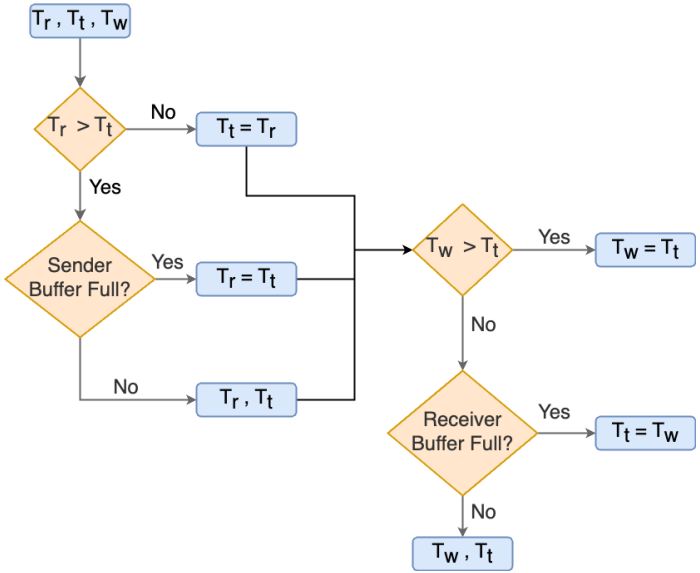
- Before training, we must "teach" the simulator about the real network.
- We run a 10-minute "random-threads" run.
- This captures two key parameter sets:
  - Max Bandwidth (B_r, B_n, B_w)
  - Throughput-per-thread (TPT_r, TPT_n, TPT_w)
- These values are used to configure the simulator.

Phase 1: The Dynamics Simulator

- Why a simulator? Online DRL training is impractical.
  - Average episodes required: ~20150
  - Each episode: 10 steps * 3 seconds = 30 seconds
  - Estimated time: 20150 * 30 seconds = 604500 seconds ~7 days.
  - Estimated waste: ~7.6 Petabytes of data in a 100Gbps network.

Phase 1: The Dynamics Simulator

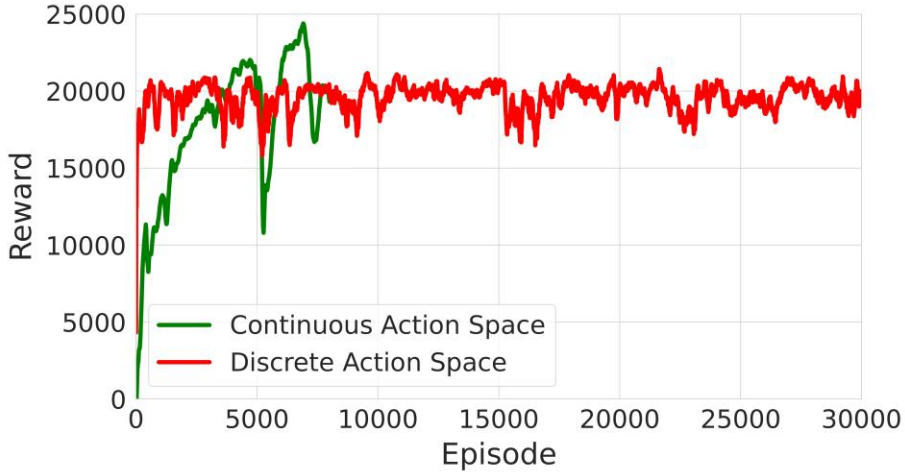
- Our Solution: A lightweight simulator emulates the memory buffer dynamics.
- It models how buffers fill and drain, which is all the agent needs to learn.



T_r = Read Throughput
T_w = Write Throughput
T_t = Transfer Throughput

Phase 2: Offline Agent Training

- We use Proximal Policy Optimization (PPO), a state-of-the-art DRL algorithm.
- The agent runs thousands of "simulated transfers" per second, learning the optimal policy.
- Result: Converges in ~45 minutes on average.



