Accelerated Real-time Network Monitoring and Profiling at Scale using OSU INAM

P. Kousha
The Ohio State University
Columbus, Ohio
kousha.2@osu.edu

Kamal Raj S.D.
The Ohio State University
Columbus, Ohio
sankarapandiandayalaganeshr.1@osu.edu

H. Subramoni
The Ohio State University
Columbus, Ohio
subramoni.1@osu.edu

D. K. Panda
The Ohio State University
Columbus, Ohio
panda.2@osu.edu

H. Na
The Ohio Supercomputer Center
Columbus, Ohio
hna@osc.edu

T. Dockendorf
The Ohio Supercomputer Center
Columbus, Ohio
tdockendorf@osc.edu

K. Tomko
The Ohio Supercomputer Center
Columbus, Ohio
ktomko@osc.edu

ABSTRACT

Designing a scalable real-time monitoring and profiling tool with low overhead for network analysis and introspection capable of capturing all relevant network events is a challenging task. Newer set of challenges come out as HPC systems are becoming larger and users are expecting to have better capabilities like real-time profiling at fine granularity. We take up this challenge by redesigning OSU INAM and making it capable to gather, store, retrieve, visualize, and analyze network metrics for large and complex HPC clusters. The enhanced OSU INAM tool provides scalability, low overhead and fined-granularity InfiniBand port counter inquiry and fabric discovery for HPC users, system administrators, and HPC developers. Our experiments show that, for a cluster of 1,428 nodes and 114 switches, the proposed design can gather fabric metrics at very fine (sub-second) granularity and discovers the complete network topology in approximately 5 minutes. The proposed design has been released publicly as a part of OSU INAM Tool and is available for free download and use from the project website.

CCS CONCEPTS

• **Networks → Network measurement**.

KEYWORDS

InfiniBand, Network Monitoring, Profiling, Fabric, Interconnect

ACM Reference Format:
P. Kousha, Kamal Raj S.D., H. Subramoni, D. K. Panda, H. Na, T. Dockendorf, and K. Tomko. 2020. Accelerated Real-time Network Monitoring and Profiling at Scale using OSU INAM. In *Practice and Experience in Advanced Research Computing (PEARC '20), July 26–30, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3311790.3396672

1 INTRODUCTION

Recent advances in High Performance Computing (HPC) have provided the fast processing and data movement needs for HPC applications. HPC systems must be highly optimized to keep up to the new needs of modern HPC applications. As a solution to deliver the required computing power for the applications, large-scale HPC cluster with multiple high-speed interconnects technology such as Peripheral Component Interconnect express (PCIe), NVIDIA NVLink, AMD Infinity Fabric, Mellanox InfiniBand, High Speed Ethernet and so on are being deployed and used by various HPC users like domain scientists, HPC software developers, and HPC administrators.

Considering these advances in interconnect technology, providing efficient data movement between compute nodes over such varied communication fabric is essential to have end-to-end high-performance solutions. In such an ecosystem, understanding the interaction between applications, MPI libraries, and the communication fabric has become even more challenging. Therefore, detailed and real-time insight of network level and the communication is needed for all types of HPC users to identify and alleviate performance bottlenecks.

Most HPC users are scientists and domain specialists who may not be experts on the lower level details of the runtime system or on how to use or gather such profiling information. Providing a tool for them to easily understand the communication patterns on the network and visualize the live and historical communication for their jobs in a scalable and real-time fashion will be of great use. However, this is a challenging task which consists multiple phases listed here — 1) gathering the fabric information, 2) storing the information, and 3) visualizing various metrics in real time. Moreover, such tool benefits various HPC users from HPC administrators to domain scientists with new user-level novelty that was not easily accessible before or required knowledge about the profiling tools.

Currently, administrators of HPC systems and developers of HPC applications/middleware rely on a plethora of tools to aid them in this interplay. There are various profiling and tracing tools for HPC systems available at system level. There exists a variety of MPI level profiling tools (TAU [15], HPCToolkit [11], Intel VTune [12], IPM [2], mpiP [4]) that give insights into the MPI communication behavior of applications. However, they are unable to profile what happens in the communication fabric. On the other hand, several network level profiling and analysis tools exist that allow system administrators to analyze and inspect the network fabric Ganglia [1], Mellanox Fabric IT [16], BoxFish [14], Lightweight Distributed Metric Service (LDMS) [7]). However, they are unable to relate network activity to their triggering events in the MPI library.

Recently, by designing the OSU InfiniBand Network Analysis and Monitoring with MPI (INAM) [17, 18, 22], we have made attempts to bridge this gap by developing a tool that is capable of allowing users to correlate MPI events and network activity. Figure 1 presents an overview of OSU INAM. OSU INAM gives holistic and unified view of various HPC layers like job scheduler, MPI library, and communication fabric. OSU INAM users have the capabilities to analyze and profile node-level, job-level and process-level activities for MPI communication. OSU INAM provides new visualizations and in-depth insights from the extent of tracking active MPI ranks communication on a given InfiniBand link to the job topology visualization across the cluster and reporting the link bandwidth utilization for modern HPC clusters.

While OSU INAM takes up this challenge, a newer set of challenges are coming out as systems are becoming larger and HPC users are expecting to have better capabilities while maintaining real-time and low overhead profiling and monitoring. Therefore, while OSU INAM visualizes and profiles clusters of smaller sizes, several challenges must be addressed to enable scaling it to larger supercomputing systems to provide real-time profiling.

