Extending and Benchmarking the ROSS PHOLD Model

John Cohen
Alexander Gibson
Salonee Patel
Ethan Xie
cohenj6@rpi.edu
gibsoa2@rpi.edu
patels21@rpi.edu
xiew6@rpi.edu
Rensselaer Polytechnic Institute
Troy, New York, USA

ACM Reference Format:
John Cohen, Alexander Gibson, Salonee Patel, and Ethan Xie. 2024. Extending and Benchmarking the ROSS PHOLD Model. In *PCPE Spring '24.* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 ABSTRACT

Our primary goal with this project is to understand how MPI I/O affects the PHOLD model's performance during simulation through robust testing. These tests aim to quantify the overhead introduced by MPI I/O and its scalability under various loads, and include weak scaling, strong scaling, I/O operation frequency, and packet size experiments. This is critical for designing simulations that are both large-scale and performance-optimized. The MPI I/O operations themselves aren't functional within the simulation's logic but are used to simulate potential real-world data handling scenarios and observe their impact on runtime. Ultimately, we confirmed that massive parallelization greatly improves the network's performance in the strong scaling case, but becomes quickly bottlenecked by MPI I/O overhead in the weak scaling case. We also found that performance is more strongly affected by I/O operation frequency than by packet size. Ultimately, keeping these dependencies in mind will greatly assist in designing networks which support MPI I/O on distributed computing systems.

2 INTRODUCTION

The focus of our project was to enhance ROSS's existing parallel network model phold, which is widely used for researching large-scale parallel and distributed computing environments. The primary objective was to explore and implement potential extensions or improvements to the ROSS phold model, thereby providing deeper insights into network communication system behaviors under various conditions. Our group performed a detailed analysis of the current ROSS phold model, identifying its limitations and areas where it could be enhanced to simulate more realistic or complex network scenarios.

Our project began with a comprehensive review of the existing ROSS phold model. ROSS is heavily inspired by the Georgia Tech Time Warp (GTW), a parallel discrete event simulator developed for optimistic synchronization. It attempts to predict packet destinations for simultaneous packet "receives" and executes rollbacks, or "warps," when it is incorrect [2].

Once we completed the review, our team identified specific areas for extension or improvement - namely, the introduction of MPI I/O. The planned extensions were designed to not only improve the model but also to add new capabilities that can address research questions previously unexplored due to the model's limitations. The development phase involved the actual coding and integration of the proposed extensions into the existing ROSS PHOLD model. This was done in a manner that maintains compatibility with parallel execution frameworks and distributed computing environments. Our team adhered to best coding practices to ensure that the extended model is reliable, maintainable, and scaleable.

To validate the enhancements and assess their impact, we designed and ran a series of benchmark scenarios, including weak and strong scaling experiments. These benchmarks stress-tested the extended network model across a range of network configurations, traffic patterns, and workload distributions. The testing was conducted on the AiMOS parallel computing platform, leveraging its capabilities to evaluate the model's performance under different conditions, and was partially inspired by performance metrics for large-scale MPI programs [1].

The final phase of our project involved a thorough analysis of the benchmark results. Our team assessed the extended model's scalability, efficiency, and reliability, comparing these metrics against those of the original model. This analysis allowed us to understand the improvements' impact and identify any remaining bottlenecks or areas for further optimization.

Overall, our project aimed to significantly enhance the utility and accuracy of the ROSS PHOLD network model, providing valuable tools for researchers studying advanced network systems and their behaviors in large-scale, parallel, and distributed environments.

3 IMPLEMENTATION

The PHOLD model, a fundamental paradigm used in discrete event simulation, has been extended in our implementation to integrate

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

MPI I/O operations, enhancing its utility for performance testing under various computational loads and data management scenarios. This adaptation is particularly designed to analyze the impact of I/O operations on the simulation's performance, offering insights into how data-intensive tasks might affect runtime in distributed computing environments. This has massive implications as we enter a time where "big data" is king, as discussed in [8].