PPO Agent Design: What it Sees, What it Does

- The agent learns to map State -> Action.
- State Space (What it *Sees*):
  - Current thread counts (n_r, n_n, n_w)
  - Achieved throughputs (t_r, t_n, t_w)
  - Sender & Receiver buffer fullness.
- Action Space (What it *Does*):
  - Predicts the *next set* of thread counts: (n_r, n_n, n_w)

PPO Agent Design: The Utility Function

- The agent's "goal" is to maximize a utility function.
- The utility function is: $U(n_i, t_i) = U_{read}(n_r, t_r) + U_{network}(n_n, t_n) + U_{write}(n_w, t_w)$
- In plain English:
  - Rewards high throughput .
  - Penalizes high thread counts (K^n).
  - This forces the agent to find the "sweet spot" of high performance with low overhead.

Where,

$$U_{read}(n_r, t_r) = \frac{t_r}{K^{n_r}}$$

$$U_{network}(n_n, t_n) = \frac{t_n}{K^{n_n}}$$

$$U_{write}(n_w, t_w) = \frac{t_w}{K^{n_w}}$$

Phase 3: Deployment in Production

- The fully trained agent is loaded and runs online during the real transfer.
- Continuous Adaptive Loop:
  - Sample the *real* network state (throughputs, buffers).
  - Predict the optimal concurrencies.
  - Apply settings & repeat.
- This allows it to adapt to changing network conditions in real-time.

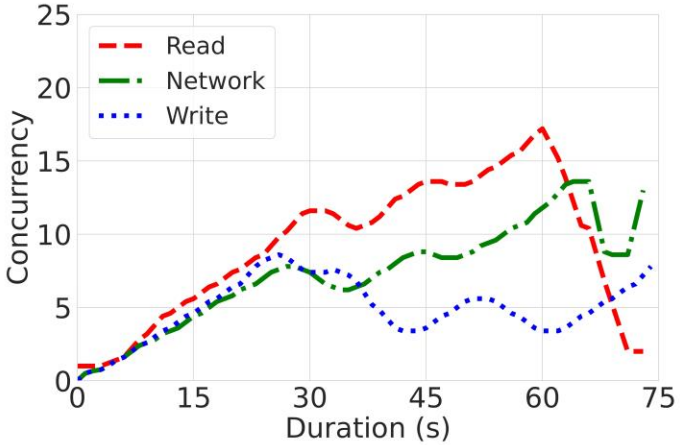
Evaluation

Evaluation: Testbeds & Baselines

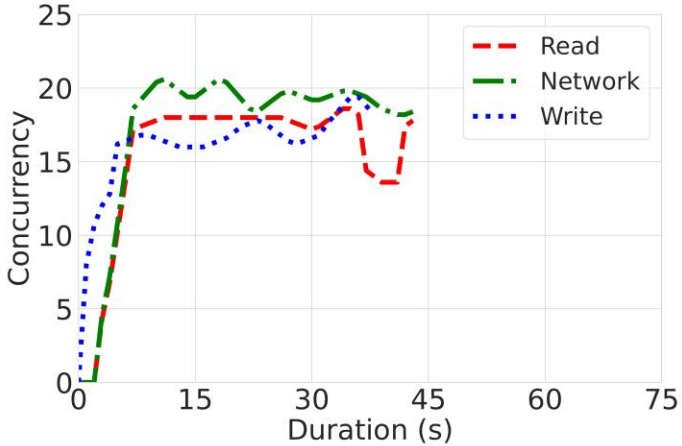
- Testbeds:
  - CloudLab (1 Gbps)
  - Fabric (NCSA to TACC): Main testbed, ConnectX-6 NICs (30 Gbps).
- Baselines:
  - Globus: Monolithic, heuristic-based (Static Solutions).
    - We used globus-url-copy from the open-source Grid Community Toolkit (GCT 6.2).
    - We set the concurrency to 4 and parallelism to 8.
  - Marlin: Modular, independent optimizers (Adaptive).

Result: AutoMDT is Stable, Marlin is Not

- AutoMDT quickly finds the optimal concurrencies (~20) and *stays there*. It is stable.
- Marlin's independent optimizers never stabilize.
- The threads constantly fluctuate and fight each other.