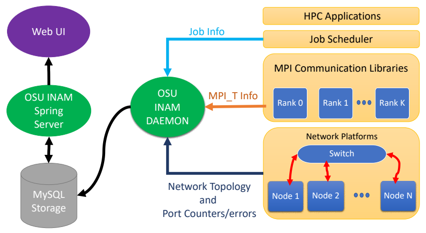


Figure 1: High-level overview of OSU INAM

1.1 Motivation and Analysis

Based on the experience gained from designing, developing, and deploying the previous version of INAM on several HPC clusters and feedback from the community, we identified the bottlenecks for scaling and maintaining real-time network monitoring of OSU INAM for large-scale (>1,000 nodes) clusters. As new HPC systems are becoming larger, like the Frontera cluster at the Texas Advanced Computing Center that contains 8008 compute nodes, a newer set of challenges are raised to maintain real-time and scalability while providing better capabilities.

Due to increase in number of nodes in a cluster and due to High bandwidth interconnects, we noticed through OSU INAM that, it is difficult to capture communication patterns by reading port counters at larger timer intervals (in seconds). Port counters have to be recorded at finer intervals (in sub-seconds) to capture communication patters in detail. Fabric discovery which takes more than 1 hour has to be reduced. Finally, purging the old data has considerable impact on the performance of database for production deployment. Further performance assessment of OSU INAM showed that the workload between modules of OSU INAM Daemon is not well-balanced. Therefore, we asked ourselves how can we further improve OSU INAM Daemon modules to boost performance of the tool compared to the simulations from previous versions and improve scalability to address the needs for large scale HPC cluster.

Section 5 mentions related work and compares tools to OSU INAM. OSU INAM provides end-to-end and holistic view of HPC system. To address scalability challenges and supporting real-time features of OSU INAM, we conducted detailed assessments of the behaviour and interplay of the components of the tool. Our studies showed that some components inside the daemon are sequentially dependent on each other while logically they can be independent (e.g. the fabric discovery and InfiniBand port inquiry can be done independently). Connections to gather and read performance counters can be reused thereby saving on connection establishment time. Further, the granularity of profiling can be improved by taking advantage of all available resources on modern multi-/many-core systems through multi-threading and improving the load balancing between OSU INAM modules and preserve load balancing between the new multi-thread design of fabric discovery and port counter inquiry.

1.2 Contributions

In the paper, we take up this newer set of challenges and come up with a new set of designs to handle scalability of InfiniBand Network Monitoring and improve OSU INAM performance to profile and visualize large InfiniBand networks in real time and fine granularity. To summarize, this paper makes the following contributions:

- Scaling and redesigning a fine-grained low-overhead profiling tool for InfiniBand fabric discovery for large HPC clusters
- Improve granularity of reading port counters and errors to sub-second for large scale HPC clusters
- Enhance gathering, storing, retrieving, and visualization of the metrics for large and complex HPC networks with low latency

Table 1 outlines the enhancements made through this paper to OSU INAM.

2 DESIGN

Figure 2 depicts the high level design and components of our proposed design for the INAM daemon. Please note that the methodology used in this paper is applicable to any tool used for profiling InfiniBand networks. We chose OSU INAM since the framework

Table 1: Comparison of features between proposed and older versions of OSU INAM

Features	V0.9.3	Proposed
Multi-threaded POSIX Modules for OSU INAM Daemon	✓	✓
Multi-threaded Fabric Discovery	✗	✓
Multi-threaded Port Inquiry	✗	✓
Load Balancing between OSU INAM Threads	✗	✓
Storage Management Module	✗	✓
Parallel Insertions into Storage	✗	✓
Bulk Storage Insertions	✓	✓
Bulk and Chunked Storage Deletion	✗	✓
Caching & Reusing MAD Connections	✗	✓
Caching & Reusing Storage Connections	✗	✓

provides an environment to gather, correlate and visualize job level, MPI level, and Fabric/Network level metrics.

Our evaluations showed that some components inside the daemon are sequentially dependent on each other while logically they can be independent (e.g. the fabric discovery and InfiniBand port inquiry can happen in parallel). Further, connections to gather and read performance counters can be reused thereby saving on connection establishment time. Additionally, the granularity of profiling can be improved by taking advantage of all available resources on modern multi-/many-core systems through multi-threading. Previous versions of OSU INAM used a simulator for discovering fabric and reading port counters. On the production, we noticed that its low level APIs will cause additional delay.

In this section, first we discuss the storage management section, then the design of fabric discovery and port inquiry modules to address the performance issues in Section 1.1. All the modules have an exit phase that only happens if a failure occurs in other components and safely cause termination of the module.

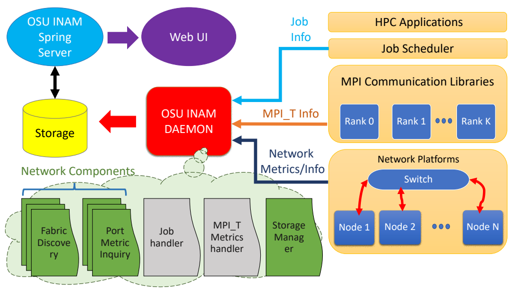


Figure 2: Detailed overview of new OSU INAM with the new multi-threading modules. The green components inside the OSU INAM daemon are new or enhanced.

We launch #n POSIX threads for each module represented by the boxes inside the green cloud in Figure 2. This is an user configurable parameter and can be tuned based on the size of the cluster and the storage needs of the users. The threads get bound to different compute cores to prevent over subscription and thus can run simultaneously. OSU INAM is designed in such a way as to make the operation of each thread independent from the operations of the others. This modular design provides improved lock-free progress and better error handling. A state machine handles creation, execution, and destruction of each thread.