In our implementation, MPI I/O operations are executed at the end of every event handling within the phold_event_handler function, shown below.

```
void
phold_event_handler(phold_state * s, tw_bf * bf,
                    phold_message * m, tw_lp * lp)
{
  (void) s;
  (void) m;
  tw_lpid         dest;

  if(tw_rand_unif(lp->rng) <= percent_remote)
  {
        bf->c1 = 1;
        dest = tw_rand_integer(lp->rng,
                               0, ttl_lps - 1);
  } else
  {
        bf->c1 = 0;
        dest = lp->gid;
  }

  if(dest >= (g_tw_nlp * tw_nnodes()))
        tw_error(TW_LOC, "bad dest");

  tw_event_send(tw_event_new(dest,
    tw_rand_exponential(lp->rng, mean)
    + lookahead, lp));
  if(test_mode) handle_io(lp); //Addition
}
```

This systematic integration allows for continuous data writing to a file, simulating a scenario where simulation state or event details are persistently logged or state checkpoints are frequently required. This setup is crucial for understanding the overhead introduced by I/O operations, especially in a high-frequency event generation context like that of the PHOLD model. Here is a code snippet for part of our MPI I/O handling:

```
void handle_io(tw_lp *lp)
{
  int r = rand() % io_frequency;
  // if frequency = 10, 10% of event handles
  // cause io writes, if ==1, every event
  // is io write
  if (r) return;
  write_mpi_io("event_data.bin", lp);
}
```

The .h file has been augmented with several configuration variables that are pivotal for steering the behavior of the simulation under different test conditions. These include settings for the frequency of I/O operations (io_frequency), the data size for each write operation (data_size), and a general test mode (test_mode) which dictates the type of scaling test to be conducted (e.g., strong or weak scaling). This configurability allows for flexible testing across a range of conditions, making the simulation adaptable to various research needs and scenarios. These parameters are vital for validating the robustness and scalability of the PHOLD model when subjected to the dual pressures of computational processing and data management tasks. The default parameters are listed here:

```
tw_stime lookahead = 1.0;
static unsigned int stagger = 0;
static unsigned int offset_lpid = 0;
static tw_stime mult = 1.4;
static tw_stime percent_remote = 0.25;
static unsigned int ttl_lps = 0;
static unsigned int nlp_per_pe = 1;
static int g_phold_start_events = 1;
static int optimistic_memory = 100;

//Addition
//Tests: 1) Strong, 2) Weak,
// 3) Frequency, 4) Data Size
static int test_mode = 1;
static int io_frequency = 100;
static int data_size = 1048576;
```

Additionally, the inclusion of these parameters supports detailed experimental setups where the effects of different I/O frequencies and data sizes can be meticulously analyzed. This enables a granular examination of how data I/O impacts the simulation dynamics, providing valuable data that can inform the optimization of distributed simulations and the design of more efficient data handling strategies within these frameworks. These enhancements not only broaden the applicability of the PHOLD model in research and practical scenarios but also contribute to the broader discourse on optimizing simulation frameworks in the era of big data and extensive parallel processing, such as in [3] [4] [6].

4 CONTRIBUTIONS

4.1 John

John was instrumental in the development of our enhanced ROSS PHOLD model. As the lead coder of our group, John was responsible for gaining a working understanding of the existing PHOLD framework and incorporating our MPI I/O ideas into it. In addition to doing an incredible job with the MPI I/O implementation, John then designed a full testing suite for our new and improved PHOLD model, developing both the required weak and strong scaling experiments as well as two additional experiments inspired by [9] for the purpose of further analyzing network load in a massively

parallel system. Additionally, he wrote the introduction section of this paper.

