End-to-end I/O Monitoring on a Leading Supercomputer

Bin Yang, *Shandong University, National Supercomputing Center in Wuxi;* Xu Ji, *Tsinghua University, National Supercomputing Center in Wuxi;* Xiaosong Ma, *Qatar Computing Research institute, HBKU;* Xiyang Wang, *National Supercomputing Center in Wuxi;* Tianyu Zhang and Xiupeng Zhu, *Shandong University, National Supercomputing Center in Wuxi;* Nosayba El-Sayed, *Emory University;* Haidong Lan and Yibo Yang, *Shandong University;* Jidong Zhai, *Tsinghua University;* Weiguo Liu, *Shandong University, National Supercomputing Center in Wuxi;* Wei Xue, *Tsinghua University, National Supercomputing Center in Wuxi*

https://www.usenix.org/conference/nsdi19/presentation/yang

This paper is included in the Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19).

February 26–28, 2019 • Boston, MA, USA

ISBN 978-1-931971-49-2

Open access to the Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19) is sponsored by

NetApp®

End-to-end I/O Monitoring on a Leading Supercomputer

Bin Yang[1,3], Xu Ji[2,3], Xiaosong Ma[4], Xiyang Wang[3], Tianyu Zhang[1,3], Xiupeng Zhu[1,3], Nosayba El-Sayed*[5], Haidong Lan[1], Yibo Yang[1], Jidong Zhai[2], Weiguo Liu[1,3], and Wei Xue†[2,3]

[1]Shandong University, [2]Tsinghua University, [3]National Supercomputer Center in Wuxi, [4]Qatar Computing Research institute, HBKU, [5]Emory University

Abstract

This paper presents an effort to overcome the complexities of production system I/O performance monitoring. We design Beacon, an end-to-end I/O resource monitoring and diagnosis system, for the 40960-node Sunway TaihuLight supercomputer, current ranked world No.3. Beacon simultaneously collects and correlates I/O tracing/profiling data from all the compute nodes, forwarding nodes, storage nodes and metadata servers. With mechanisms such as aggressive online+offline trace compression and distributed caching/storage, it delivers scalable, low-overhead, and sustainable I/O diagnosis under production use. Higher-level per-application I/O performance behaviors are reconstructed from system-level monitoring data to reveal correlations between system performance bottlenecks, utilization symptoms, and application behaviors. Beacon further provides query, statistics, and visualization utilities to users and administrators, allowing comprehensive and in-depth analysis without requiring any code/script modification.

With its deployment on TaihuLight for around 18 months, we demonstrate Beacon's effectiveness with real-world use cases for I/O performance issue identification and diagnosis. It has successfully helped center administrators identify obscure design or configuration flaws, system anomaly occurrences, I/O performance interference, and resource under- or over-provisioning problems. Several of the exposed problems have already been fixed, with others being currently addressed. In addition, we demonstrate Beacon's generality by its recent extension to monitor interconnection networks, another contention point on supercomputers. Both Beacon codes and part of collected monitoring data are released.[1]

1 Introduction

Modern supercomputers are networked systems with increasingly deep storage hierarchy, serving applications with growing scale and complexity. The long I/O path from storage media to application, combined with complex software stacks and hardware configurations, makes I/O optimizations increasingly challenging, both for application developers and supercomputer administrators. In addition, since I/O utilizes heavily shared system components (unlike computation or memory accesses), it usually suffers substantial inter-workload interference, causing high performance variance [37, 44, 47, 55, 60, 71].

Online tools that can capture/analyze I/O activities and guide optimization are highly needed. They also need to provide I/O usage information and performance records to guide future systems' design, configuration, and deployment. To this end, several profiling/tracing tools and frameworks have been developed, including application-side (e.g., Darshan [31] and ScalableIOTrace [77]), backend-side (e.g., LustreDU [29], IOSI [50] and LIOProf [85]), and multi-layer tools (e.g., Modular Darshan [70] and GUIDE [91]).

These proposed tools, however, suffer one or more of the following limitations. Application-oriented tools often require developers to instrument their source code or link extra libraries. They also do not offer intuitive ways to analyze inter-application I/O performance behaviors such as interference issues. Backend-oriented tools can collect system-level performance data and monitor cross-application interactions, but have difficulty in identifying performance issues for specific applications and in finding their root causes. Finally, problematic applications issuing *inefficient I/O requests* escape the radar of backend-side analytical methods [50, 52] relying on high-bandwidth applications.

This paper reports our design, implementation, and deployment of a light-weight, end-to-end I/O resource monitoring and diagnosis system, Beacon, for TaihuLight, currently the world's No.3 supercomputer [75]. It works with TaihuLight's 40,960 compute nodes (over ten-million cores in total), 288 forwarding nodes, 288 storage nodes, and 2 metadata nodes. Beacon integrates frontend tracing and backend profiling into a seamless framework, enabling tasks such as automatic per-application I/O behavior profiling, I/O bottleneck/interference analysis, and system anomaly detection.

To our best knowledge, this is the first system-level, multi-layer monitoring and real-time diagnosis framework deployed on ultra scale supercomputers. Beacon collects per-

*Most work conducted at Qatar Computing Research Institute.

†Wei Xue is the corresponding author. Email: xuewei@tsinghua.edu.cn

[1]Github link: https://github.com/Beaconsys/Beacon

formance data simultaneously from different types of nodes (including compute, I/O forwarding, storage, and metadata nodes) and analyzes them collaboratively, *without requiring any involvement of application developers*. Its elaborated collection scheme and aggressive compression minimize the system cost: only 85 *part-time* servers to monitor the entire 40960-node system, with $< 1\%$ performance overhead on user applications.

We have deployed Beacon for production use since April, 2017. It has helped the TaihuLight system administration and I/O performance team to identify several performance degradation problems. With its rich I/O performance data collection and real-time system monitoring, Beacon successfully exposes the mismatch between application I/O patterns and widely-adopted underlying storage design/configurations. To help application developers and users, it enables detailed per-application I/O behavior study, with novel inter-application interference identification and analysis. Beacon also performs automatic anomaly detection. Finally, we have recently started to expand Beacon beyond I/O, to network switch monitoring.

Based on our design and deployment experience, we argue that having such end-to-end detailed I/O monitoring framework is a highly rewarding practice. Beacon's all-system-level monitoring decouples it from language, library, or compiler constraints, enabling monitoring data collection and analysis for all applications and users. Much of its infrastructure reuses existing server/network/storage resources, and it has proven to bring with it negligible overhead. In exchange, users and administrators harvest deep insights into the complex I/O system components' operation and interaction, and save both human resources and machine core-hours wasted on unnecessarily slow/jittery I/O, or system anomalies.

2 Background: TaihuLight Network Storage

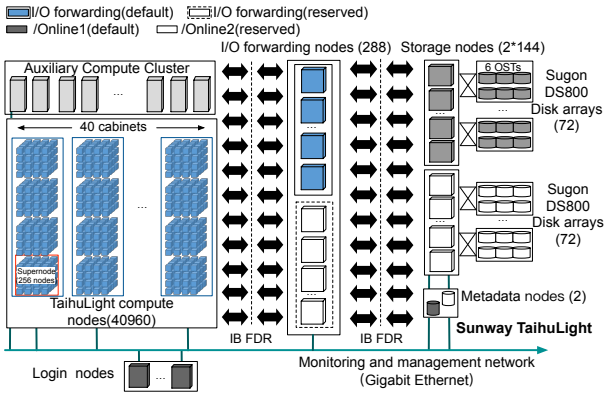


Figure 1: TaihuLight and its Icefish storage system architecture overview. Beacon uses the separate monitoring and management Ethernet network shown at the bottom.

We first introduce the TaihuLight supercomputer (and its Icefish I/O subsystem), where we performed our implementation and deployment. Though the rest of our discussion will be based on this specific platform, many aspects of Beacon's design and operation can be applied to other large-scale supercomputers or clusters.

TaihuLight is currently the world No.3 supercomputer, a many-core accelerated 125-petaflop system [36]. Figure 1 illustrates its architecture, highlighting the Icefish storage subsystem. The 40,960 260-core compute nodes are organized in 40 cabinets, each containing 4 supernodes. Through dual-rail FDR InfiniBand, all the 256 compute nodes in one supernode are fully connected and then connected to Icefish via a Fat-tree network. In addition, Icefish serves an Auxiliary Compute Cluster (ACC) with Intel Xeon processors, mainly used for data pre- and post-processing.

The Icefish backend employs the Lustre parallel file system [26], with an aggregate capacity of 10 PB on top of 288 storage nodes and 144 Sugon DS800 disk enclosures. An enclosure contains 60 1.2 TB SAS HDD drives, composing 6 OSTs, each an 8+2 RAID6 array. The controller within each enclosure connects to two storage nodes, via 2 fiber channels for path redundancy. Therefore every storage node manages 3 OSTs, while the two adjacent storage nodes sharing a controller form a failover pair.

Between the compute nodes and the Lustre backend is a layer of 288 *I/O forwarding nodes*. Each plays a dual role, both as a LWFS (Lightweight File System) based on Gluster [6] server to the compute nodes and client to the Lustre backend. This I/O forwarding practice is adopted by multiple other platforms that operate at such scale [15, 28, 54, 78, 90].

A forwarding node provides a bandwidth of 2.5 GB/s, aggregating to over 720 GB/s for the entire forwarding system. Each backend controller provides about 1.8 GB/s, amounting to a file system bandwidth of around 260 GB/s. Overall Icefish delivers 240 GB/s and 220 GB/s aggregate bandwidths for reads and writes, respectively.

TaihuLight debuted on the Top500 list in June 2016. At the time of this study, Icefish was equally partitioned into two namespaces: Online1 (for everyday workloads) and Online2 (reserved for ultra-scale jobs that occupy the majority of the compute nodes), with disjoint sets of forwarding nodes. A batch job can only use either namespace. I/O requests from a compute node are served by a specified forwarding node using a static mapping strategy for easy maintenance (48 fixed forwarding nodes for ACC and 80 fixed forwarding nodes for Sunway compute nodes).

Therefore the two namespaces, along with statically partitioned backend resources, are currently utilized separately by routine jobs and "VIP" jobs. One motivation for deploying an end-to-end monitoring system is to analyze the I/O behavior of the entire supercomputer's workloads and to design more flexible I/O resource allocation/scheduling mechanisms. For example, motivated by the findings of our monitoring system, a dynamic forwarding allocation system [43] for better forwarding resource utilization has been developed and test deployed.