2.1 Storage Management Module

OSU INAM uses MySQL, due to its ubiquitous nature, to store data. The collected data size is proportional to the granularity of data collection and the time since the last database purge. Therefore, it is important to discuss how we purge the old data since it has an impact on performance of the database. OSU INAM will keep the history for several days for InfiniBand port counters and errors. The routes, nodes and links are updated each time fabric discovery module (described in Section 2.2) executes. A separate thread (called the purger thread) is used to purge the old port data in order to maintain the minimum overhead for the other components. The interval of checking for old data and the history of keeping data are user-defined variables. Section 4.3 describes the factors impacting the size of the database and gives guidance on how to set these variables properly depending on the size of the cluster. Our proposed storage handler has three phases: startup, purge, and exit.

- **Startup Phase**: In this phase, the INAM daemon will setup the database connection and then the create the tables. All other threads wait for creation of the tables. After completion of table creation, the main thread is signalled to indicate the database setup is complete so that other components can proceed.
- **Purge Phase**: After the startup phase, the purger thread will periodically purge the port counters, port errors, process information, and process communication tables. Figure 3 shows that deletion time increases with increase in the number of rows deleted. To avoid the high cost for deleting, we chunk the deletes. Performing deletion in chunks allowed faster insertion of new fetched data by port inquiry module (described in Section 2.3). Therefore, the purge happens as a MySQL procedure to delete "X" number of rows. "X" is an user configurable parameter and guidance is provided in the OSU INAM userguide on how to select this parameter for different configurations and cluster sizes. In between each call to the delete procedure, there is an interval to wait which enables the port inquiry module to insert new data. By having the described procedure, we significantly reduce the total purging latency and time fluctuation of data insertion for the database.

2.2 FB_thread: Fabric Discovery Module

The fabric discovery module in OSU INAM is built on the top of OpenSM [5]. The module uses the "ibnd_discover_fabric" API to discover the fabric object connected to the node on which INAM is running. It then uses "ibnd_iter_nodes_type" to classify the discovered objects as Host Channel Adapter (HCA), switches, and routers. During this step, we construct a data structure of routers, switches and HCAs and populate it with the detailed fabric information such as Local Identifier (LID), Global Unique Identifier(GUID), node

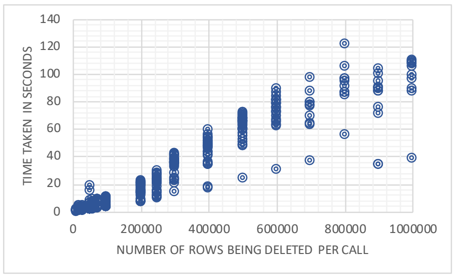


Figure 3: Effect of number of rows being deleted on the latency of MySQL database operations. During the delete, the table is locked for other updates.

name, number of ports, etc. Once this operation is complete, we begin the route discovery to identify the routes between all the nodes in the fabric as a pair of GUID and port number.

The "fabric object" is the only sequential dependency that fabric discovery and port inquiry module have at the initialization of OSU INAM. The port inquiry module will get the fabric object from the fabric module. The rationale behind this modular designs is as follows — the API for fabric discovery is costly, therefore by reusing the fabric object in port inquiry and decoupling fabric discovery and reading port counters, we can read the port counters faster. The successive port inquiry calls will use the fabric object obtained from fabric discovery module. Once fabric inquiry updates the fabric object, it will reflect to port inquiry as well. It is expected that HPC networks are stable, but in the worst case of failures in the fabric, we capture port errors before detecting a change in the fabric object.

The fabric module uses multiple OpenMP threads to exploit parallelism to discover routes. The number of threads assigned to the fabric discovery module is a user-defined variable. A detailed discussion on how to choose the best number of threads is presented in Section 3.

Our proposed fabric discovery component has three phases: startup, query and write, and exit.

- **Startup Phase**: FB_thread will wait for the signal of storage thread in section 2.1. Then, a database connection is established for each thread to store its data after discovering the routes. The database connections will be reused. Then, it signals port inquiry module to proceed after passing the fabric object to it.
- **Query and Write Phase**: For each thread, we use Management Datagram (MAD) Remote Procedure Call (RPC) port to perform the route discovery for the fabric component to discover the routes among assigned nodes. The MAD ports will be reused thereby avoiding costly new MAD connection creation. After each thread discovers its portion of the routing in the fabric, it will write the gathered metrics into the database independently using bulk insertions. Since, for n nodes, the total number of routes are O(n^2), parallel insertion of routes provides significant benefits. The first set of threads that finish the route insertion will proceed to the next phase of updating the "nodes" and "links" tables. Then the threads will go to sleep until the next periodic invocation.

The time between invocation of the fabric discovery threads is an user configurable value and guidance is provided in the OSU INAM userguide on how to select this parameter for different configurations and cluster sizes.

2.3 Port Metrics Inquiry Module

The port metrics inquiry module is responsible for reading the InfiniBand port counters and port errors and writing them into the database. Since querying port counters depends on the size of the cluster and the architecture of the cluster like number of switches and the number of ports per switch, there are three modes that user can activate: serial, switch-level multi-threading, port-level multi-threading.

The serial read has one thread reading all the port counters and errors for the underling network. The switch-level threading uses OpenMP threads to read the switches port counters in parallel by getting the number of threads as a user-defined variable. The port-level threading uses OpenMP threads to read the metrics of a switch in parallel and is useful when the switch has dense number of port like [3] that has 648 ports.