4.2 Alexander

After revising our approach to the project, we needed to brainstorm ways to utilize MPI I/O in enhancing the ROSS PHOLD model. Alexander came up with the idea of each rank writing its current event details to a global file periodically, which would both allow the PHOLD model to keep a "running tally" of its progress as well as give us a benchmark for measuring the network overhead associated with MPI I/O operations. The purpose of these MPI I/O operations is to primarily to emulate real-life network load, without actually influencing the results of the underlying PHOLD simulations.

Alexander was also responsible for expanding and executing the testing suite developed by John on the AiMOS supercomputer. Specifically, Alexander modified the original strong scaling test to properly account for the PHOLD model's handling of problem size by MPI rank, as a scale factor of 16 existed under the hood that had to be accounted for. After performing the weak scaling, strong scaling, I/O operation frequency and packet size experiments, Alexander created graphs for all of the data collected, as well as calculated the % speedups for all relevant data points relative to their respective "base cases."

Additionally, Alexander was responsible for writing the abstract and results sections of this paper, as well as handling all of the formatting due to his experience with LaTeX.

4.3 Salonee

A significant part of Salonee's contributions to this research involved an in-depth investigation into the CODES and ROSS simulation platforms. These tools were crucial for accurately modeling and simulating the complex interactions within MPI I/O operations across various storage systems, especially when dealing with large-scale HPC environments. By analyzing the existing literature and software documentation, Salonee developed a comprehensive understanding of how these systems function and their potential applications in MPI I/O simulation. This understanding was pivotal in adapting these simulations to reflect more accurately the real-world performance impacts of MPI I/O strategies, particularly those involving burst buffers.

Moreover, Salonee's technical contributions extended to setting up and configuring the CODES and ROSS simulation environments on our local machines. This process required a detailed evaluation of system requirements, dependencies, and optimal configuration settings to ensure that our simulations were both accurate and efficient. Salonee's efforts in this area ensured that our team could smoothly transition from theoretical models to practical, executable simulations. She streamlined the setup process by creating a series of scripts and guides that allowed our team members to replicate the setup quickly on their own systems, thus enhancing our collaborative capabilities and increasing our research's overall efficiency and productivity.

4.4 Ethan

Our team initially aimed to improve the performance of message handling of Rensselaer's Optimistic Simulations Systems by accelerating it using a CUDA kernel. We collaboratively investigated the feasibility of integrating CUDA kernel with the system, specifically focusing on the time warp scheduler component tw_sched.c from core. Unfortunately, the code's lack of comments, complex variables naming, and the complexity of the system significantly hampered our attempts to understand the code. After analysis, we believed that this approach wouldn't provide significant performance improvements.

Recognizing the limitations of CUDA kernel approach, our team shifted the focus. Ethan went back to researching within the ROSS project to identify possible areas where we could use Message Passing Interface I/O (MPI I/O) to enhance performance. Reading through the documentation regarding tw_output() and LP printing, Ethan believed this could be a promising area where Message Passing Interface I/O could be implemented to potentially improve performance of the ROSS PHOLD model. The utility component in ROSS core contains functions that control LP printing, which helped us with our next steps.

After Ethan shared these findings with the group, we decided our next steps. Later Ethan researched how exactly we could use it to improve the ROSS PHOLD model with the rest of the team members. The biggest challenge during research was that the code was not well-commented and we all found the naming of the variables confusing. Add on to the complexity of the system, it really took us some time to set up ROSS and test our models to continue working, but Ethan's dedication to understanding the ROSS documentation paid off in the end.

