Survey of Open Source Tools for Monitoring I/O & Storage Performance of HPC Systems

Anil Kumar Gupta, Basit Budroo, Sakshat Shinde, Puneet Bakshi, Priya Joshi, Rutik Pol

Abstract — *One of the challenging things in today's High-Performance Computing is overcoming the I/O bottlenecks in traditional Parallel File Systems. The CPU and Memory have improved compared to I/O Systems, tremendously due to which data-intensive applications spend much time performing I/O. The I/O bottlenecks take away a lot of the system's performance capability, so it is necessary to identify these bottlenecks so the efficiency of data-intensive applications can be improved on HPC. This Literature survey presents an overview of various performance monitoring tools that can be employed to capture Performance Metrics. Furthermore, it also introduces the concept of using these tools in harmony to extract a broad set of data points to figure out potential bottlenecks in a system. In this paper, we report numerous open-source tools which are to monitor and test I/O's performance in HPC Systems. The numerous discussed in this paper are* `iosnoop`, `iolatency`, `execsnoop`, `syscount`, `bitesize`, `uiuc-recorder`, `iotop`, `collectl`, `collectd`, `grafana`, `hpcmd`. *These tools run as a daemon on every compute node, perform measurements at regular intervals, and compute derived metrics. Estimating I/O usage of the entire cluster may not be the easiest thing to do, but these tools can help.*

Keywords: I/O Monitoring Tools, Bottlenecks, High-Performance Computing (HPC) Storage, Performance Metrics, Parallel File System.

Anil Kumar Gupta, Senior Member IEEE, Center for Development of Advanced Computing, Pune, India
Email: anilkgupta@iee.org

Basit Budroo, Sakshat Shinde, Priya Joshi, Rutik Pol
MIT Academy of Engineering,
Center for Development of Advanced Computing,
Pune, India
Email: {bbbudroo, sakshat-shinde, pljoshi, rppol} @mitaoe.ac.in

Puneet Bakshi, CSE Department, IIT Guwahati
Email: b.puneet@iitg.ac.in

*B. Budroo and S. Shinde contributed equally to this work.

I. INTRODUCTION

High-Performance Computing, generally refers to consolidating computing resources in a form that remits much higher performance than a generic standalone desktop or workstation. HPC takes care of significant issues in a variety of fields namely, business, engineering, and science [1]. A Compute Cluster group comprises a lot of freely or firmly associated compute nodes that work together to form a single framework in many regards. HPC Solutions have three main components [2].

- Compute
- Network
- Storage

Be that as it may, I/O execution has not picked up as much improvement as have CPU and memory. Processors continue to improve their performance than I/O devices. For instance, the CPU cycles and primary memory (RAM) access times take up few nanoseconds at worst. Meanwhile, the I/O latency of an ordinary disk has milliseconds as magnitude. As a result, I/O acts as a performance bottleneck for many data-intensive applications rather than a CPU or memory [3]. Since I/O Systems are the performance bottleneck, evaluating and estimating I/O Systems has become a significant issue confronting the HPC community and the tools that capture and analyze I/O activity, are highly desired. Therefore, the perspective of making system-wide performance equivalent with CPU speed is becoming less credible [18]. Fig 1 plots disk performance, depicted by the Throughput of accessing a random 8 KB block of data with IBM's mainframe CPU performance [19]. Apparently, IBM's mainframe CPU performance has improved by over 3000% whereas IBM disk performance has hardly doubled over the earlier two decades, If CPU performance advances at its current pace and disk performance continue to obtain modest improvements, eventually, all applications that do any input or output (I/O)

will be limited by that I/O component and further CPU performance improvements will be wasted [18]. Meaning, applications will spend more time doing I/O.