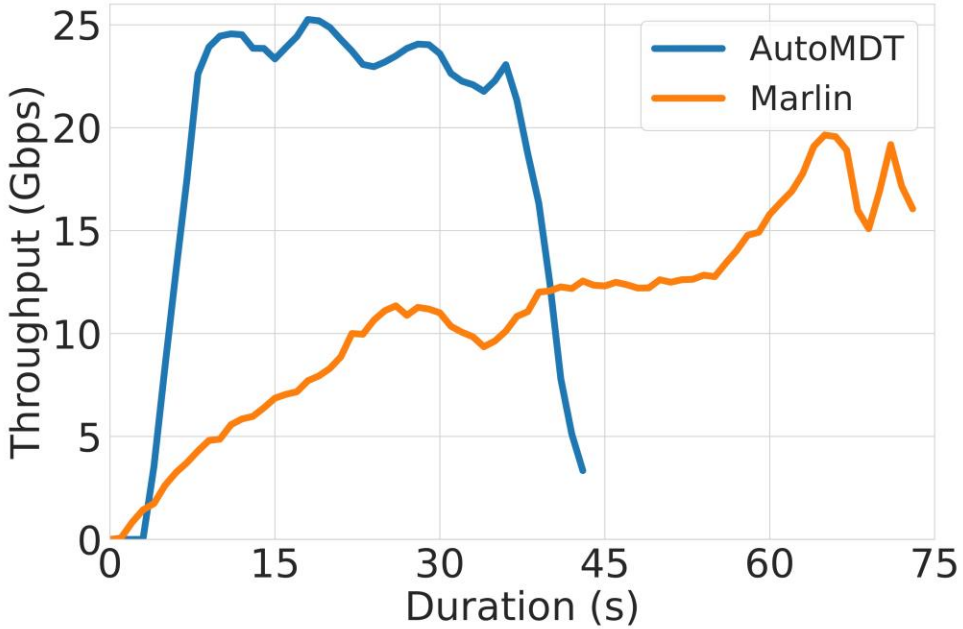
Marlin



AutoMDT

Result: Stability = Speed

- Key Takeaways:
  - AutoMDT (blue): Reaches 20+ Gbps in 7 seconds.
  - Marlin (orange): Takes 62 seconds to reach 14 Gbps.
  - AutoMDT has 8x faster convergence.
  - Bottom Line: AutoMDT finishes the transfer 68% faster.

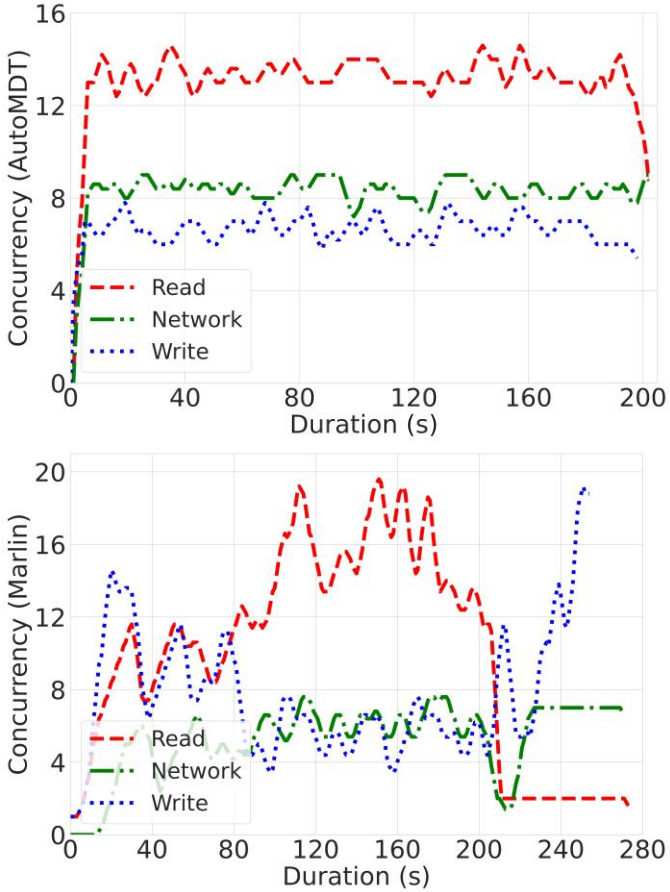


Bottleneck Analysis: Advantage of Modular Design

- This is the *real* test of a smart, modular system.
- We created 3 bottleneck scenarios on the Cloudlab testbed:
  - Read I/O Bottleneck (Slow disk at source)
  - Network Bottleneck (Slow network path)
  - Write I/O Bottleneck (Slow disk at destination)