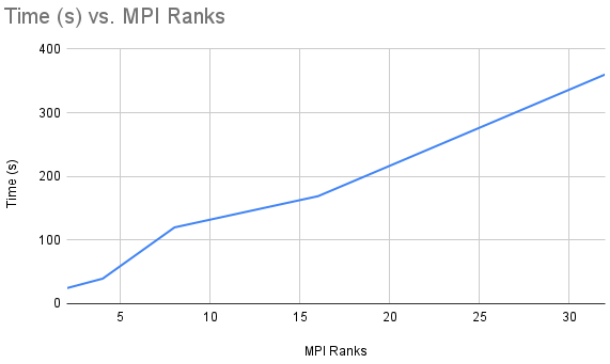
5 RESULTS

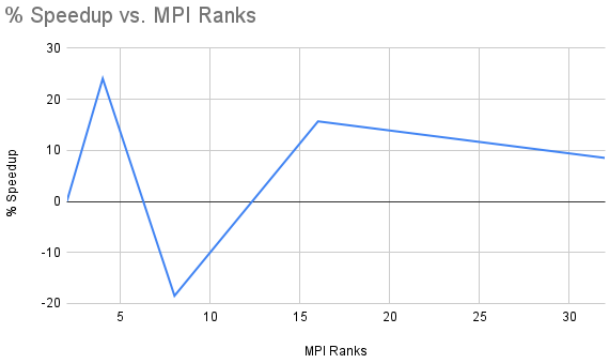
5.1 Weak Scaling Experiment

We performed a weak scaling experiment to observe how adding compute resources affects the supercomputer's performance as the problem size increases. In this experiment, we progressively doubled the number of MPI ranks, while keeping the number of operations per rank constant, thus linearly scaling the problem. We then obtained the following results:

Table 1: Compute Time for Weak Scaling

MPI Ranks	Time (s)
2	24.424
4	39.378
8	119.86
16	168.91
32	360.14





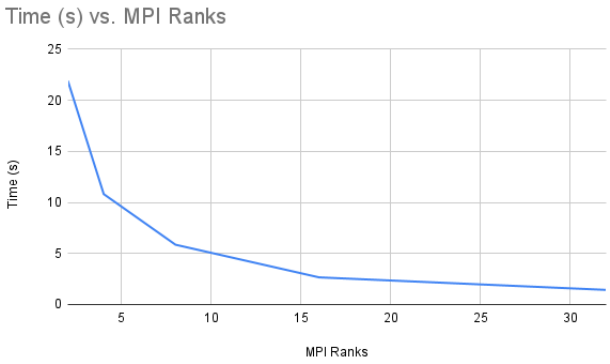
From these results, there exists a roughly linear relationship between the number of MPI ranks and the compute time, although as the number of ranks increases % speedup also increases slightly. This was expected since the bulk of our runtime comes from performing MPI I/O operations. Since all MPI ranks write to a shared file, it makes sense that increasing the number of ranks results in a linear increase in compute time. Additionally, higher rank counts should be slightly better since the PHOLD model is highly parallelizable, and we observe this phenomena in the data too. This all comes down to Amdahl's Law: in a highly parallel system, performance is dictated by the serial components of the program. In our case, the "serial" component is the linear scaling of MPI I/O operations with the number of MPI ranks, which largely outweighs the performance improvements from distributing the computations done in the native PHOLD model.

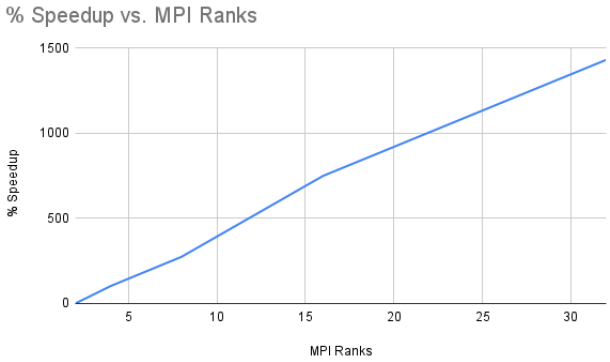
5.2 Strong Scaling Experiment

We also performed a strong scaling study to observe how increasing the degree of parallelization on a problem of constant size affects compute time. In this experiment, we worked with a fixed data event count of 64 1KB events for the PHOLD simulation and thus a fixed number of MPI I/O operations across all ranks. Note that to accomplish this, we had to scale the PHOLD instance counter down by a factor of 16 for each rank doubling in order to offset the weak scaling built into the native PHOLD model. We then obtained the following results:

Table 2: Compute Time for Strong Scaling

MPI Ranks	Time (s)
2	21.915
4	10.814
8	5.8555
16	2.6580
32	1.4306





As we can see from the data, there is a strong inverse relationship between the number of MPI ranks and the total compute time. At first glance, this does not appear to make sense considering the results of the weak scaling experiment - but the crux of the problem is the massively altered proportion of MPI I/O between the two. In the weak scaling experiment, MPI I/O overhead increases proportionally to the number of MPI ranks. However, due to the design of the original PHOLD model, for the strong scaling experiment to guarantee constant problem size, the MPI I/O must have a limit on the amount of writes to a file (in other words, parallel output for parallel work). This effectively drops the linear MPI I/O overhead out of the equation, leaving behind the PHOLD simulation to be performed. This is massively parallelizable, and we can clearly see that this is the case by the strength of the inverse relationship between number of MPI ranks and compute time.

This is additionally supported by the the linear % speedup graph for the strong scaling experiment. The slight decrease in slope towards the right side of the graph can be explained by the MPI_Collect_All

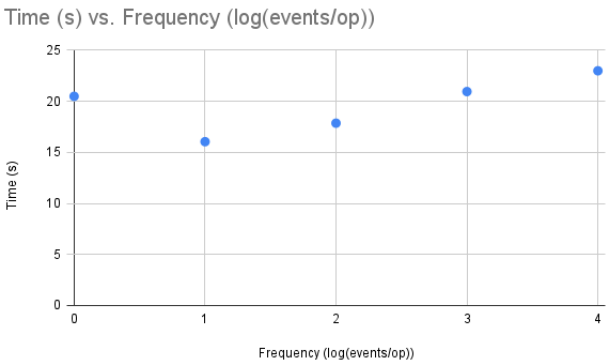
operation at the end of the PHOLD simulation - as the number of ranks increases past a certain point, the overhead for collecting the data from these ranks outweighs the diminishing returns of cutting the number of operations per rank in half. Overall, the results from the strong scaling experiment are much better than those of the weak scaling experiment, demonstrating that MPI I/O overhead is a strong limiting factor in the performance of a parallelized network.

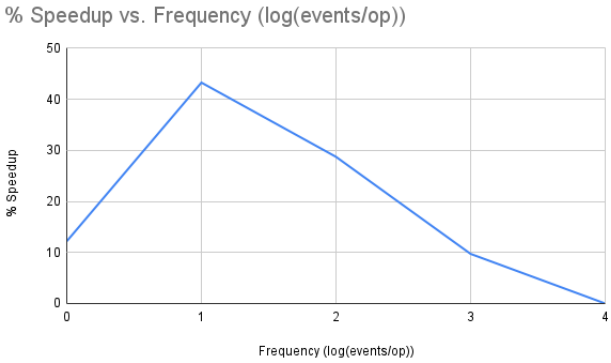
5.3 I/O Operation Frequency Experiment

To further explore additional factors of overall execution time, we performed a study on MPI I/O operation frequency. By varying the operation frequency on a $\log_{10}$ scale we were able to directly observe the overhead associated with rank-by-rank MPI I/O operations. We obtained the following results:

Table 3: Compute Time for I/O Operation Frequency

log Frequency	Time (s)
4	22.988
3	20.952
2	17.857
1	16.045
0	20.487





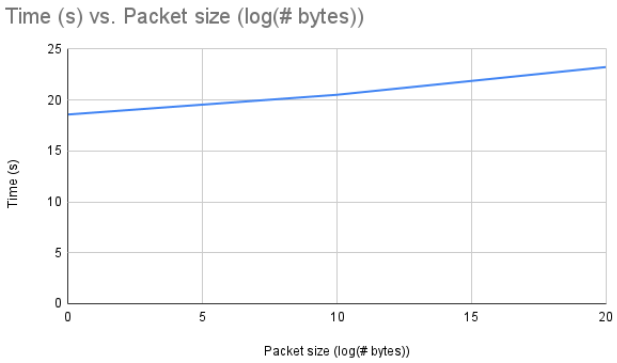
As one might expect, there exists a log-linear relationship between I/O operation frequency and compute time. Our group believes that a compute time of 20.487 seconds for an I/O frequency of 10 is an outlier, likely caused by external demand on the AiMOS compute node. Ultimately, we were unsurprised to find that fewer MPI I/O operations lead to faster compute times.

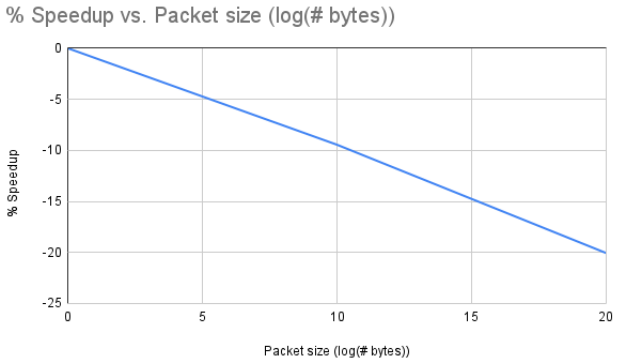
5.4 Packet Size Experiment

Our group capped off our analysis by doing a brief experiment on the effect of packet size on compute speed. For this we fixed our problem at 2 MPI ranks and default PHOLD parameters with an I/O frequency of 100. In doing so we hoped to gain insight as to whether packet size or MPI I/O operation frequency had a larger impact on the computational overhead of a hypothetical network. Our findings were as follows:

Table 4: Compute Time for Packet Size

log Frequency	Time (s)
1	18.575
10	20.513
20	23.236





Comparing the graphs of the I/O operation frequency experiment and the packet size experiment, we see that the slopes of the I/O operation frequency graphs are larger. We can then conclude that I/O operation frequency has a larger impact on the computational load on our PHOLD model network.

6 RELATED WORK

In the world of high-performance computing, optimizing MPI I/O operations for enhanced simulation performance is pivotal. Julian Martin Kunkel's exploration into the performance validation of MPI-IO implementations provides a foundational understanding of how simulations can predict MPI performance issues and inefficiencies. Kunkel highlights that the complexity of collective MPI operations often results in performance degradation due to algorithm inefficacies and run-time exceptions, which simulation can help predict and mitigate [5].

Building on the framework of simulation to enhance MPI I/O operations, our study draws inspiration from Ning Liu et al., who delve into burst buffers' role in leadership-class storage systems. They propose integrating a tier of solid-state burst buffers to manage peak I/O demands efficiently, a method confirmed to enhance the perceived throughput of applications dramatically [7]. This approach aligns with our group's exploration into using burst buffers within the ROSS and CODES simulation environments, where we aim to identify the optimal configurations that reduce I/O bottlenecks while scaling up simulation fidelity.

Our research leverages the ROSS optimistic simulation system to project and enhance the MPI I/O performance across varying scales of system architecture, integrating insights from both Kunkel's and Liu's methodologies. The burst buffer strategy, as modeled in our simulations, aligns with Liu's demonstrated benefits, showing significant throughput improvements without compromising data durability and access speeds.

Furthermore, our implementation extends the CODES framework to incorporate real-time dynamic adjustments of I/O operations, inspired by observed patterns in bursty I/O behaviors that Liu et al. detailed. This integration allows for a more granular analysis of I/O patterns, enhancing the predictive capabilities of our simulations regarding MPI I/O performance in large-scale systems as described in [10].