3 Beacon Design and Implementation

3.1 Beacon Architecture Overview

Figure 2 outlines the working of Beacon on top of the Icefish I/O architecture, designed to operate at different levels of distribution for both scalability and ease of management.

Beacon performs I/O monitoring at all five Icefish components: the LWFS client (on compute nodes), the LWFS server and Lustre client (both on forwarding nodes), the Lustre server (on storage nodes), and the Lustre metadata server (on metadata nodes). At each of these *monitoring points*, Beacon deploys lightweight monitoring daemons that collect I/O-relevant events, status, and performance data locally, then transmit data for aggregation. Aggressive first-pass compression is conducted on all compute nodes for efficient per-application I/O trace collection/storage.

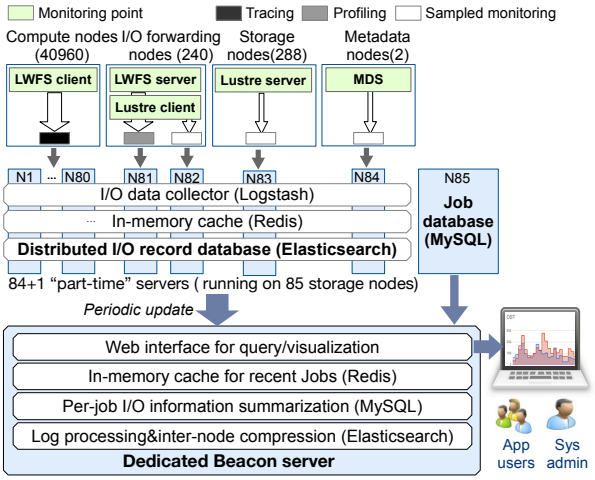


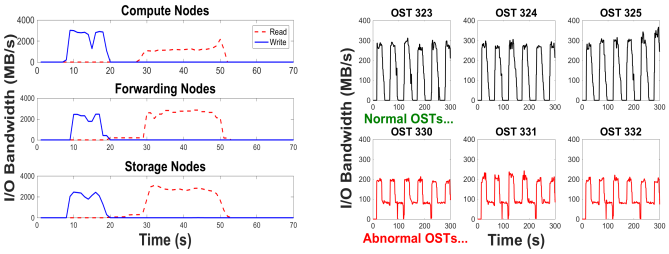
Figure 2: Beacon's main components: daemons at monitoring points, distributed I/O record database, job database, plus a dedicated Beacon server. The different width of arrows into and out from a module indicates *data compression.*

Beacon has its major backend processing and storage workflow distributed to part of the storage nodes on their node-local disks, utilizing hardware resources and parallelism. To this end, Beacon divides the 40,960 compute nodes into 80 groups and enlists 80 of the 288 storage nodes to communicate with one group each. Two more storage nodes are used to collect data from the forwarding nodes, plus another for storage nodes and one last for MDS. Together, these 84 "part-time" servers (shown as "N1" to "N84" in Figure 2) are called *log servers*, which host a *distributed I/O record database* of Beacon. The numbers of such servers were selected empirically considering Icefish's peak monitoring data processing workload.

These log servers adopt a layered software architecture built upon mature open-source frameworks. They collect I/O-relevant events, status and performance data through Logstash [9], a server-side log processing pipeline for simultaneously ingesting data from multiple sources. The data are then imported to Redis [16], a widely-used in-memory data store, acting as a cache to quickly absorb monitoring output. Persistent data storage and subsequent analysis are done via Elasticsearch [5], a distributed, lightweight search and analytics engine supporting a NoSQL database. It also supports efficient Beacon queries, for real-time and offline analysis.

One more storage node (N85 in Figure 2) is used to host Beacon's *job database* (implemented using MySQL [11]), which interacts with the job queuing system and keeps track of per-job information obtained by Beacon.

Finally, Beacon processes and presents its monitoring results to users (either system administrators or application users) using a *dedicated Beacon server*. There it performs two kinds of offline data analysis periodically: (1) second-pass, inter-node compression to further remove data redundancy by comparing and combining logs from compute nodes running the same job, and (2) extracting and caching in MySQL using SQL views the per-job statistic summary, while generating and caching in Redis common performance visualization results, to facilitate speedy user response. Log and monitoring data, after the two-pass compression, are permanently stored using Elasticsearch on this dedicated Beacon server. [2] Considering the typical daily data collection size of 10-100 GB, its 120 TB RAID5 capacity far exceeds the system's lifetime storage space needs.



(a) For user: per-job I/O performance (b) For admin: OST anomaly detection

Figure 3: Sample display from Beacon's web interface: (a) cross-layer read/write bandwidth of one user job, (b) bandwidth of three OSTs identified as undergoing anomaly.

On top of its Elasticsearch-MySQL-Redis stack, Beacon's web interface provides users with a friendly GUI for I/O-related job/system information query processing and visualization. For instance, application users could query a summary of their programs' I/O behavior based on job ID, along the entire I/O path, to help diagnosing I/O performance problems; system administrators can monitor real-time load levels on all forwarding, storage nodes and metadata servers, facilitating future job scheduling optimizations and center-level resource allocation policies. Figure 3 shows corresponding screenshots. Section 4 provides more details with concrete case studies.

All communication among Beacon entities uses a low-cost, easy-to-maintain Ethernet connection (marked in green in Figure 1), separate from *both* the main computation and

[2] Data in the *distributed I/O record database* are kept for 6 months.

the storage interconnects.

3.2 Multi-layer I/O Monitoring

Compute nodes On each of the 40,960 compute nodes, Beacon collects LWFS client trace logs. Each log entry contains the node's IP, I/O operation type, file descriptor, offset, request size, and timestamp.

On a typical day, such raw trace data alone amounts to over 100 GB, making their collection/processing a non-trivial task on Beacon's I/O record database, which takes away resources from the storage nodes. However, there exists abundant redundancy in HPC workloads' I/O operations. For example, since each compute node is usually dedicated to one job at a time, the job IDs are identical among many trace entries. Similarly, due to the regular, tightly coupled nature of many parallel applications, adjacent I/O operations likely have common components such as target file, operation type, and request size. Recognizing this, Beacon performs aggressive online compression on each compute node to dramatically reduce the I/O trace size. This is done by a simple, linear algorithm comparing adjacent log items and combining those with identical operation type, file descriptor, and request size, and accessing *contiguous* areas. These log items are replaced with a single item plus a counter. Considering the low computing overhead, we perform such parallel first-pass compression on compute nodes.

Beacon conducts offline log processing and second-pass compression on the dedicated server. Here it extracts the feature vector *<time, operation, file descriptor, size, offset>* from the original log records, and performs inter-node compression by comparing feature vector lists from all nodes and merging identical vectors, using a similar approach as in block trace modeling [74] or ScalaTrace [61].

The compute-node-side first-pass compression reduces the raw trace size by a factor of **5.4** to **34.6** across 8 real-world, large-scale applications, where the gain relies on the amount of "immediate" redundancy in an application's I/O operations. The second-pass compression on the dedicated Beacon server further delivers a 2- to 4-fold reduction. Detailed results are given in Appendix A.

Forwarding nodes On each forwarding node, Beacon profiles both the LWFS server and Lustre client. It collects the latency and processing time for each LWFS server request, and the request queue length for each LWFS server (by sampling the queue status once per 1000 requests). Rather than saving the per-request traces, the Beacon daemon periodically processes new traces and only saves I/O request statistics such as latency and queue length distribution.

For the Lustre client, Beacon collects request statistics by sampling the status of all outstanding RPC requests, once every second. Each sample contains the forwarding ID and RPC request size to the Lustre server.

Storage nodes and MDS On the storage nodes, Beacon daemons periodically sample the Lustre OST status table, record data items such as the OST ID and OST total data size, and further send high-level statistics such as count of RPC requests and average per-RPC data size in the past time window. On the Lustre MDS, Beacon also periodically collects and records statistics on active metadata operations (such as *open* and *lookup*) at 1-second intervals, while storing a summary of the periodic statistics in its database.

3.3 Multi-layer I/O Profiling Data Analysis

All the aforementioned monitoring data are transmitted for long-term storage and processing at the Elasticsearch-based database on the dedicated Beacon server as JSON objects, on top of which Beacon builds I/O monitoring/profiling services. These include automatic anomaly detection that runs periodically, as well as query and visualization tools that supercomputer users and administrators could use interactively. Below we give more detailed descriptions.

Automatic anomaly detection Outright failures are relatively easy to detect in a large system, commonly handled by tools such as heartbeat detection [67, 72], and is beyond the scope of this work. However, alive yet very slow components, such as forwarding nodes and OSTs under performance degradation, may continue to serve requests, but at a much lower pace that drags down entire applications' performance and reduces overall system utilization. With a busy storage system serving multiple platforms and on each many concurrent applications, such stragglers are rather hard to be identified. Assisted by its continuous, end-to-end I/O monitoring, Beacon enables automatic I/O system anomaly detection, identifying system components processing I/O workload at a significantly slower pace than their peers.

This is done by processing I/O monitoring data from the current and historical execution(s) of the same application, using clustering to detect apparent performance degradation on forwarding nodes and OSTs. The frequency of running such detection processing is configurable and is currently set at once every hour. Upon the identification of a serious system anomaly, an alarm email will be automatically generated and sent to TaihuLight administrators. We give a use case study in Section 4.2, plus detailed workflow description in Appendix B.

Per-job I/O performance analysis Upon a job's completion, Beacon performs automatic analysis of its I/O monitoring data collected from all layers. It performs inter-layer correlation by first identifying jobs from the job database that run on given compute node(s) at the log entry collection time. The involved forwarding nodes, leading to relevant forwarding monitoring data, are then located via the compute-to-forwarding node mapping using a system-wide mapping table lookup. Finally, relevant OSTs and corresponding storage node monitoring data entries are found by file system lookup using the Lustre command `lfs`.

From the above data, Beacon derives and stores coarse-grained information for quick query, including average and

peak I/O bandwidth, average IOPS, runtime, number of processes (and compute nodes) performing I/O, I/O mode, total count of metadata operations, and average metadata operations per second during I/O phases. Among them, the *I/O mode* indicates the parallel file sharing mode among processes, where common modes include "N-N" (each compute process accesses a separate file), "N-1" (all processes share one file), "N-M" (N processes perform I/O aggregation to access M files, M<N), and "1-1" (only one of all processes performs sequential I/O on a single file).