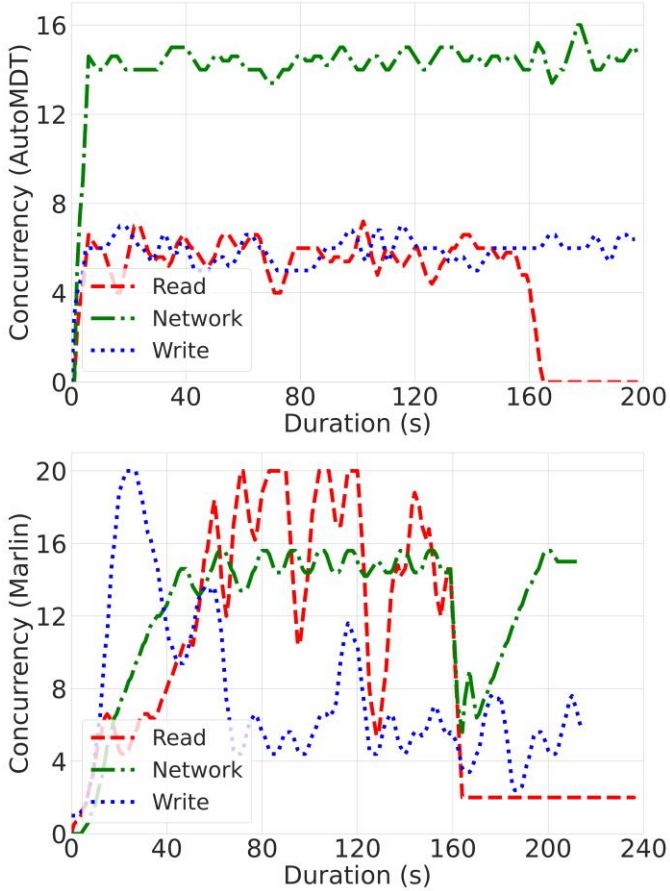
Read I/O Bottleneck

- AutoMDT (top):
  - Correctly identifies the `Read` bottleneck.
  - Raises `Read` threads (red) to compensate.
  - Keeps `Network` (green) and `Write` (blue) threads low (no wasted resources).
- Marlin (bottom):
  - Is completely confused. All three optimizers fluctuate, failing to find the true bottleneck.