Our proposed port metric inquiry component has four phases: startup, query, write and exit.

- **Startup Phase**: Based on the number of threads that user assigned to PC component, the PC_thread establishes the database connections. Then, the PC_thread will wait for the signal from the purge thread in section 2.1. Next, PC_thread will open the MAD port connections based on the number of threads and sets up the shared data structure among threads. Then, it will wait for the Fabric to load the fabric object. Please note that we only wait for the fabric object at this phase.
- **Query Phase**: The user selects one of the design based on their cluster specifications. PC_thread will perform an iteration over the network to get the nodes types and information. Then, it will use OpenMP threading with dynamic scheduling to read port data counters and errors by using "pma_query_via" API to get the port metrics. The metrics are stored as a data structure for each thread to be stored into database.
- **Write Phase**: After each thread will gather the port metrics, each thread will write the data into database without waiting for each other. By using bulk insertions, we reduce the storage time as discussed in [22]. After the write phase the PC_thread will go to sleep. The time between subsequent queries of port metrics is an user configurable value and guidance is provided in the OSU INAM userguide on how to select this parameter for different configurations and cluster sizes.

3 RESULTS AND EVALUATION

In this section, we evaluate our extended design for two clusters. A large-scale cluster with more than 1400 nodes and small-scale HPC cluster. All measure timings are collected using monotonic clock. Port inquiry sweep means that the total time taken to remotely gather all InfiniBand port counters and errors from all the nodes in the cluster.

3.1 Experimental Setup

We conducted our experiments on two clusters, RI, a medium scale cluster at The Ohio State University (OSU) and Ohio Supercomputer Center(OSC)[10], 3 clusters connected with a single fabric. Table 2 summarizes the specification of each the clusters.

Table 2: The hardware specification of clusters used for experiments

	RI	**OSC**
Overview	Equipped with MT26428 QDR ConnectX-2 HCAs with PCI-Express interfaces	3 heterogeneous clusters connected to the same InfiniBand fabric
#Nodes	146	1,428
#Links	542	3,402
#Switches	20 (36 ports/switch)	114 (36 ports/ switch)
CPU	Intel Xeon 2.53Ghz	Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz
Cores/node for INAM	8	24
L3 cache	12MB	30MB
Memory/node	12GB	128GB
Switch Technology	Mellanox MTS3610QDR	Two clusters have Mellanox EDR Infiniband (100Gbps) and one with FDR/EN (56Gbps)
Job Scheduler	SLURM	PBS

3.2 Comparison with Older OSU INAM Version

To clarify and compare the improvements of the new design, OSU INAM v0.9.5, we compared the fabric discovery time and port inquiry sweep for RI cluster. As mentioned in the new design, each thread for port inquiry and fabric discovery writes its portion of data into the database. However, in the previous OSU INAM version, another POSIX thread used to perform database insertions. For quantifying timings for OSU INAM v0.9.3, timings of querying and database storing are measured separately and added in table 3. The experiment used an interval of 30 seconds for fabric discovery and port inquiry modules while using only one thread for port inquiry and one thread for fabric discovery. Although using single thread for both network modules, by using cached MAD connections and overlapping storing and querying data, the new design achieved 29% and 64% improvement for fabric discovery and port inquiry sweep, respectively.

3.3 Fabric Discovery Evaluation

In this section, we measure the performance and scalability of fabric discovery component for small and large scale supercomputers.

Table 3: Comparison of fabric discovery and port inquiry sweep modules for RI cluster between improved version of OSU INAM(v0.9.5) and previous version. Both network modules only use a single thread. Numbers are in seconds.

OSU INAM module	**v0.9.3**	**v0.9.5**	%Improvement
Fabric discovery	85.59s	66.13s	29%
Port Inquiry Sweep	0.23s	0.14s	64%

3.3.1 Impact of Multi-threading on Fabric Discovery. We ran the improved OSU INAM daemon to measure the effect of multi-threading for the fabric discovery module. To be fair and mimic the behavior of real deployment, we set some threads for port inquiry while measuring performance of fabric inquiry. In this experiment we set 20 threads for port inquiry with interval of 5 seconds for OSC cluster and 1 thread with an interval of 1 second for RI cluster. The timings measured include discovery of fabric and insertion of data into the database. Figure 4(b) shows the result for RI cluster and a speed-up of 6x by using 8 threads. Figure 4(a) shows the same experiment with 8, 16, 20, and full subscription mode for OSC clusters.

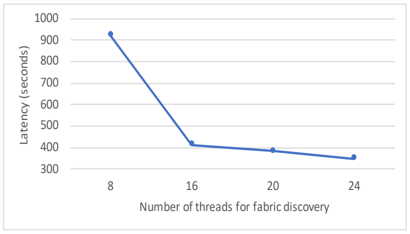
3.3.2 Evaluating Performance Variability in Fabric Discovery. We conducted a test to measure the performance variation of Fabric Discovery module at RI cluster over 200 samples. In this test, 8 threads were used for fabric discovery with 30 seconds querying interval and 1 thread for port inquiry with 1 second querying interval. We found that the median is 14.26 seconds with standard deviation of 1.39 seconds. The anomaly is due to over subscription of threads when all the modules (Port Inquiry, Fabric Discovery, MPI_T Handler, Job Handler, Purge Thread) inside the Daemon were active. We are investigating methods to address this issue.