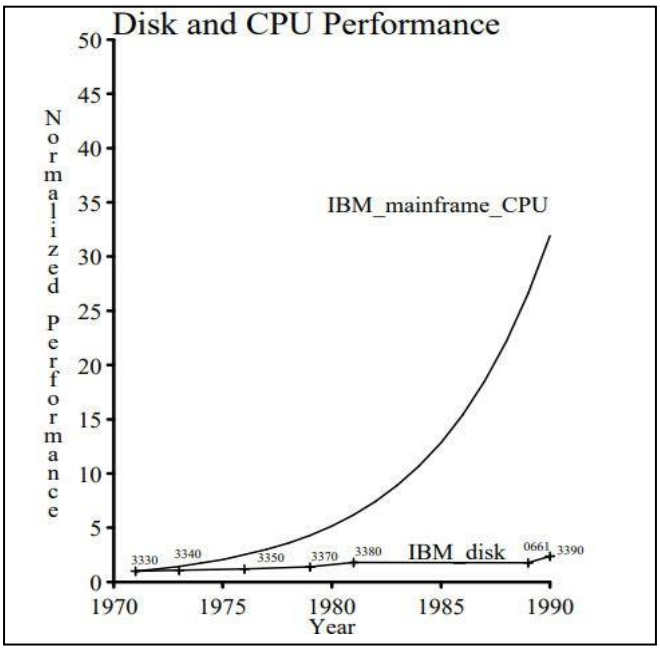

Fig 1. Comparing Trends of CPU and Disk Performance Improvements [19].

The remainder of this paper is arranged as explained below: Section II deals with the related work in I/O and Storage performance. We discuss various performance metrics in Section III. In Section IV, we list out and describe multiple performance monitoring tools that can be used, and in Section IV, we summarize our findings and outline future scope.

II. RELATED WORK

There are various monitoring tools widely used to monitor I/O behaviour, the latest widely-used being Darshan [23]. Darshan is an I/O characterization giving a reflection of the application behaviour, including patterns of access within files with minimal overhead and gathers statistics on several levels of I/O stack, primarily I/O Middleware (MPI-IO) and POSIX I/O. Luu et al. have implemented an I/O tracing and trace data analysis framework which has multiple levels and can help understand the behaviour of the application and I/O subsystems, called Recorder [24]. The Recorder is equipped with tools to capture I/O function calls at multiple levels of I/O stack, namely HDF5, MPI-IO, and POSIX I/O. He et al. have revealed the limitations of existing I/O metrics and introduced a new I/O metric, Blocks Per Second (BPS), to measure the performance of I/O systems [3]. Existing I/O metrics are less able to catch characteristics of I/O system performance. Still, BPS provides the overall I/O system performance, not the file system performance or disk performance. Chen et al. have stressed upon the fact that current I/O benchmarks suffer from several persistent problems: they become obsolete in no time. Generally, they do not provide deep insights into the I/O system performance. Their proposal is a self-scaling benchmark capable enough to dynamically adjusts features of its workload according to the performance characteristics of the system and its predicted performance, which uses the results to estimate the performance quickly for workloads [25]. The RIOT I/O tracing toolkit [26] intercepts MPI-IO and POSIX I/O function calls and records timestamp, data size, and file offset details. The kit also includes a post-processor to create statistical and graphical reports displaying application I/O activity.

III. PERFORMANCE METRICS

Performance is characterized by how well a system performs or the overall efficiency of the system. Identifying the measurements relevant to the resource is an essential part of architecting or evaluating I/O system's performance and availability. There are various metrics under which we can measure the system's performance, such as response time, Bandwidth, Throughput, availability, IOPS. One of the most frequently used I/O performance metrics is Throughput [3]. Throughput is the metric that is usually used to characterize I / O output of random and small blocks. Throughput is segmented as [22]:

- Read IOPS: Read I/Os per second;
- Write IOPS: Write I/Os per second;
- Total IOPS: Sum of read and write I/Os per second.

The rate of data transfer by I/O is indicated by Throughput. It is generally measured in two ways: I/O rate and data rate. I/O rate is indicated by Input/output Operations Per Second (IOPS) measured in *access/second*, and the data rate designated by Bandwidth is measured in *bytes/second* or *megabytes/second* [3]. IOPS is also known as the storage array transaction rate or total throughput [27]. Bandwidth determines the capacity of a link, bus, channel, interface, or device and is defined as the amount of data which can be transmitted over a fixed time and measured in *bits/ second* or *bytes/second* [27]. Storage-system bandwidths are measured in *megabytes/second* (MB/s) or *gigabytes/second* (GB/s). Due to the consequences of packet overhead, and the network protocols, data often travels at less than theoretical Bandwidth because of protocol overhead [22]. IOPS proves to be the better performance metric, especially when each request's size is tiny, like Online Transaction Processing (OLTP) [5]. Similarly, Bandwidth is the preferred metric for applications where the size of each request is significantly larger.