To help users understand/debug their applications' I/O performance, Beacon provides web-based I/O data visualization. This diagnosis system can be queried using a job ID, after appropriate authentication, and allows visualizing the I/O statistics of the job, both real-time and post-mortem. It reports the measured I/O metrics (such as bandwidth and IOPS) and inferred characteristics (such as the number of I/O processes and I/O mode). Users are also presented with user-configurable visualization tools, showing time-series measurement in I/O metrics, statistics information such as request type/size distribution, and performance variances. Our powerful I/O monitoring database allows further user-initiated navigation such as per-compute-node traffic history, and zooming control to examine data at different granularity. For security/privacy, users are only allowed to view I/O data from compute, forwarding, and storage nodes involved in and for the duration of their jobs' execution.

I/O subsystem monitoring for administrators Beacon also provides administrators with the capability of monitoring the I/O status for any time period, on any node.

Besides all the user-visible information and facilities mentioned above, the administrators can further obtain and visualize: (1) detailed I/O bandwidth and IOPS for each compute node, forwarding node, and storage node, (2) resource utilization status of forwarding nodes, storage nodes and the MDS, including detailed request queue length statistics, and (3) I/O request latency distribution on forwarding nodes. Additionally, Beacon authorizes administrators with direct I/O record database access, to facilitate in-depth analysis.

Combining such facilities, administrators could perform powerful and thorough I/O traffic and performance analysis, e.g., by checking multi-level traffic, latency, and throughput monitoring information regarding a certain job execution.

3.4 Generality and Limitations

Beacon can be adopted by other platforms. The I/O forwarding architecture is widely used, by 9 out of the current Top 20 machines (listed in Table 1). It is also targeted by the DAOS Exascale storage design [54] and the TOKIO I/O monitoring framework [24].

Beacon's building blocks, such as operation log collection and compression, scheduler-assisted per-application data correlation and analysis, history-based anomaly identification, automatic I/O mode detection, and built-in interference analysis, can all be performed on other supercomputers. Its data management components, such as Logstash, Redis, and ElasticSearch, are open-source software that will run on these machines as well. Our forwarding layer design validation and load analysis could also help recent platforms with a layer of burst buffer nodes, such as NERSC's Cori [2].

Rank	Machine	Vendor	File system
3	Taihulight [19]	NRCPC	Lustre
4	Tianhe-2A [87]	NUDT	Lustre+H2FS
5	Piz Daint [15]	Cray	Lustre+GPFS
6	Trinity [21]	Cray	Lustre
9	Titan [20]	Cray	Lustre
10	Sequoia [17]	IBM	Lustre
12	Cori [1]	Cray	Lustre+GPFS
14	Oakforest-PACS [12]	Fujitsu	Lustre
18	K computer [8]	Fujitsu	FEFS [65]

Table 1: I/O forwarding adopters among top-20 supercomputers, as of November 2018

Meanwhile, the current Beacon system has limitations that can be addressed in future work or application to other platforms. For example, it currently performs data analysis and detects anomalies by bringing unusual patterns to the attention of system administrators. Additional historical data collection to correlate symptoms and solutions would make the process more intelligent and reduce human labor requirement [86]. Similarly, application users who are not parallel I/O experts could benefit from system-generated direct suggestions (such as for I/O mode or request size change, and against using the parallel file system for metadata-heavy interactive tasks), beyond performance data visualization.

4 Beacon Use Cases

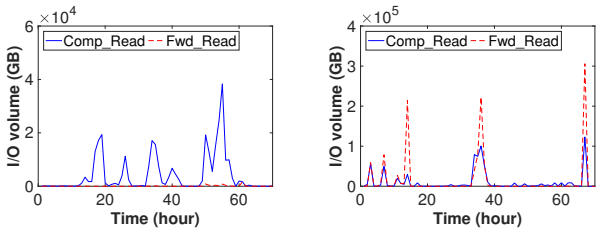
Beacon has been deployed on TaihuLight for around 18 months and has gathered massive I/O information. So far it has accumulated around 10 TB of trace data (after two passes of compression). This history contains 116,765 jobs that used at least 32 compute nodes, consuming 323,951,208 core-hours in total. 28,330 (24%) of these jobs featured non-trivial I/O, with per-job I/O volume over 200 MB.

The insights and issues revealed by Beacon's monitoring and diagnosis have already helped TaihuLight administrators fix several design flaws, develop a dynamic and automatic forwarding node allocation tool, and improve system reliability/consistency plus application efficiency. Due to space limit, we focus on three types of use cases: (1) Performance issue diagnosis, (2) Automatic I/O anomaly diagnosis, and (3) Application and user behavior analysis.

4.1 Performance Issue Diagnosis

Forwarding node cache thrashing Beacon's end-to-end monitoring facilitates *cross-layer correlation of I/O profiling data*, at different temporal or spatial granularity. By comparing the total request volume at each layer, Beacon has helped TaihuLight's infrastructure management team in identifying a previously unknown performance issue, as detailed next.

A major reason driving the adoption of I/O forwarding or burst buffer layer is the opportunity to perform prefetching, caching, and buffering, to reduce the pressure on slower disk storage. Figure 4 shows the read volume on compute and forwarding node layers, during two sampled 70-hour periods in August 2017. Figure 4(a) shows a case with expected behavior, where the total volume requested by the compute nodes is significantly higher than that requested by the forwarding nodes, signaling good access locality and effective caching. Figure 4(b), however, tells the opposite story (that surprised system administrators): the forwarding layer could incur much higher read traffic from the backend than requested by user applications, reading much more data from the storage nodes than returning to compute nodes. Such situation does not apply to *writes*, where Beacon always shows matching aggregate bandwidth across the two levels.



(a) 70 hours starting from 08/14/2017 (b) 70 hours starting from 08/01/2017

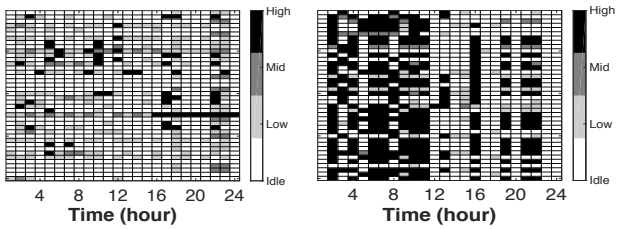
Figure 4: Sample segments of TaihuLight read volume history, each collected at two layers

Further analysis of the applications executed and their assigned forwarding nodes during the problem period in Figure 4(b) reveals an unknown *cache thrashing* problem, caused by the N-N sequential data access behavior. By default, the Lustre client has a 40 MB read-ahead cache for each file. Under the N-N sequential read scenarios, such aggressive prefetching causes severe memory contention, with data repeatedly read from the backend (and evicted on forwarding nodes). E.g., an 1024-process `Shentu` [25] execution has each I/O process read an 1-GB single file, incurring a 3.5× I/O amplification at the Lustre backend of Icefish. This echos previous finding on the existence of I/O self-contention within a single application [55].

Solution This problem can be addressed by adjusting the Lustre prefetching cache size per file. For example, changing it from 40MB per file to 2MB is shown to remove the thrashing. Automatic, per-job forwarding node cache reconfiguration, which leverages real-time Beacon monitoring results, is currently under development for TaihuLight. Alternatively, switching the application from an N-N to N-M mode (performing *I/O aggregation*, by having each set of N/M compute processes group their I/O to one file) also eliminates cache thrashing, and brings 3× I/O performance improvement. Given the close collaboration between application teams and machine administrators, making performance-critical program changes as suggested by monitoring data analysis is an accepted practice on leading supercomputers.

Bursty forwarding node utilization Beacon's continuous end-to-end I/O monitoring gives center management a global picture on system resource utilization. While such systems were often built and configured using rough estimates based on past experience, Beacon collects detailed resource usage history to help both in improving the current system's efficiency and in assisting future system upgrade and design.

Figure 5 gives one example, again on forwarding load distribution, by showing two one-day samples from July 2017. Each row portraits the by-hour peak load on one of the same 40 forwarding nodes randomly sampled from the 80 active ones. The darkness reflects the maximum bandwidth reached within that hour. The labels "high", "mid", "low", and "idle" correspond to that maximum residing in the >90%, 50-90%, 10-50%, or 0-10% interval (relative to the benchmarked per-forwarding-node peak bandwidth), respectively.



(a) Regular load (07/01/2017) (b) Heavy load (07/28/2017)

Figure 5: Sample TaihuLight one-day load summary, showing peak load level by hour, across 40 randomly sampled forwarding nodes

Figure 5(a) shows the more typical load distribution, where the majority of forwarding nodes stay lightly used for the vast majority of time (90.7% of cells show maximum load being under 50% of peak bandwidth). Figure 5(b) gives a very different picture, with a significant set of sampled forwarding nodes serving I/O-intensive large jobs for a good part of the day. 35.7% of the cells actually see a maximum load of over 99% of peak forwarding node bandwidth.

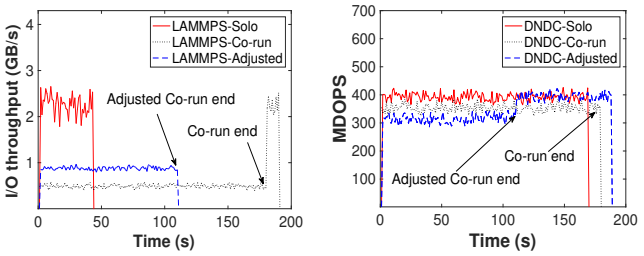
These results indicate that (1) overall there is forwarding resource overprovisioning (confirming prior findings [41,52, 57,64]), (2) even with the more representative low-load scenarios, it is not rare for forwarding node bandwidth to be saturated by application I/O, and (3) load imbalance across forwarding nodes exists regardless of load level, presenting idle resources potentially helpful to I/O-intensive applications.

Solution Recognizing the above, recently TaihuLight has enlisted more of its "backup forwarding nodes" into regular service. Meanwhile, a dynamic, application-aware forwarding node allocation scheme is designed and partially deployed (turned on for a subset of applications) [43]. Leveraging application-specific job history information, such an allocation scheme is intended to replace the default, static mapping between compute and forwarding nodes.