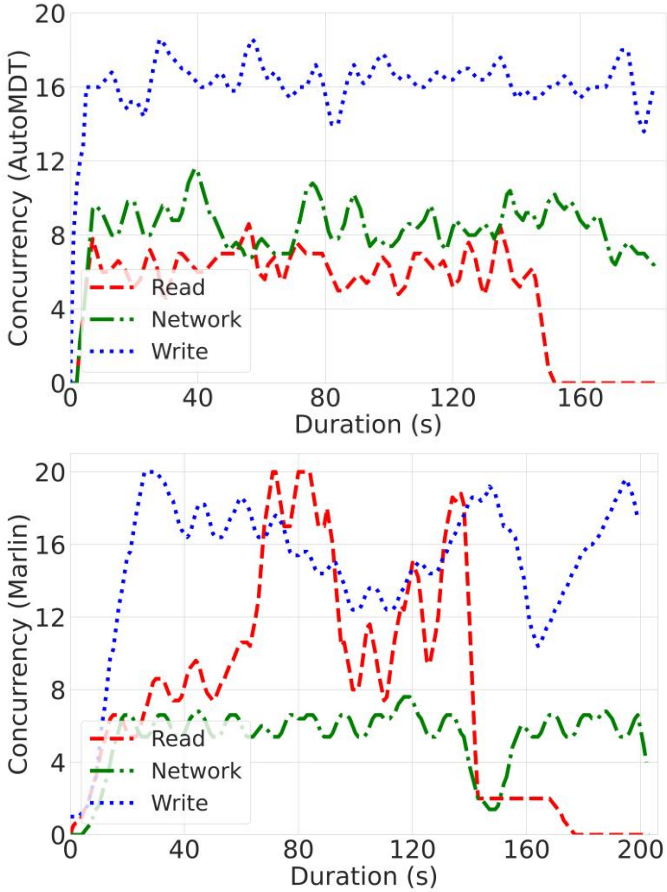
Network Bottleneck

- AutoMDT (top):
  - Correctly identifies the `Network` bottleneck.
  - Raises `Network` threads (green).
  - Keeps I/O threads (`Read`/`Write`) low and stable.
- Marlin (bottom):
  - Again, is unstable and wastes resources fluctuating I/O threads.



Write I/O Bottleneck

- AutoMDT (top):
  - Correctly identifies the `Write` bottleneck.
  - Raises `Write` threads (blue) while keeping others low.
- Marlin (bottom):
  - Large fluctuations in read and write concurrencies.
- This proves the independent optimizer approach is fundamentally flawed.



Performance in High-Speed Network

- Periodic data transfers were conducted in the FABRIC Testbed (NCSA → TACC).
- Compared Globus, Marlin, and AutoMDT across two 1TB datasets:
  - Large dataset: 500 files, each 2 GB.
  - Mixed dataset:

    80% medium files (1–100 MB)

    10% large files (1–2 GB)

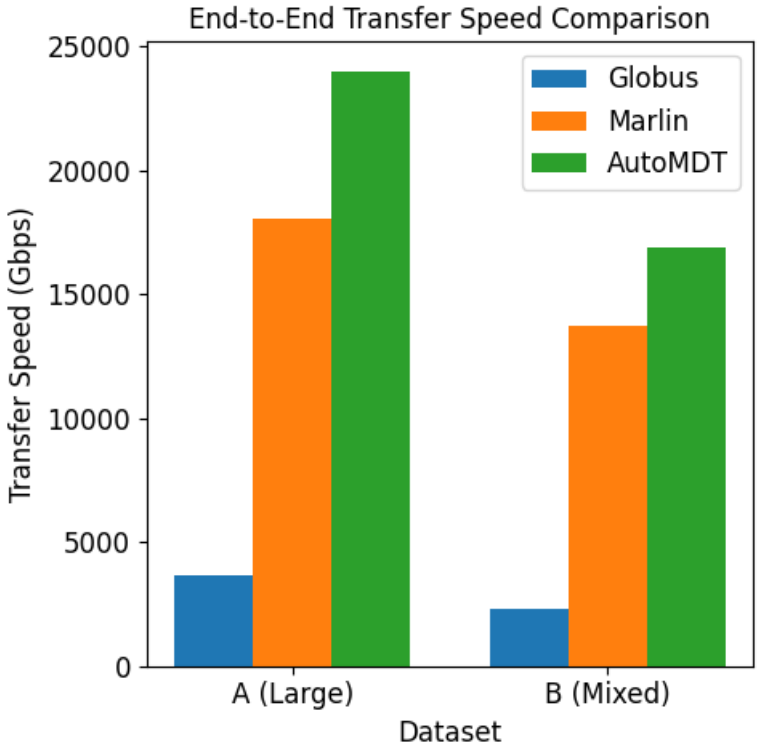
    5% small files (500 KB–1 MB)
- Each experiment was repeated multiple times to ensure consistency.

Performance in High-Speed Network

- Key Numbers (Large Files):

  AutoMDT (23.9 Gbps) is:
  - 1.33x faster than Marlin
  - 6.57x faster than Globus
- Key Numbers (Mixed Files):

  AutoMDT (16.9 Gbps) is:
  - 1.23x faster than Marlin
  - 7.28x faster than Globus



Summary

- Monolithic optimizers are inefficient, wasting I/O resources.
- AutoMDT's modular design decouples Read, Network, and Write.
- A single, joint DRL agent (PPO) outperforms unstable independent optimizers.
- A novel simulator enables fast, practical offline training (~45 min vs. 7 days).
- Result: AutoMDT achieves 8x faster convergence and up to 68% faster transfers than the state-of-the-art.

Thank You!

Rasman Mubtasim Swargo

Questions?

Missouri University of Science and Technology