3.4 Port Inquiry Evaluation

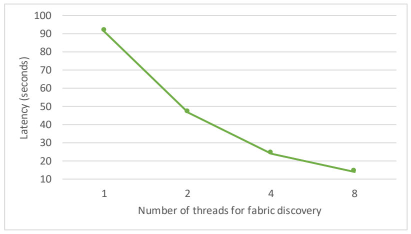
In this section, we performed tests to measure the scalability and performance variation of Port Inquiry module inside OSU INAM Daemon.

3.4.1 Impact of Multi-threading on Port Inquiry. We ran OSU INAM daemon with the improved design to measure the effect multi-threading for the port inquiry module. To be fair and mimic the behavior of real deployment, we set some threads for fabric discovery while measuring performance of port inquiry module. We set 20 threads and 6 threads for OSC and RI, respectively with one-hour querying interval. The timings reported in this section is the sum of gathering port metrics and insertion into the database.

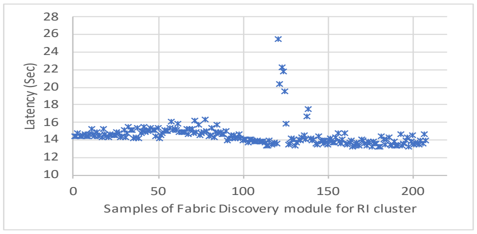
Figure 5(a) and 5(b) depicts the scalability of our design and the effect of the number of threads for port inquiry component. In Figure 5(a) and and 5(b) the interval of querying ports is set to 1 seconds. The test population for RI experiment is 2,000 samples and for OSC it is 1,000 samples. Since there are only 20 switches on RI cluster, the overhead of threading is out weighting multi-threading benefits. Our experiments shows that when we over subscribe to the hardware cores, the variation between samples increases since both fabric threads and port inquiry threads are overlapping for some CPU cores. For example for OSC cluster, the standard deviation doubles when using more than 16 threads since there are 20 threads



(a) Impact of multi-threading on fabric discovery module on OSC cluster with hourly interval

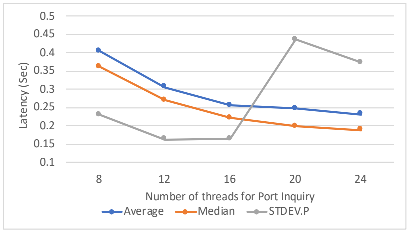


(b) Impact of multi-threading on fabric discovery module on RI cluster with 1 minute interval

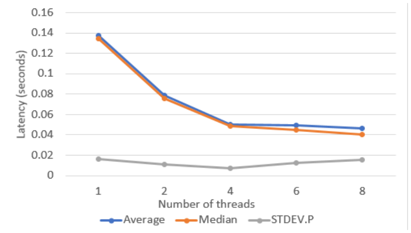


(c) Histogram of the time taken for fabric discovery module using 8 threads on RI cluster. Port inquiry frequency is set to 1 second

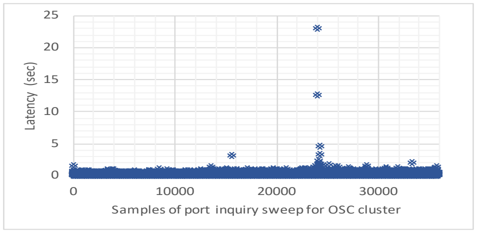
Figure 4: Fabric Discovery latency evaluation for the entire cluster - the fabric is discovered remotely from the node OSU INAM is running on



(a) Impact of multi-threading on the latency of port inquiry sweep with 1 second query interval on the OSC clusters. OSC has 3,402 active ports



(b) Impact of multi-threading on latency of port inquiry sweep with 1 second querying interval for RI cluster. RI cluster has 542 active port



(c) The latency of port inquiry sweep for 35,000 samples for OSC cluster with query interval of 250 ms and using 16 thread

Figure 5: Port inquiry performance evaluation for the entire cluster - the port counters/errors are gathered remotely from the node OSU INAM is running on for the entire cluster

for fabric inquiry and 3 threads for other components of OSU INAM depicted in Figure 2.

3.4.2 Port Inquiry Performance Variability. We conducted experiments on OSC cluster to capture the variability of port inquiry module with the interval of 250ms. The node that OSU INAM was running on has 24 cores and for the test, we allocated 16 threads for port inquiry and 8 threads for fabric discovery. Figure 5(c) shows the variation of port inquiry sweep for more than 35,000 samples. The latency is mostly consistent and has a median of 0.26 second with standard deviation of 0.13 second. The abnormally is due to over subscription of threads as discussed in Section 3.4 when all the OSU INAM modules inside the OSU INAM Daemon were active. Moreover, all of the CPU cores were allocated to OSU INAM, while MySQL needs some hardware resources to handle the concurrency of insertions.

3.5 Overhead of Visualization and Rendering

In this section, we evaluated the time of visualizing the graph in Figure 6(b) and showing the live jobs pages, the two pages that are the most time consuming. By using PhantomJS, rendering the graph in Figure 6(b) takes around 1.4 second on average and it happens once when user reloads the page. Table 4 shows the timing of updating and showing the changes in the network graph in Figure 6(b) and live jobs tab.