MDS request priority setting While overall we found that most TaihuLight jobs are rather metadata-light, Beacon

does observe a small fraction of parallel jobs (0.69%) with high metadata request rate (more than 300 metadata operations/s on average during I/O phases). Beacon found that these metadata-heavy ("high-MDOPS") applications tend to cause significant I/O performance interference. Among jobs with Beacon-detected I/O performance anomaly, those sharing forwarding nodes with high-MDOPS jobs experienced an average 13.6× increase in read/write request latency during affected time periods.

Such severe delay and corresponding Beacon forwarding node queue status history prompted us to examine the TaihuLight LWFS server policy. We found that metadata requests were given priority over file I/O, based on the single-MDS design and the need to provide fast response to interactive user operations such as `ls`. Here, as neither disk bandwidth nor metadata server capacity was saturated, such interference could easily remain undetected using existing approaches that focus on I/O-intensive workloads only [37, 52].



(a) LAMMPS (b) DNDC

Figure 6: Impact of metadata operations' priority adjustment

Solution As a temporary solution, we added probabilistic processing across priority classes to the TaihuLight LWFS scheduling. Instead of always giving metadata requests high priority, an LWFS server thread now follows a $P:(1-P)$ split (P configurable) between picking the next request from the separate queues hosting metadata and non-metadata requests. Figure 6 shows the "before" and "after" pictures, with `LAMMPS` [34] (a classical molecular dynamics simulator with middle scale 256 compute nodes) running against the high-MDOPS `DNDC` [39] (a bio-geochemistry application for agro-ecosystems simulation). Throughput of their solo-runs, where each application runs by itself on an isolated testbed, is given as reference. With a simple equal probability split, `LAMMPS` co-run throughput doubles, while `DNDC` only perceives a 10% slowdown. For a long-term solution, we plan to leverage Beacon to automatically adapt the LWFS scheduling policies, considering operation types, the MDS load level, and application request scheduling fairness.

4.2 Automatic I/O Anomaly Diagnosis

In extreme-scale supercomputers, users typically accept jittery application performance, recognizing wide-spread resource sharing among jobs. System administrators, meanwhile, see different behaviors among system components with homogeneous configuration, but cannot tell how much of that difference comes from these components' functioning, and how much from the diversity of tasks they perform.

Beacon's multi-layer monitoring capacity, therefore, presents a new window for supercomputer administrators to examine system health, by connecting statistics on application-issued I/O requests all the way to that of individual OST's bandwidth measurement. Such connection guides Beacon to deduce what is considered the norm and what an exception. Leveraging this capability, we design and implement a lightweight, automatic anomaly detection tool to identify such *apparent exceptions* that signal significant performance degradation or faulty system components.

Application-driven anomaly detection Most I/O-intensive applications have distinct I/O phases, i.e., episodes in their execution where they perform I/O continuously, such as those to read input files during initialization or to write intermediate results or checkpoints. For a given application, such I/O phase behavior is often quite consistent. Taking advantage of such repeated I/O operations and its multi-layer I/O information collection, Beacon performs automatic I/O phase recognition, on top of which it conducts I/O-related anomaly detection. More specifically, larger applications (such as those using 1024 compute nodes or more) are spreading their I/O load to multiple forwarding nodes and backend nodes, giving us opportunities to directly compare the behavior of these servers processing requests *known to Beacon* as homogeneous or highly similar.

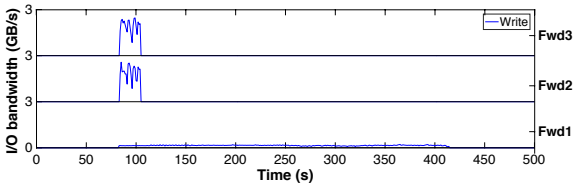


Figure 7: Forwarding bandwidth in a 6000-process `LAMMPS` run

Figure 7 gives an example of a 6000-process `LAMMPS` run with checkpointing. The 1500 compute nodes are assigned to 3 forwarding nodes, whose bandwidth and I/O time are reflected in the time-series data from Beacon. We can clearly see here the `Fwd1` node is a straggler in this case, serving at a bandwidth much slower than its peak (without answering to other applications). As a result, there is a 20× increase in the application-visible checkpoint operation time, estimated using the other two forwarding nodes' I/O phase duration.

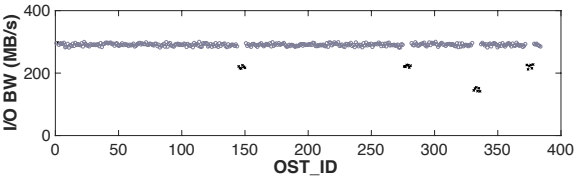


Figure 8: Per-OST bandwidth during a `Shentu` execution

Anomaly alert and node screening Such continuous, online application performance anomaly detection could iden-

tify forwarding nodes or backend units with deviant performance metrics, which in turn will trigger Beacon's more detailed monitoring and analysis. If it finds such a system component to consistently under-perform relative to peers serving similar workloads, with configurable thresholds in monitoring window and degree of behavior deviation, it reports this as an automatically detected system anomaly. By generating and sending an alarm email to the system administration team, Beacon prompts system administrators to do a thorough examination, where its detailed performance history information and visualization tools are also helpful.

Such anomaly screening is especially important for expensive, large-scale executions. For example, among all applications running on TaihuLight so far, the parallel graph engine `Shentu` [49] has the most intensive I/O load. It scales well to the entire supercomputer in both computation and I/O, with 160,000 processes and large input graphs distributed evenly to nearly 400 Lustre OSTs. During test runs preparing for its Gordon Bell bid in April 2018, Beacon's monitoring discovered a few OSTs significantly lagging behind in the parallel read, slowing down the initialization as a result (Figure 8). By removing them temporarily from service and relocating their data to other OSTs, `Shentu` cut its production run initialization time by 60%, saving expensive dedicated system allocation and power consumption. In this particular case, further manual examination attributed the problem to these OSTs' RAID controllers, which were later fixed.

Had it not been for Beacon's backend monitoring, applications like `Shentu` would have accepted whatever bandwidth they got, without suspecting I/O performance being abnormal. Similarly, had it not been for Beacon's routine frontend tracing, profiling, and per-application performance anomaly detection, it would not have noticed the backend outliers. As full-system benchmarking requires taking the supercomputer offline and cannot be regularly attempted, Beacon provides a much more affordable way for continuous system health monitoring and diagnosis, by coupling application-side and server-side tracing/profiling information.

Duration (hours)	Location of anomaly	
	Forwarding node (times)	OSS+OST (times)
(0,1)	23	31
[1,4)	14	17
[4,12)	5	9
[12,96)	3	6
≥96, manually verified	6	8

Table 2: Duration of Beacon-identified system anomalies

Beacon's deployment on TaihuLight started around April 2017, with features and tools incrementally developed and added to production use. Table 2 summarizes the automatically identified I/O system anomaly occurrences at the two service layers, from Apr 2017 to Aug 2018. Such identification adopted a minimum threshold of measured maximum bandwidth under 30% of the known peak value, as well as a minimum duration of 60 minutes. Such parameters can be configured to adjust the anomaly detection system sensitivity. Most performance anomaly occurrences are found to be transient, lasting under 4 hours.

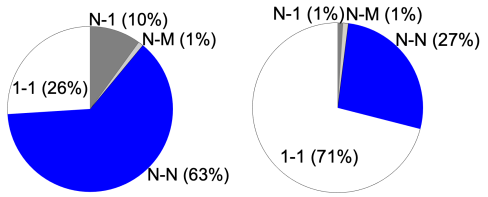
There are a total of 14 occasions of performance anomaly over 4 hours on the forwarding layer, and 23 on the backend layer, confirming the existence of fail-slow situations found common with data centers [42]. Reasons for such relatively long yet "self-healed" anomalies include service migration and RAID reconstruction. With our rather conservative setting during such initial deployment period, Beacon was set to send the aforementioned alert email when a detected anomaly situation lasted beyond 96 hours (except for large-scale production runs as in the `Shentu` example above, where the faulty units were immediately reported). With all these occasions, the Beacon detected anomaly was confirmed by human examination.

4.3 Application and User Behavior Analysis

With its powerful information collection and multi-layer I/O activity correlation, Beacon provides new capability to perform detailed application or user behavior analysis. Results of such analysis assist in performance optimization, resource provisioning, and future system design. Here we showcase several application/user behavior studies, some of which have brought corresponding optimizations or design changes to the TaihuLight system.

Type	$(0,1K]$	$(1K,10K]$	$(10K,100K]$	$(100K,1000K]$	$(1000K,\infty)$
Read	8.1 GB	101.0 GB	166.9 GB	1172.9 GB	2010.6 GB
Write	18.2 GB	83.9 GB	426.6 GB	615.9 GB	41458.8 GB

Table 3: Avg. per-job I/O volume by core-hour consumption



(a) Read (b) Write

Figure 9: Distribution of file access modes, in access volume

I/O mode	Avg. read volume	Avg. write volume	Job count
N-N	96.8 GB	120.1 GB	11073
N-M	36.2 GB	63.2 GB	324
N-1	19.6 GB	19.3 GB	2382
1-1	33.0 GB	142.3 GB	16251

Table 4: Avg. I/O volume and job count by I/O mode

Application I/O mode analysis First, Table 3 gives an overview of I/O volume across all profiled jobs with non-trivial I/O, categorized by per-job core-hour consumption. Here, 1000K core-hours correspond to a 10-hour run using 100,000 cores on 25,000 compute nodes, and jobs with such consumption or higher write more than 40 TB of data on average.[3] Overall, the amount of data read/written grows as

[3]Further examination reveals that in each core-hour category, average read/write volumes are influenced by a minority group of heavy consumers.

the jobs consume more compute node resources. The less resource-intensive applications tend to perform more reads, while the larger consumers are more write-intensive.

Figure 9 shows the breakdown of I/O-mode adoption among all TaihuLight jobs performing non-trivial I/O, by total read/write volume. The first impression one gets from these results is that the rather "extreme" cases, such as N-N and 1-1, form the dominant choices, especially in the case of writes. We suspected that this distribution could be skewed by a large number of small jobs doing very limited I/O, and calculated the average per-job read/write volume for each I/O mode. The results (Table 4) show that this is not the case. Actually, applications that choose to use 1-1 mode for writes actually have a much higher overall write volume.