In conclusion, our research synthesizes the theoretical underpinnings provided by Kunkel with the practical implementations and enhancements suggested by Liu et al., facilitated by the advanced simulation capabilities of ROSS and CODES. This blend of theory and practical application aims to contribute to the broader field of HPC by providing a robust framework for simulating and improving MPI I/O operations in an era where computational demands are ever-increasing.

ACKNOWLEDGMENTS

We would like to express our gratitude to Professor Carothers and Teaching Assistants for their help. This project would not have been possible without the guidance of Professor Carothers. His insightful directions helped us navigate the complexities of Rensselaer's Optimistic Simulation System and ensured we were on the right track throughout the process. We are also grateful to the Teaching Assistants for their support. Their willingness to answer our questions and provide guidance when we were in need was instrumental. Without their help, we wouldn't have overcame the challenges. Finally, we would like to thank Rensselaer Polytechnic Institute and IBM for maintaining AiMOS, which facilitated the extensive testing during our project.

REFERENCES

[1] Amir Bahmani and Frank Mueller. 2014. Scalable performance analysis of exascale MPI programs through signature-based clustering algorithms. In *Proceedings of the 28th ACM International Conference on Supercomputing* (Munich, Germany) *(ICS '14)*. Association for Computing Machinery, New York, NY, USA, 155–164. https://doi.org/10.1145/2597652.2597676

[2] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. 1994. GTW: a time warp system for shared memory multiprocessors. In *Proceedings of Winter Simulation Conference*. 1332–1339. https://doi.org/10.1109/WSC.1994.717527

[3] Augustin Degomme, Arnaud Legrand, George S. Markomanolis, Martin Quinson, Mark Stillwell, and Frédéric Suter. 2017. Simulating MPI Applications: The SMPI Approach. *IEEE Transactions on Parallel and Distributed Systems* 28, 8 (2017), 2387–2400. https://doi.org/10.1109/TPDS.2017.2669305

[4] Wolfgang E. Denzel, Jian Li, Peter Walker, and Yuho Jin. 2010. A Framework for End-to-End Simulation of High-performance Computing Systems. *SIMULATION* 86, 5-6 (2010), 331–350. https://doi.org/10.1177/0037549709340840 arXiv:https://doi.org/10.1177/0037549709340840

[5] Julian Martin Kunkel. 1970. Using simulation to validate performance of MPI(-IO) implementations. https://link.springer.com/chapter/10.1007/978-3-642-38750-0_14

[6] Alexey Kupriyanov, Dmitrij Kissler, Frank Hannig, and Jürgen Teich. 2007. Efficient event-driven simulation of parallel processor architectures. In *Proceedingsof the 10th International Workshop on Software & Compilers for Embedded Systems* (Nice, France) *(SCOPES '07)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/1269843.1269854

[7] Ning Liu. [n. d.]. On the role of burst buffers in leadership-class storage systems. https://ieeexplore.ieee.org/

[8] Ioan Raicu, Ian T. Foster, Yong Zhao, Philip Little, Christopher M. Moretti, Amitabh Chaudhary, and Douglas Thain. 2009. The quest for scalable support of data-intensive workloads in distributed systems. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing* (Garching, Germany) *(HPDC '09)*. Association for Computing Machinery, New York, NY, USA, 207–216. https://doi.org/10.1145/1551609.1551642

[9] Mahdi Abed Salman, Cyrille Bertelle, and Eric Sanlaville. 2014. The Behavior of Load Balancing Strategies with Regard to the Network Structure in Distributed Computing Systems. In *2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems*. 432–439. https://doi.org/10.1109/SITIS.2014.42

[10] Yusuke Tanimura, Rosa Filgueira, Isao Kojima, and Malcolm Atkinson. 2012. Abstract: Reservation-Based I/O Performance Guarantee for MPI-IO Applications Using Shared Storage Systems. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. 1382–1383. https://doi.org/10.1109/SC.Companion.2012.203