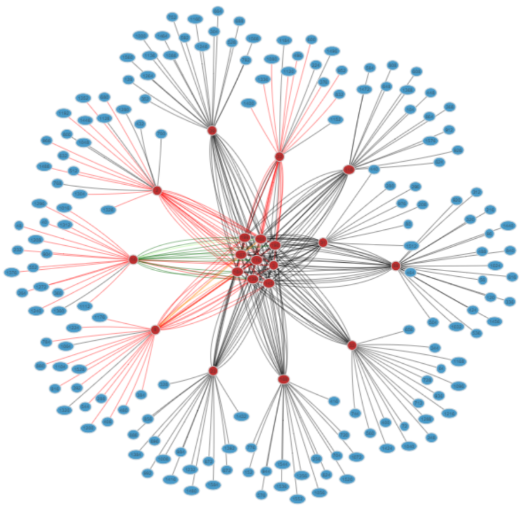
Table 4: Network and live jobs view retrieval timing for OSC for 1K jobs

View	Average	Min	Max	Standard deviation
Network View	196.15 ms	187 ms	206.09 ms	5.75 ms
Live Jobs View	18.17 ms	16 ms	20 ms	1 ms

3.6 Comparison of Simulator with the Deployed Version

In this section, we illustrate how close OSU INAM simulator performance is to real tool deployment. The simulator allows the user to specify attributes of the system to mimic. The simulator helps users to play around with the tool without allocating resources and deploying it. Another benefit is for user to see how different hardware resources allocations for each modules of the tool will impact the module and overall performance.

From our experiments, we noticed that the performance of the simulator is a close approximation of how INAM v0.9.5 would perform in a live environment. We compared most active network module of OSU INAM, port metric inquiry in RI cluster across OSU INAM v0.9.3 (the old design), the improved design (v0.9.5) and the simulator in a single-threaded mode. The old design takes 0.23 seconds while Similator and the improved design takes 0.14 seconds to perform port metric inquiry sweep of the cluster.


(a) RI cluster - 146 nodes, 542 links, and 20 switches

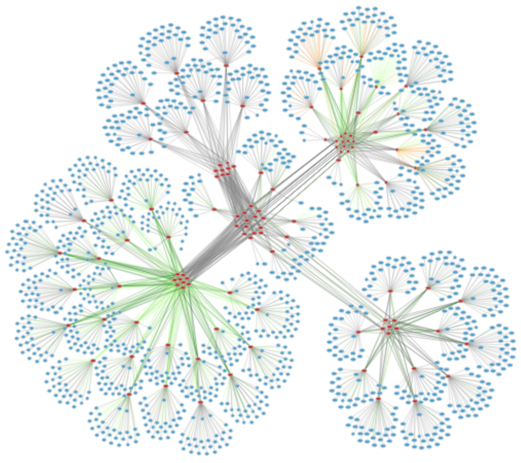

(b) OSC cluster - 1,428 nodes, 3,402 links, and 114 switches

Figure 6: Network topology as rendered by OSU INAM. The links are colored based on the live network traffic

The simulator is very close to how the improved version (v0.9.5) of INAM would perform on the actual cluster. When we performed the same experiment on the OSC cluster, we noticed that the single-threaded simulator was able to perform port inquiry in 0.95 seconds, while the deployed instance with 8-threads completed port inquiry in 0.40 seconds.

4 DISCUSSION

4.1 Best Practices with OSU INAM

In this section, we provide some guidance on the proper resource management for all three components of OSU INAM depicted in Figure 2. OSU INAM consist of Daemon, storage (MySQL), and web-based front end. The node resources like hardware cores should be divided in a fair way to all the components to avoid bottlenecks on one.

There is a trade-off between increasing the availability for the web front and increasing the performance of profiling. The more users use the tool, the more read traffic goes through the storage component and it can impact its availability. Considering that MySQL locks the tables for insertions and concurrent insertions are sequential, having a very low interval of querying by increasing the number of threads can cause starvation for the read queries. We proposed the purge thread design to prevent starvation of inserts and reads in Section 2.1. However, as experiments show in the experiments of Section 3, when all OSU INAM Daemon components are active there is variation in performance of OSU INAM Daemon. The issue can be reduced by choosing a proportional core allocation for the OSU INAM daemon.

A challenging question for the OSU INAM deployment is the proper allocation of number of thread based on number of CPU cores for each module inside OSU INAM Daemon. Port counters and errors are the biggest tables inside storage. Two threads need to go over port counters and errors for purging old data. As a result, it is recommended to have 2 of the CPU cores for the purge thread to avoid performance fluctuation. The two cores can handle job scheduler modules and MPI_T handler module as well. Since HPC fabrics are mostly stable, the interval of fabric discovery can be set to a large value. In that case, the user can depend on finding the errors on the links for fast failure discovery.

Such consideration will allow allocating more resources to the port inquiry module to maintain sub-second granularity. Based on the short interval of port inquiry and the importance of fine-granularity, the CPU cores allocated for the port inquiry should not overlap with the cores used for threads for other components. In other words, the hardware cores allocated for port inquiry should not be oversubscribed. The remaining cores can be divided between fabric discovery and the OSU INAM web front. Table 5 summarizes our suggestion deployment.

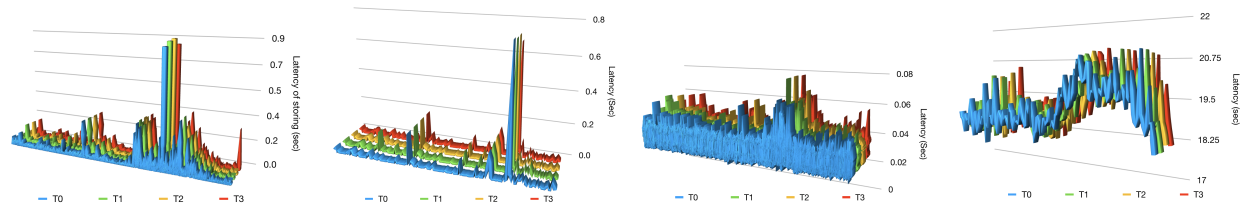
4.2 Load Balancing of Threads in Port Inquiry

Without losing generality, we show the result of the query and write phase for each thread for RI cluster. The experiment used 4 threads for fabric discovery with interval of 30 seconds and 4 thread for port inquiry with interval of 1 second. Figures 7(a) and 7(c) show the latency of write and query phase for each of the thread in port inquiry module. We noted that although there are variations between the performance of each thread, they all follow almost the same pattern. Both figures shows the samples/time for each port inquiry thread.