The 1-1 mode is the closest to sequential processing behavior and is conceptually simple. However, it obviously lacks scalability and fails to utilize the abundant hardware parallelism in the TaihuLight I/O system. The wide presentation of this I/O mode might help explain the overall underutilization of forwarding resources, discussed earlier in Section 4.1. Echoing similar findings (though not so extreme) on other supercomputers [57] (including Intrepid [7], Mira [10] and Edison [4]), effective user education on I/O performance and scalability would both help improve storage system utilization and reduce wasted compute resources.

The N-1 mode has a different story. It is an intuitive parallel I/O solution that allows compute processes to directly read to or write from their local memory without gather-scatter operations, while retaining the convenience of having a single input/output file. However, our detailed monitoring finds it a quite damaging I/O mode that users should steer away from, as explained below.

First, our monitoring results confirm findings by existing research [23, 56] that the N-1 mode offers low application I/O performance (by reading/writing to a shared file). Even with a large N, such applications receive no more than 250 MB/s I/O aggregate throughput, despite the peak TaihuLight backend combined bandwidth of 260 GB/s. For read operations, users here also rarely modify the default Lustre stripe width, confirming behavior reported in a recent ORNL study [48]. The problem is much worse with writes, as performance severely degrades due to file system locking.

This study, however, finds N-1 applications to be extraordinarily *disruptive* as they harm all kinds of neighbor applications that share forwarding nodes with them, particularly when N is large (e.g., over 32 compute nodes).

The reason is that each forwarding node operates an LWFS server thread pool (currently sized at 16), providing forwarding service to assigned compute nodes. Applications using the N-1 mode tend to flood this thread pool with requests in bursts. Unlike with the N-N or N-M modes, N-1 suffers from the aforementioned poor backend performance by using a single shared file. This, in turn, makes N-1 requests slow to process, further exacerbating their congestion in the queue and delaying requests from other applications, even when those victims are accessing disjoint backend servers and OSTs.

Here we give a concrete example of I/O mode-induced performance interference, featuring the earthquake simulation `AWP` [35] (2017 Gordon Bell Prize winner) that started with N-1 mode. In this sample execution, it co-runs with the weather forecast application `WRF` [69] using the 1-1 mode, each having 1024 processes on 256 compute nodes. Under the "solo" mode, we assign each application a dedicated forwarding node in a small testbed partition of TaihuLight. In the "co-run" mode, we let them share one forwarding node (as the default compute-to-forwarding mapping is 512-to-1).

Operation	Avg. wait time	Avg. proc. time	Avg. queue length
`WRF` write (solo)	2.73 ms	0.052 ms	0.22
`WRF` write (co-run)	30.06 ms	0.054 ms	208.51
`AWP` read (solo)	58.17 ms	3.44 ms	226.37
`AWP` read (co-run)	58.18 ms	3.44 ms	208.51

Table 5: Performance interference during `WRF` and `AWP` co-run sharing a forwarding node

Table 5 lists the two applications' average request wait/processing time and forwarding node queue length during these runs. Note that with "co-run", the queue is shared by both applications. We find that the average wait time of `WRF` has been increased by $11\times$ when co-running, but `AWP` is not affected. This result reveals the profound malpractice of the N-1 file sharing mode and confirms prior finding that I/O interference is access-pattern-dependent [47, 53].

Solution Our tests confirm that increasing the LWFS thread pool size does not help in this case, as the bottleneck lies on the OSTs. Meanwhile, avoiding the N-1 mode has been advised in prior work [23, 84], as well as numerous parallel I/O tutorials. Considering our new inter-application study results, it is rather an obvious "win-win" strategy that simultaneously improves large applications' I/O performance and reduces their disruption to concurrent workloads. However, based on our experience with real applications, this message needs to be better promoted.

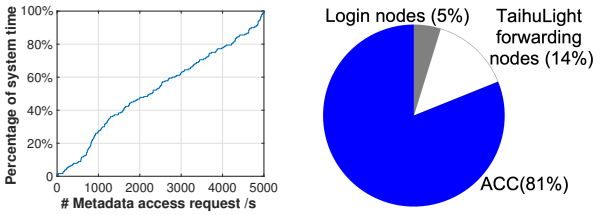
In our case, the Beacon developers worked with the `AWP` team to replace its original N-1 file read (for initialization/restart) with the N-M mode, during the 2017 Gorden Bell Prize final submission phase. This change produced an over 400% enhancement in I/O performance. Note that the GB Prize submission does not report I/O time; we found that `AWP`'s 130,000-process production runs spend the bulk of their execution time reading around 100 TB of input or checkpoint data. Significant reduction in this time greatly facilitated `AWP`'s development/testing and saved non-trivial supercomputer resources.

Metadata Server Usage Unlike forwarding nodes utilization (discussed earlier), the Lustre MDS is found with rather evenly distributed load levels by Beacon's continuous load monitoring (Figure 10(a)). In particular, in 26.8% of the time, the MDS experiences a load level (in requests per second) above 75% of its peak processing throughput.

Beacon allows us to further split the requests between systems sharing the MDS, including the TaihuLight forwarding nodes, login nodes, and the ACC. TaihuLight administrators were surprised to find that over 80% of the metadata access workload actually comes from the ACC (Figure 10(b)).

Note that the login node and ACC have their own local file systems, ext4 and GPFS [66], respectively, which users are encouraged to use for purposes such as application compilation and data post-processing/visualization. However, since the users are likely TaihuLight users too, we found most of them prefer to directly use the main Lustre scratch file system intended for TaihuLight jobs, for convenience. While the I/O bandwidth/IOPS resources consumed by such tasks are negligible, user interactive activities (such as compiling or post-processing) turn out to be metadata heavy.

Large waves of unintended user activities correspond to the most heavy-load periods at the tail end in Figure 10(a), and have led to MDS crashes that directly affected applications running on TaihuLight. According to our survey, many other machines, including 2 out of the top 10 supercomputers (Sequoia [17] and Sierra [18]), also have a single MDS, assuming that their users follow similar usage guidelines.



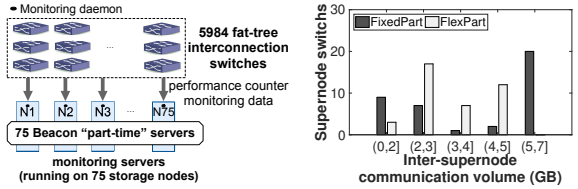
(a) CDF of MDS utilization (b) MDS load across platforms

Figure 10: TaihuLight Lustre MDS load statistics

Solution There are several potential solutions to this problem. With the help of Beacon, we can identify and remind users performing metadata-heavy activities to avoid using the PFS directly. Or we can support more scalable Lustre metadata processing with an MDS cluster. A third approach is to facilitate intelligent workflow support that automatically performs data transfer, based on users' needs. This third approach is the one we are currently developing.

4.4 Extension to network monitoring

Encouraged by Beacon's success in I/O monitoring, in summer 2018 we began to design and test its extension to monitor and analyze network problems, motivated by the network performance debugging needs of ultra large-scale applications. Figure 11(a) shows the architecture of this new module. Beacon samples performance counters on the 5984 Mellanox InfiniBand network switches, such as per-port sent and received volumes. Again the data collected are passed to low-overhead daemons on Beacon log servers, more specifically, 75 of its 85 part-time servers, each assigned 80 switches. Similar processing and compression are conducted, with result data persisted in Beacon's distributed database, then periodically relocated to its dedicated server for user queries and permanent storage.



(a) Overview of Beacon's network monitoring module (b) Distribution of communication volume inter-supernode

Figure 11: Network monitoring architecture and its use case

This Beacon network monitoring prototype was tested in time to help in the aforementioned `Shentu` [49] production runs, for its final submission to Supercomputing'18 as an ACM Gordon Bell Award finalist. Beacon was called upon to identify the reason of aggregate network bandwidth significantly lower than theoretical peak. Figure 11(b) illustrates this with a 3-supernode `Shentu` test run. The dark bars (FixedPart) form a histogram of communication volumes measured on 40 switches connecting these 256-node supernodes for inter-supernode communication, reporting the count of switches within 5 volume brackets. There is a clear bipolar distribution, showing severe load imbalance and more than expected inter-supernode communication. This monitoring result led to discovery that due to the existence of faulty compute nodes within each supernode, the fixed partitioning relay strategy adopted by `Shentu` led to a subset of relay nodes receiving twice the "normal" load. Note that `Shentu`'s own application-level profiling found communication volume across compute nodes very well balanced, hence the problem was not obvious to application developers until Beacon provided such switch-level traffic data.

Solution This finding prompted `Shentu` designers to optimize their relay strategy, using a topology-aware scholastic assignment algorithm to uniformly partition source nodes to relay nodes [49], whose results are shown by gray bars (FlexPart) in Figure 11(b). The peak per-switch communication volume is reduced by 27.0% (from 6.3 GB to 4.6 GB), with significantly improved load balance, bringing a total communication performance enhancement of around 30%.

5 Beacon Framework Evaluation

We now evaluate Beacon's per-application profiling accuracy, as well as its performance overhead.

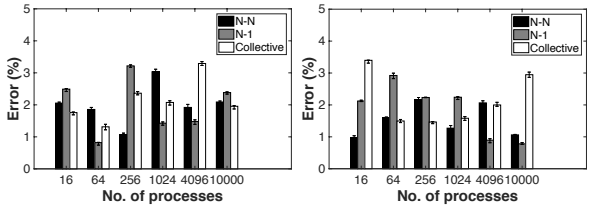
5.1 Accuracy Verification

Beacon collects full traces from the compute node side, thus has access to complete application-level I/O operation information. However, since the LWFS client trace interface provides only coarse timestamp data (at per-second granularity), and due to the clock drift across compute nodes, it is possible that the I/O patterns recovered from Beacon logs deviate from the application-level captured records.

To evaluate the degree of such errors, we compare the I/O throughput statistics reported by `MPI-IO Test` [40] to those by Beacon. In the experiments, we use `MPI-IO Test` to test different parallel I/O modes, including N-N and N-1 independent operations, plus MPI-IO library collective calls. 10 experiments were repeated at each execution scale.

The accuracy evaluation results are shown in Figure 12. We plot the average error in Beacon, measured as the percentage of deviation of the recorded aggregate compute node-side I/O throughput from the application-level throughput reported by the MPI-IO library.