Figures 7(b) and 7(d) depict the latency of write and query phase for each of the thread in fabric discovery module for the same experiment samples. Please note that capturing the thread activity has increased the overall timing of both components due to expensive API calls and barriers. Therefore, the detailed timing decreases the overlap between fetching the data and storing into the database.

Table 5: Suggested number of cores per functional component when deploying OSU INAM for best performance

Cluster Size	Fabric Discovery	Port Inquiry	MPI_T and Job Handler	Storage Purge
<500	2	2+	1	2
500 <size <1000	4	8+	1	2
>1000	8	16+	2	2



(a) Timing of write phase for each thread for fabric discovery (b) Timing of write phase for each thread for port inquiry (c) Timing of query phase for each thread for port inquiry (d) Timing of query phase for each thread for fabric discovery

Figure 7: Timing of write and query phase for each thread of port inquiry and fabric inquiry for RI cluster - T# represents the time taken by that thread

4.3 Modeling of the Aggregated Data on the Network

In this section, we identify the amount of traffic added to the network from our tool. The port inquiry for each node adds 116 bytes to collect port errors and 84 bytes for port data counters. The traffic goes through virtual lane 15 with the highest priority compared to others and uses Unreliable Datagram (UD) transport service. [21] discuss the overhead of profiling from end to end perspective. Interval of querying impacts the amount of traffic added to network. Therefore, the maximum traffic for an entire sweep is

$$Traffic_{(tool)} = Traffic_{(PC\_total)} + Traffic_{(FB\_total)} \quad (1)$$

$$Traffic_{(PC\_total)} = 200Bytes \times QueryFrequency$$
$$\times NumberofSwitches \times NumberofSwitchPorts \quad (2)$$

For fabric discovery, it depends on the topology of the network and the diameter of the network. For each switch connected to the node that OSU INAM is running on, fabric discovery sends back the GUID and port number of switch. The maximum size of route discovery traffic is

$$Traffic_{(FB\_total)} = NetworkDiameter \times 20Bytes \times$$
$$NumberofSwitchPorts \times QueryFrequency \times NumberofSwitches \quad (3)$$

For port inquiry of OSC cluster, OSU INAM traffic will be maximum 1,603 KB/sec with interval of half second. We indicate maximum since not all the ports on the switches are active. For the fabric discovery, the traffic will be a maximum of 801 KB/sec when using interval of 1 second for updating fabric. Given that the OSC clusters are capable of 56 Gbps to 100 Gbps, this is an insignificant amount of data.

5 RELATED WORK

In the world of supercomputing, as it stands today, there is a need and lack of full-stack monitoring tools. Monitoring MPI jobs down to the port counters are essential for debugging job failures and finding network bottlenecks. In this section, we give details of existing tools (in literature and those available as products) other than those that we have already mentioned in Section 1. The Texas Advanced Computing Center has developed a tool called *TACC STATS* [8]. This tool correlates job and system level statistics to create reports. The reports are used to identify issues in the jobs or systems infrastructure, but, not in real time. In [23], the authors proposed a suite built on MPI and allows for different metrics for each job and thus may not provide a full system view of how the jobs are interacting with the fabric. The authors in [13] describe a library that goes beyond the PMPI interface to gather MPI states and lower-level network statistics. Paraver [9] is used to provide visualization of the data. However, the tool could not visualize / model network activities. NVIDIA released [20] to support fine grained analysis tool for GPUs but it has portability issues. Further, it is complicated for typical application developers. Authors of [19] proposed a profiler on the top of LLVM to instrument CUDA code, however, it does not provide insight into MPI level. As described in Section 1, there are several tools that allow systems administrators to analyze and inspect high-performance networks such as Mellanox, FabricIT, BoxFish, Nagios, or Ganglia. With all of these tools, there is no integration with the MPI Library, so correlating network traffic to MPI jobs is a manual endeavor. On the other hand, there are several tools such as TAU, HPCToolkit, Intel VTune, IPM, and mpiP that focus on the MPI applications but do not show the network traffic of the MPI job.

6 SOFTWARE AVAILABILITY AND DEPLOYMENT

The proposed design has been released publicly as a part of OSU INAM Tool v0.9.5 and is available for free download and use from the project website [6]. So far this version had over 700 downloads from the project site. OSU INAM is currently deployed at the Ohio Supercomputer Center to monitor multiple HPC clusters. We are working with other institutions to deploy OSU INAM on their clusters.

7 CONCLUSION AND FUTURE WORK

In this paper, we proposed several enhancements to OSU INAM which enabled gathering, storing, retrieving, and visualization of the metrics for large and complex HPC networks with low latency. We enabled scalable and high performance real-time fabric profiling and discovery by using different levels of parallelism, bulk storage interactions, and independent parallel components. The enhanced OSU INAM tool provides low overhead, scalable, and fined-granularity InfiniBand port counter inquiry and fabric discovery for HPC users, system administrators and HPC developers. Our experiments show that, for a cluster size of 1,428 nodes and 114 switches the proposed design can gather fabric metrics at sub-second granularity.