We find Beacon able to accurately capture application performance, even for applications with non-trivial parallel I/O activities. More precisely, Beacon's recorded throughput deviates from `MPI-IO Test` reported values by only 0.78–3.39% (1.84% on average) for the read test and 0.81–3.31% (2.03% on average) for write, respectively. Results are similar with high-IOPS applications, omitted due to space limit.



(a) Read tests (b) Write tests

Figure 12: Average error rate of Beacon reported bandwidth (error bars show 95% confidence intervals.)

Beacon's accuracy can be attributed to that it records *all* compute node-side trace logs, facilitated by its efficient and nearly lossless compression method (described in Section 3.2). We find that even though individual trace items may be off in timestamps, data-intensive applications on supercomputers seldom perform isolated, fast I/O operations (which are not of interest for profiling purposes); instead, they exhibit I/O phases with sustained high I/O intensity. By collecting a large set of per-application I/O trace entries, Beacon is able to paint an accurate picture of an application's I/O behavior and performance.

5.2 Monitoring and Query Overhead

We now evaluate Beacon's monitoring overhead in a production environment. We compare the performance of important I/O-intensive real-world applications and the `MPI-IO Test` benchmark discussed earlier, with and without Beacon turned on (T_w and $T_{w/o}$, respectively). We report the overall run time of each program and calculate the slowdown introduced by turning on Beacon. Table 6 shows the results, listing the average slowdown measured from at least 5 runs for each program (variance of slowdown across runs very low: under 2%). Note that for the largest applications, such testing was piggybacked on actual production runs of stable codes, with Beacon turned on during certain allocation time frames. Applications like `AWP` often break their executions to run a certain number of simulation timesteps at a time.

Application	#Process	$T_{w/o}$ (s)	T_w (s)	%Slowdown
`MPI-IO`$_N$	64	26.6	26.8	0.79%
`MPI-IO`$_N$	128	31.5	31.6	0.25%
`MPI-IO`$_N$	256	41.6	41.9	0.72%
`MPI-IO`$_N$	512	57.9	58.4	0.86%
`MPI-IO`$_N$	1024	123.1	123.5	0.36%
`WRF`$_1$	1024	2813.3	2819.1	0.21%
`DNDC`	2048	1041.2	1045.5	0.53%
`XCFD`	4000	2642.1	2644.6	0.09%
`GKUA`	16384	297.5	299.9	0.82%
`GKUA`	32768	182.8	184.1	0.66%
`AWP`	130000	3233.5	3241.5	0.25%
`Shentu`	160000	5468.2	5476.3	0.15%

Table 6: Avg. Beacon monitoring overhead on applications

These results show that the Beacon tool introduces very low overhead: under 1% across all test cases. Also, the overhead does not grow with application execution scale, and actually appears smaller (below 0.25%) for the two largest jobs, which use 130K processes or more. Such cost is particularly negligible considering the significant I/O performance enhancement and run time saving produced by optimizations or problem diagnosis from Beacon-supplied information.

Table 7 lists the CPU and memory usage of Beacon's data collection daemon. In addition, the storage overhead from Beacon's deployment on TaihuLight since April 2017 is around 10 TB. Such low operational overhead and scalable operation attest to Beacon's lightweight design, with background trace-collection and compression generating negligible additional resource consumption. Also, having separate monitoring network and storage avoids potential disturbance to application execution.

Level	CPU usage	Memory usage (MB)
Compute node	0.0%	10
Forwarding node	0.1%	6
Storage node	0.1%	5

Table 7: System overhead of Beacon

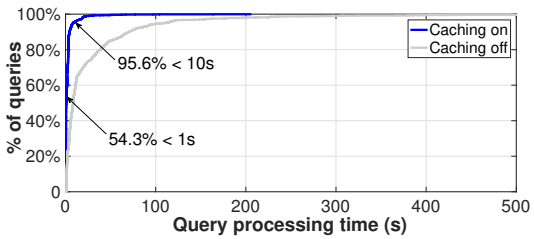


Figure 13: CDF of Beacon query processing time

Finally we assess Beacon's query processing performance. We measured the query processing time of 2000 Beacon queries in September 2018, including both application users accessing job performance analysis and system administrators checking forwarding/storage nodes performance. In particular, we examined the impact of Beacon's in-memory cache system between the web interface and Elasticsearch, as shown in Figure 2. Figure 13 gives the CDF of queries

in processing time and demonstrates that (1) the majority of Beacon user queries can be processed within 1 second and 95.6% of them under 10 seconds (visualization queries take longer), and (2) Beacon's in-memory caching significantly improves user experience. Additional checking reveals that about 95% of these queries can be served from data cached.

6 Related Work

Several I/O tracing and profiling tools have been proposed for HPC systems, which can be divided into two categories: application-oriented tools and backend-oriented tools.

Application-oriented tools can provide detailed information about a particular execution on a function-by-function basis. Work at this front includes Darshan [31], IPM [76], and RIOT [81], all aiming at building an accurate picture of application I/O behavior by capturing key characteristics of the mainstream I/O stack on compute nodes. Carns et al. evaluated performance and runtime overheads of Darshan [30]. Wu et al. proposed a scalable methodology for MPI and I/O event tracing [58, 82, 83]. Recorder [56] focused on collecting additional HDF5 trace data.

Tools like Darshan provide user-transparent monitoring via automatic environment configuration. Still, instrumentation based tools have restrictions on programming languages or libraries/linkers. In contrast, Beacon is designed to be a non-stop, full-system I/O monitoring system capturing I/O activities at the system level.

Backend-oriented tools collect system-level I/O performance data across applications and provide summary statistics (e.g. LIOProf [85], LustreDU [29,48,62] and LMT [38]). However, identifying application performance issues and finding the cause of application performance degradation are difficult with these tools. While backend analytical methods [50, 52] made progress in identifying high-throughput applications using backend logs only, they lack application-side information. Beacon, on the other hand, holds complete cross-layer monitoring data to afford such tasks.

Along this line, there exist tools collecting multi-layer data. Static instrumentation was used to trace parallel I/O calls from MPI to PVFS servers [46]. SIOX [80] and IOPin [45] characterize HPC I/O workloads across the I/O stack. These projects extended the application-level I/O instrumentation approach that Darshan [31] used, to other system layers. However, their overhead hinders its deployment on large-scale production environments [70].

Regarding end-to-end frameworks, the TOKIO [24] architecture combined frontend tools (Darshan, Recorder) and backend ones (LMT). E.g., the UMAMI monitoring inferface [53] provided cross-layer I/O performance analysis and visualization. In addition, OVIS [27] used the Cray specific tool LDMS [22] to provide scalable failure and anormaly detection. GUIDE [91] performed center-wide and multi-source log collection and motivated further analysis and optimizations. Beacon differs by its aggressive real-time performance and utilization monitoring, automatic anomaly detection, and continuous per-application I/O pattern profiling.

I/O interference is identified as an important cause for performance variability in HPC systems [52,63]. Kuo et al. [47] focused on interference from different file access patterns with synchronized time-slice profiles. Yildiz et al. [88] studied root causes of cross-application I/O interference across software and hardware configurations. To our knowledge, Beacon is the first monitoring framework with built-in features for inter-application interference analysis. Our study confirms findings on large-scale HPC applications' adoption of poor I/O design choices [57]. It further suggests that aside from workload-dependent, I/O-aware scheduling [33,52], interference should be countered with application I/O mode optimization and adaptive I/O resource allocation.

Finally, on network monitoring, there are dedicated tools [51, 59, 68] for monitoring switch performance, anomaly detection, and resource utilization optimization. There are also tools specializing in network monitoring/debugging for data centers [14, 73, 89]. However, these tools/systems typically do not target InfiniBand interconnections commonly used on supercomputers. To this end, Beacon adopts the open-source OFED stack [13, 32] to retrieve relevant information from IB network. More importantly, it leverages its scalable and efficient monitoring infrastructure, originally designed for I/O, for network problems.

7 Conclusion

We present Beacon, an end-to-end I/O resource monitoring and diagnosis system for the leading supercomputer TaihuLight. It facilitates comprehensive I/O behavior analysis along the long I/O path and has identified hidden performance and user I/O behavior issues, as well as system anomalies. Enhancement enabled by Beacon in the past 18 months has significantly improved ultra large-scale applications' I/O performance and the overall TaihuLight I/O resource utilization. More generally, our results and experience indicate that this type of detailed multi-layer I/O monitoring/profiling is affordable at state-of-the-art supercomputers, offering valuable insights while incurring very low cost.

Acknowledgement

We appreciate the thorough and constructive comments from all reviewers. Particularly, we thank our shepherd, Haryadi Gunawi, for his responsiveness and detailed guidance. We also thank Prof. Zheng Weimin for his valuable guidance and advice, and colleagues Xiongchao Tang and Haojie Wang for their input. We thank NSCC-Wuxi for supporting our development, data collection, and deployment. This work is partially supported by the National Key R&D Program of China (Grant No. 2017YFA0604500 and 2016YFA0602100) and the National Natural Science Foundation of China (Grant No. 61722208, 41776010, and U1806205).

References

[1] Cori supercomputer. http://www.nersc.gov/users/computational-systems/cori/.

[2] Cray burst buffer in Cori. http://www.nersc.gov/users/computational-systems/cori/burst-buffer/burst-buffer/.

[3] DBSCAN. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.

[4] Edison supercomputer. http://www.nersc.gov/users/computational-systems/edison/.

[5] Elasticsearch. https://www.elastic.co/products/elasticsearch.

[6] GlusterFS. https://www.gluster.org/.

[7] Intrepid. https://www.alcf.anl.gov/intrepid.

[8] K supercomputer. http://www.aics.riken.jp/en/.

[9] Logstash. https://www.elastic.co/products/logstash.

[10] Mira supercomputer. https://www.alcf.anl.gov/mira.

[11] MySQL database. https://www.mysql.com/.

[12] Oakforest-PACS supercomputer. http://jcahpc.jp/eng/ofp_intro.html.

[13] Open Fabrics Alliance. http://www.openfabrics.org/.

[14] PathDump. https://github.com/PathDump.

[15] Piz Daint supercomputer. https://www.cscs.ch/computers/dismissed/piz-daint-piz-dora/.

[16] Redis. http://redis.io/.

[17] Sequoia supercomputer. https://computation.llnl.gov/computers/sequoia.

[18] Sierra supercomputer. https://hpc.llnl.gov/hardware/platforms/sierra.

[19] Sunway TaihuLight supercomputer. https://http://www.nsccwx.cn/.