The paper designs have been publicly released as a part of 0.9.5 release and are being used by many users. As part of future work, we aim to address the storage bottleneck by exploring high-performance database engines capable of handling concurrent reads and writes while ensuring high availability. Moreover, we plan to support more intra-node metrics and correlation between high-level metrics and low-level metrics to capture the interaction of applications, MPI and the network level metrics.

ACKNOWLEDGMENTS

This research is supported in part by NSF grants #CNS-1513120, #ACI-1450440, #CCF-1565414, and #ACI 1664137. We thank Nick Sarkauskas, Mark Arnold, and Meagan Haupt for their efforts in the project.

REFERENCES

[1] [n.d.]. Ganglia Cluster Management System. http://ganglia.sourceforge.net/ Accessed: June 19, 2020.
[2] [n.d.]. Integrated Performance Monitoring (IPM). http://ipm-hpc.sourceforge.net/
[3] [n.d.]. Mellanox CS7500 Switch Series. http://www.mellanox.com/page/products_dyn?product_family=191&mtag=cs7500 Accessed: June 19, 2020.
[4] [n.d.]. mpiP: Lightweight, Scalable MPI Profiling. http://www.llnl.gov/CASC/mpip/.
[5] [n.d.]. OpenSM. https://github.com/linux-rdma/opensm Accessed: June 19, 2020.
[6] 2020. OSU INAM. http://mvapich.cse.ohio-state.edu/tools/osu-inam/ Accessed: June 19, 2020.
[7] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, Mahesh Rajan, Michael Showerman, Joel Stevenson, Narate Taerat, and Tom Tucker. 2014. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications *(SC '14)*. IEEE Press, Piscataway, NJ, USA, 154–165. https://doi.org/10.1109/SC.2014.18
[8] B. Barth, T. Evans and J. McCalpin. [n.d.]. TACC STATS. https://www.tacc.utexas.edu/research-development/tacc-projects/tacc-stats.
[9] Barcelona Supercomputing Center. [n.d.]. Paraver. http://www.bsc.es/computer-sciences/performance-tools/paraver.
[10] Ohio Supercomputer Center. 1987. Ohio Supercomputer Center. http://osc.edu/ark:/19495/f5s1ph73
[11] HPCToolkit. 2019. http://hpctoolkit.org/ Accessed: June 19, 2020.
[12] Intel Corporation. [n.d.]. Intel VTune Amplifier. https://software.intel.com/en-us/intel-vtune-amplifier-xe.
[13] Rainer Keller, George Bosilca, Graham Fagg, Michael Resch, and Jack J. Dongarra. 2006. Implementation and Usage of the PERUSE-Interface in Open MPI. In *Proceedings, 13th European PVM/MPI Users' Group Meeting (Lecture Notes in Computer Science)*. Springer-Verlag, Bonn, Germany.
[14] Lawrence Livermore National Laboratory. [n.d.]. PAVE: Performance Analysis and Visualization at Exascale. https://computation.llnl.gov/project/performance-analysis-through-visualization/software.php.
[15] Allen D. Malony and Sameer Shende. 2000. Performance Technology for Complex Parallel and Distributed Systems. In *Proc. DAPSYS 2000, G. Kotsis and P. Kacsuk (Eds)*. 37–46.
[16] Mellanox Integrated Switch Management Solution. [n.d.]. http://www.mellanox.com/page/ib_fabricit_efm_management.
[17] OSU InfiniBand Network Analysis and Monitoring Tool. [n.d.]. http://mvapich.cse.ohio-state.edu/tools/osu-inam/.
[18] Pouya Kousha, Bharath Ramesh, Kaushik Kandadi Suresh, Ching-Hsiang Chu, Arpan Jain, Nick Sarkauskas, Hari Subramoni and Dhabaleswar Panda. 2019. Designing a Profiling and Visualization Tool for Scalable and In-Depth Analysis of High-Performance GPU Clusters . In *International Conference on High Performance Computing, Data, and Analytics*.
[19] Du Shen, Shuaiwen Leon Song, Ang Li, and Xu Liu. 2018. CUDAAdvisor: LLVM-based Runtime Profiling for Modern GPUs. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization* (Vienna, Austria) *(CGO 2018)*. ACM, New York, NY, USA, 214–227. https://doi.org/10.1145/3168831
[20] Mark Stephenson, Siva Kumar Sastry Hari, Yunsup Lee, Eiman Ebrahimi, Daniel R. Johnson, David Nellans, Mike O'Connor, and Stephen W. Keckler. 2015. Flexible Software Profiling of GPU Architectures. *SIGARCH Comput. Archit. News* 43, 3 (June 2015), 185–197. https://doi.org/10.1145/2872887.2750375
[21] Hari Subramoni, Albert Mathews Augustine, Mark Arnold, Jonathan Perkins, Xiaoyi Lu, Khaled Hamidouche, and Dhabaleswar K Panda. 2016. INAM 2: InfiniBand Network Analysis and Monitoring with MPI. In *International Conference on High Performance Computing*. Springer, 300–320.
[22] Hari Subramoni, Albert Mathews Augustine, Mark Daniel Arnold, Jonathan L. Perkins, Xiaoyi Lu, Khaled Hamidouche, and Dhabaleswar K. Panda. 2016. INAM2: InfiniBand Network Analysis and Monitoring with MPI. In *ISC*.
[23] Virtual Institute - High Productivity Supercomputing. [n.d.]. HOPSA: A Holistic Performance System Analysis. http://www.vi-hps.org/projects/hopsa/overview.