[20] Titan supercomputer. https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/.

[21] Trinity supercomputer. http://www.lanl.gov/projects/trinity/.

[22] AGELASTOS, A., ALLAN, B., BRANDT, J., CASSELLA, P., ENOS, J., FULLOP, J., GENTILE, A., MONK, S., NAKSINEHABOON, N., OGDEN, J., RAJAN, M., SHOWERMAN, M., STEVENSON, J., TAERAT, N., AND TUCKER, T. The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2014).

[23] BENT, J., GIBSON, G., GRIDER, G., MCCLELLAND, B., NOWOCZYNSKI, P., NUNEZ, J., POLTE, M., AND WINGATE, M. PLFS: A Checkpoint filesystem for parallel applications. In *Proceedings of Supercomputing* (2009).

[24] BERKELEY, L., AND ANL. TOKIO:Total knowledge of I/O, 2017. http://www.nersc.gov/research-and-development/tokio.

[25] BOWEN YU, YOUWEI ZHOU, H. L. X. T. W. C. J. Z. W. Y., AND ZHENG, W. Scalable graph traversal on Sunway TaihuLight with ten million cores. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2017).

[26] BRAAM, P. J., AND ZAHIR, R. Lustre: A scalable, high performance file system. *Cluster File Systems, Inc* (2002).

[27] BRANDT, J., GENTILE, A., MAYO, J., PEBAY, P., ROE, D., THOMPSON, D., AND WONG, M. Resource monitoring and management with OVIS to enable HPC in cloud computing environments. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2009).

[28] BUDNIK, T., KNUDSON, B., MEGERIAN, M., MILLER, S., MUNDY, M., AND STOCKDELL, W. Blue Gene/Q resource management architecture. In *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)* (2010).

[29] CARLYLE, A. G., MILLER, R. G., LEVERMAN, D. B., RENAUD, W. A., AND MAXWELL, D. E. Practical support solutions for a workflow-oriented Cray environment. In *Proceedings of Cray User Group Conference (CUG)* (2012).

[30] CARNS, P., HARMS, K., LATHAM, R., AND ROSS, R. Performance analysis of Darshan 2.2.3 on the Cray XE6 platform. *Argonne National Laboratory (ANL)* (2012).

[31] CARNS, P. H., LATHAM, R., ROSS, R. B., ISKRA, K., LANG, S., AND RILEY, K. 24/7 characterization of petascale I/O workloads. In *Proceedings of the First Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS)* (2009).

[32] DANDAPANTHULA, N., SUBRAMONI, H., VIENNE, J., KANDALLA, K., SUR, S., PANDA, D. K., AND BRIGHTWELL, R. INAM-a scalable infiniband network analysis and monitoring tool. In *European Conference on Parallel Processing (Euro-Par)* (2011).

[33] DORIER, M., ANTONIU, G., ROSS, R., KIMPE, D., AND IBRAHIM, S. CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2014).

[34] DUAN, X., CHEN, D., MENG, X., YANG, G., GAO, P., ZHANG, T., ZHANG, M., LIU, W., ZHANG, W., AND XUE, W. Redesigning LAMMPS for petascale and hundred-billion-atom simulation on Sunway TaihuLight. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2018).

[35] FU, H., HE, C., CHEN, B., YIN, Z., ZHANG, Z., ZHANG, W., ZHANG, T., XUE, W., LIU, W., YIN, W., YANG, G., AND CHEN, X. 18.9-Pflops nonlinear earthquake simulation on Sunway TaihuLight: Enabling depiction of 18-Hz and 8-meter scenarios. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2017).

[36] FU, H., LIAO, J., YANG, J., WANG, L., SONG, Z., HUANG, X., YANG, C., XUE, W., LIU, F., QIAO, F., ZHAO, W., YIN, X., HOU, C., ZHANG, C., GE, W., ZHANG, J., WANG, Y., ZHOU, C., AND YANG, G. The Sunway TaihuLight supercomputer: System and applications. *Science China Information Sciences* (2016).

[37] GAINARU, A., AUPY, G., BENOIT, A., CAPPELLO, F., ROBERT, Y., AND SNIR, M. Scheduling the I/O of HPC applications under congestion. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2015).

[38] GARLICK, J. Lustre monitoring tool, 2010. https://github.com/LLNL/lmt.

[39] GILTRAP, D. L., LI, C., AND SAGGAR, S. DNDC: A process-based model of greenhouse gas fluxes from agricultural soils. *Agriculture, Ecosystems & Environment* (2010).

[40] GRIDER, G., NUNEZ, J., AND BENT, J. LANL MPI-IO test, 2008. http://freshmeat.sourceforge.net/projects/mpiiotest.

[41] GUNASEKARAN, R., ORAL, S., HILL, J., MILLER, R., WANG, F., AND LEVERMAN, D. Comparative I/O workload characterization of two leadership class storage clusters. In *Proceedings of the 10th Parallel Data Storage Workshop* (2015).

[42] GUNAWI, H. S., SUMINTO, R. O., SEARS, R., GOLLIHER, C., SUNDARARAMAN, S., LIN, X., EMAMI, T., SHENG, W.,

BIDOKHTI, N., MCCAFFREY, C., SRINIVASAN, D., PANDA, B., BAPTIST, A., GRIDER, G., FIELDS, P. M., HARMS, K., ROSS, R. B., JACOBSON, A., RICCI, R., WEBB, K., ALVARO, P., RUNESHA, H. B., HAO, M., AND LI, H. Fail-slow at scale: evidence of hardware performance faults in large production systems. In *16th USENIX Conference on File and Storage Technologies (FAST)* (2018).

[43] JI, X., YANG, B., ZHANG, T., MA, X., ZHU, X., WANG, X., EI-SAYED, N., ZHAI, J., LIU, W., AND XUE, W. Automatic, Application-Aware I/O Forwarding Resource Allocation for High-end System. In *17th USENIX Conference on File and Storage Technologies (FAST)* (2019).

[44] JOKANOVIC, A., SANCHO, J. C., RODRIGUEZ, G., LUCERO, A., MINKENBERG, C., AND LABARTA, J. Quiet neighborhoods: Key to protect job performance predictability. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2015).

[45] KIM, S. J., SON, S. W., LIAO, W.-K., KANDEMIR, M., THAKUR, R., AND CHOUDHARY, A. IOPin: Runtime profiling of parallel I/O in HPC systems. In *High Performance Computing, Networking, Storage and Analysis (SCC)* (2012).

[46] KIM, S. J., ZHANG, Y., SON, S. W., PRABHAKAR, R., KANDEMIR, M., PATRICK, C., LIAO, W.-K., AND CHOUDHARY, A. Automated tracing of I/O stack. In *European MPI Users' Group Meeting* (2010).

[47] KUO, C.-S., SHAH, A., NOMURA, A., MATSUOKA, S., AND WOLF, F. How file access patterns influence interference among cluster applications. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2014).

[48] LIM, SEUNG-HWAN AND SIM, HYOGI AND GUNASEKARAN, RAGHUL AND VAZHKUDAI, SUDHARSHAN S. Scientific user behavior and data-sharing trends in a petascale file system. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2017).

[49] LIN, H., ZHU, X., YU, B., TANG, X., XUE, W., CHEN, W., ZHANG, L., HOEFLER, T., MA, X., LIU, X., ZHENG, W., AND XU, J. Shentu: Processing multi-trillion edge graphs on millions of cores in seconds. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2018).

[50] LIU, Y., GUNASEKARAN, R., MA, X., AND VAZHKUDAI, S. S. Automatic identification of application I/O signatures from noisy server-side traces. In *12th USENIX Conference on File and Storage Technologies (FAST)* (2014).

[51] LIU, Z., MANOUSIS, A., VORSANGER, G., SEKAR, V., AND BRAVERMAN, V. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016).

[52] LIU, YANG AND GUNASEKARAN, RAGHUL AND MA, XIAOSONG AND VAZHKUDAI, SUDHARSHAN S. Server-side log data analytics for I/O workload characterization and coordination on large shared storage systems. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2016).

[53] LOCKWOOD, G. K., YOO, W., BYNA, S., WRIGHT, N. J., SNYDER, S., HARMS, K., NAULT, Z., AND CARNS, P. UMAMI: A recipe for generating meaningful metrics through holistic I/O performance analysis. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems* (2017).

[54] LOFSTEAD, J., JIMENEZ, I., MALTZAHN, C., KOZIOL, Q., BENT, J., AND BARTON, E. Daos and friends: A proposal for an exascale storage system. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2016).

[55] LOFSTEAD, J., ZHENG, F., LIU, Q., KLASKY, S., OLDFIELD, R., KORDENBROCK, T., SCHWAN, K., AND WOLF, M. Managing variability in the I/O performance of petascale storage systems. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2010).

[56] LUU, H., BEHZAD, B., AYDT, R., AND WINSLETT, M. A multi-level approach for understanding I/O activity in HPC applications. In *IEEE International Conference on Cluster Computing (CLUSTER)* (2013).

[57] LUU, H., WINSLETT, M., GROPP, W., ROSS, R., CARNS, P., HARMS, K., PRABHAT, M., BYNA, S., AND YAO, Y. A multi-platform study of I/O behavior on petascale supercomputers. In *International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)* (2015).

[58] MUELLER, F., WU, X., SCHULZ, M., DE SUPINSKI, B. R., AND GAMBLIN, T. ScalaTrace: tracing, analysis and modeling of HPC codes at scale. In *International Workshop on Applied Parallel Computing* (2010).

[59] NATHAN, V., NARAYANA, S., SIVARAMAN, A., GOYAL, P., ARUN, V., ALIZADEH, M., JEYAKUMAR, V., AND KIM, C. Demonstration of the marple system for network performance monitoring. In *Proceedings of the SIGCOMM Posters and Demos* (2017).

[60] NEUWIRTH, S., WANG, F., ORAL, S., VAZHKUDAI, S., ROGERS, J., AND BRUENING, U. Using balanced data placement to address I/O contention in production environments. In *International Symposium on Computer Architecture and High PERFORMANCE Computing (SBAC-PAD)* (2016).

[61] NOETH, M., RATN, P., MUELLER, F., SCHULZ, M., AND DE SUPINSKI, B. R. Scalatrace: Scalable compression and replay of communication traces for high-performance computing. *Journal of Parallel and Distributed Computing* (2009).

[62] ORAL, S., SIMMONS, J., HILL, J., LEVERMAN, D., WANG, F., EZELL, M., MILLER, R., FULLER, D., GUNASEKARAN, R., KIM, Y., GUPTA, S., VAZHKUDAI, D. T. S. S., ROGERS, J. H., DILLOW, D., SHIPMAN, G. M., AND BLAND, A. S. Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2014).

[63] OUYANG, J., KOCOLOSKI, B., LANGE, J. R., AND PEDRETTI, K. Achieving performance isolation with lightweight co-kernels. In *International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)* (2015).

[64] PAUL, A. K., GOYAL, A., WANG, F., ORAL, S., BUTT, A. R., BRIM, M. J., AND SRINIVASA, S. B. I/O load balancing for big data HPC applications. In *IEEE International Conference on Big Data (Big Data)* (2017).

[65] SAKAI, K., SUMIMOTO, S., AND KUROKAWA, M. High-performance and highly reliable file system for the K computer. *Fujitsu Scientific & Technical Journal* (2012).

[66] SCHMUCK, F. B., AND HASKIN, R. L. Gpfs: A shared-disk file system for large computing clusters. In *1st USENIX Conference on File and Storage Technologies (FAST)* (2002).

[67] SERGENT, N., DÉFAGO, X., AND SCHIPER, A. Impact of a failure detection mechanism on the performance of consensus. In *Proceedings 2001 Pacific Rim International Symposium on Dependable Computing* (2001).

[68] SHEN, S.-H., AND AKELLA, A. Decor: A distributed coordinated resource monitoring system. In *IEEE International Workshop on Quality of Service (IWQoS)* (2012).

[69] SKAMAROCK, W. C., KLEMP, J. B., DUDHIA, J., GILL, D. O., BARKER, D. M., WANG, W., AND POWERS, J. G. A description of the advanced research wrf version 2. *National Center For Atmospheric Research Boulder Co Mesoscale and Microscale Meteorology Div* (2005).

[70] SNYDER, S., CARNS, P., HARMS, K., ROSS, R., LOCKWOOD, G. K., AND WRIGHT, N. J. Modular HPC I/O characterization with Darshan. In *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools* (2016).

[71] SONG, H., YIN, Y., SUN, X. H., THAKUR, R., AND LANG, S. Server-side I/O coordination for parallel file systems. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2011).

[72] TAI, A. T., TSO, K. S., AND SANDERS, W. H. Cluster-based failure detection service for large-scale ad hoc wireless network applications. In *International Conference on Dependable Systems and Networks (DSN)* (2004).

[73] TAMMANA, P., AGARWAL, R., AND LEE, M. Distributed network monitoring and debugging with switchpointer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2018).

[74] TARASOV, V., KUMAR, S., MA, J., HILDEBRAND, D., POVZNER, A., KUENNING, G., AND ZADOK, E. Extracting flexible, replayable models from large block traces. In *10th USENIX Conference on File and Storage Technologies (FAST)* (2012).

[75] Top 500 list. `https://www.top500.org/resources/top-systems/`.

[76] USELTON, A., HOWISON, M., WRIGHT, N., SKINNER, D., KEEN, N., SHALF, J., KARAVANIC, K., AND OLIKER, L. Parallel I/O performance: From events to ensembles. In *Proceedings of the International Parallel Distributed Processing Symposium (IPDPS)* (2010).

[77] VIJAYAKUMAR, K., MUELLER, F., MA, X., AND ROTH, P. C. Scalable I/O tracing and analysis. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage (PDSW)* (2009).

[78] VISHWANATH, V., HERELD, M., ISKRA, K., KIMPE, D., MOROZOV, V., PAPKA, M. E., ROSS, R., AND YOSHII, K. Accelerating I/O forwarding in ibm blue gene/p systems. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2010).

[79] WANG, Y., LIU, J., QIN, H., YU, Z., AND YAO, Y. The accurate particle tracer code. *Computer Physics Communications* (2017).

[80] WIEDEMANN, M. C., KUNKEL, J. M., ZIMMER, M., LUDWIG, T., RESCH, M., BÖNISCH, T., WANG, X., CHUT, A., AGUILERA, A., NAGEL, W. E., KLUGE, M., AND MICKLER, H. Towards I/O analysis of HPC systems and a generic architecture to collect access patterns. *Computer Science-Research and Development* (2013).

[81] WRIGHT, S. A., HAMMOND, S. D., PENNYCOOK, S. J., BIRD, R. F., HERDMAN, J., MILLER, I., VADGAMA, A., BHALERAO, A., AND JARVIS, S. A. Parallel file system analysis through application I/O tracing. *The Computer Journal* (2013).

[82] WU, X., AND MUELLER, F. Elastic and scalable tracing and accurate replay of non-deterministic events. In *International Conference on Supercomputing (ICS)* (2013).

[83] WU, X., VIJAYAKUMAR, K., MUELLER, F., MA, X., AND ROTH, P. C. Probabilistic communication and I/O tracing with deterministic replay at scale. In *International Conference on Parallel Processing (ICPP)* (2011).

[84] XIE, B., CHASE, J., DILLOW, D., DROKIN, O., KLASKY, S., ORAL, S., AND PODHORSZKI, N. Characterizing output bottlenecks in a supercomputer. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2012).

[85] XU, C., BYNA, S., VENKATESAN, V., SISNEROS, R., KULKARNI, O., CHAARAWI, M., AND CHADALAVADA, K. LIOProf: Exposing Lustre file system behavior for I/O middleware. In *Proceedings of Cray User Group Conference (CUG)* (2016).

[86] XU, T., JIN, X., HUANG, P., ZHOU, Y., LU, S., JIN, L., AND PASUPATHY, S. Early Detection of Configuration Errors to Reduce Failure Damage. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2016).

[87] XU, W., LU, Y., LI, Q., ZHOU, E., SONG, Z., DONG, Y., ZHANG, W., WEI, D., ZHANG, X., CHEN, H., XING, J., AND YUAN, Y. Hybrid hierarchy storage system in MilkyWay-2 supercomputer. *Frontiers of Computer Science* (2014).

[88] YILDIZ, O., DORIER, M., IBRAHIM, S., ROSS, R., AND ANTONIU, G. On the root causes of cross-application I/O interference in HPC storage systems. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2016).

[89] YU, M., GREENBERG, A. G., MALTZ, D. A., REXFORD, J., YUAN, L., KANDULA, S., AND KIM, C. Profiling network performance for multi-tier data center applications. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2011).

[90] YU, W., VETTER, J. S., AND ORAL, H. S. Performance characterization and optimization of parallel I/O on the Cray XT. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2008).

[91] ZIMMER, CHRISTOPHER J. GUIDE: A scalable information directory service to collect, federate, and analyze logs for operational insights into a leadership HPC facility. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (2017).

Appendix A Evaluation of Beacon Data Compression

App.	1st-pass	2nd-pass (lossless)	2nd-pass (lossy)
APT	5.4	2.1	2.3
WRF	14.2	3.8	5.5
DNDC	10.1	3.4	5.3
XCFD	12.2	3.8	6.2
GKUA	34.6	3.6	5.1
CAM	9.2	4.4	5.4
AWP	15.1	3.2	11.3
Shentu	22.2	2.6	5.7

Table 8: Compression ratio of sample applications

Table 8 summarizes the effectiveness of Beacon's monitoring data compression. It gives the compression ratio under three kinds of methods of 8 applications, 5 of which are `Shentu`, `LAMMPS`, `DNDC`, `WRF` and `AWP`, discussed in more details previously. The other three are `APT` [79] (particle dynamics simulation), plus `GKUA` and `XCFD` (both closed-source computational fluid dynamics simulators).

We report the compression ratio of the 1st-pass compression (intra-node compression during monitoring data collection) and 2nd-pass compression (inter-node compression during offline log processing on dedicated Beacon server). We experimented with two compression techniques for the latter, one lossless and one lossy (with reduced data precision in file descriptor and offset).

Results in Table 8 indicate significant data size reduction by the 1st-pass compression, with a factor of 5.4 to 34.6 right at the source of monitoring. The second pass, on the other hand, achieves less impressive reduction, partly due to that data have already undergone one pass of compression. Here, though the compute nodes are performing similar I/O operations, different values in parameters such as file offset make it harder to combine data entries. In particular, lossy compression may bring an additional 2.3×-11.3× after 1st-pass compression improvement in compression ratio, however trading the capability of performing certain analysis tasks. Considering our dedicated Beacon server's storage capacity (120 TB) and Beacon's data collection rate (10 TB in 18 months), we elect to use a lossless algorithm for our 2nd-pass compression.

Appendix B Anomaly Detection

Beacon performs two types of automatic anomaly detection, to identify *job I/O performance anomaly* and *node anomaly*, respectively.

Beacon detects job I/O performance anomaly by checking newly measured I/O performance results against historical records, based on the assumption that most data-intensive applications have rather consistent I/O behavior. First, it adopts the automatic I/O phase identification technique as in the IOSI system [50] developed on the Oak Ridge National Laboratory Titan supercomputer, which uses Discrete Wavelet Transform (DWT) to find distinct "I/O bursts" from continuous I/O bandwidth time-series data. It then deploys DBSCAN algorithm [3], also used in IOSI, to check whether I/O phases from the new job execution conform to known clusters of the same application's past executions at the same scale. More specifically, it performs 2-D clustering in terms of the I/O phases' time duration and total I/O volume. When outliers are found, Beacon further utilizes its rich monitoring data to examine neighbor jobs that share forwarding node(s) with the job in question. In particular, it determines whether such neighbors have interference-prone features, such as high MDOPS, high I/O bandwidth, high IOPS, or N-1 I/O mode. Such findings are saved in the Beacon database and provided to users via the Beacon web-based application I/O query tool. Applications of course will need to accumulate at least several executions for such detection to take effect.

Beacon's node anomaly detection relies on the execution of large-scale jobs (those using 1024 or more compute nodes in our current implementation), where it leverages the common homogeneity in I/O behavior across compute and server nodes to spot outliers. Its multi-level monitoring allows the correlation of I/O activities or loads back to actual client side issued requests. Again by using clustering algorithms like DBSCAN and configurable thresholds, Beacon performs outlier detection across forwarding nodes and OSTs involved in a single job, where the vast majority of entities report highly similar performance while a few members produce contrasting readings. Figure 8 in Section 4.2 gives an example of per-OST bandwidth data within the same execution.