Metric	Definition
Throughput	I/O system transfer rate. Measured in IOPS or Bandwidth
IOPS	Storage array transaction. Measured in access/second
Bandwidth	The capacity of a connection, bus, channel, interface or device. Measured in bits/second or bytes/ second.
Utilization	Amount of time device is busy with requests for service
Service time	Time required to process a specific request.
Queue time	Waiting time of the request.
Response time	The interval between the submission of the request and the response.
Workload skew	Percentage of the total I/O load as a percentage of total data capacity, where the sum of these values is 100%

[20, 21]. The other performance metrics include utilization, service time, queue time, response time, and workload skew [27]. Utilization measures the amount of time a device gets busy with service requests. Service time measures how long it takes to process a specific request. Queue time is the duration of time a *request* is waiting to be processed [27]. Response time is the interval between submitting a request and receiving a response. There are two types of response time: User response time and I/O response time. User response time, also known as client-side response time or host response time, is the user experience with the system and is the amount of time it takes a system to respond to a request. I/O response time is a storage-system-only metric and measures how long a single I/O (read or write) takes to complete. I/O response time includes time spent waiting in queues as well as time spent servicing a request [22]. A *workload* is the number of work units assigned to a resource over some time. Workload skew shows how much of the storage system handles the bulk of the workload. It is the percentage of the total I/O load as a percentage of total data capacity, where the sum of these values is 100% [27]. Total workload upon the total data capacity is called skew. Total workload and capacity percentage add up to 100 %. The diagonal (blue) dashed line in Fig 2 shows a linear distribution of workload over the available data area (storage capacity). The solid (blue) line, for a skew of 90%, just 10% capacity has 90% of the workload distribution.

Table 1. List of performance metrics.

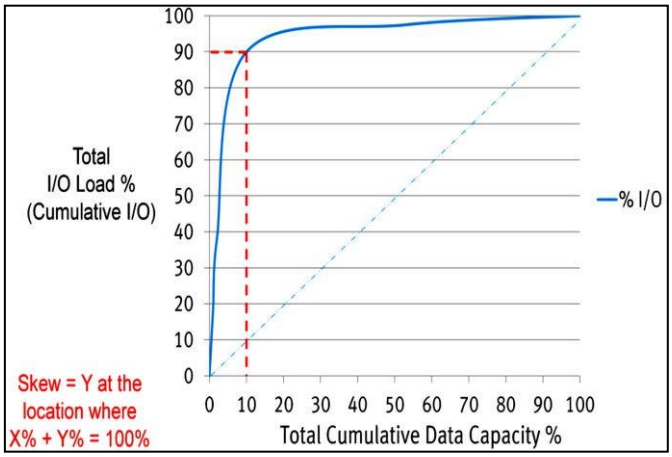

Fig 2. The plot of a workload skew showing 90% of I/O activity is performed on 10% of data. The x-axis is cumulative data, expressed as a percentage, and the y-axis is cumulative I/O expressed as a percentage [27].

IV. MONITORING TOOLS

Existing I/O tools fall into two categories. The first category of tools, trace and log each individual I/O operation but lack postprocessing capability. The second category of tools relies on profiling and sampling to reduce overhead but sacrifice details about access patterns being generated [23]. There are two main challenges in implementing a performance monitoring system on HPC. The first one is data-related, and the second one is regarding the deployment of such a system. The data-related challenge is to choose which metrics should be collected, how to interpret those metrics collected, and ultimately determine if there is a performance bottleneck. The deployment issue is to deploy a system that works reliably on the HPC cluster and effectively collects data from all nodes into a centralized database [6]. The following section discusses various Open Source tools that can help us capture multiple performance metrics, as discussed in the previous section, of High-Performance Computing Systems and log these metrics properly. By these metrics, we can analyze the performance of I/O and storage systems. These Open Source monitoring tools include `sysstat`, `Darshan`, `grafana`, `iosnoop`, `iolatency`, `execsnoop`, `syscount`, `bitesize`, `uiuc-recorder`, `iotop`, `collectl`, `collectd`, `hpcmd`, `iorate`, `fio`, `smartctl`.

No.	Name	Metrics	Open Source	References	Special Comments
1	sysstat	IOPS count, CPU load, number of tasks created, average load statistics by date, Virtual memory, paging and fault stats, CPU interrupt count, Network throughput, File system utilization	Yes	[28]	
2	darshan	Average I/O cost per Process, IOPS count, Access Size (POSIX)	No	[29]	Only Derivative works allowed.
3	grafana	_	Yes	[30]	It is a graphing tool, needs an external data source.
4	iosnoop	IO Latency	Yes	[31]	
5	iolatency	Latency distribution, Block I/O queue time.	Yes	[32]	
6	bitesize	Block I/O size Distribution	Yes	[33]	
7	syscount	Syscall count by syscall name	Yes	[34]	
8	uiuc-recorder	Number of I/O function calls	Yes	[35]	
9	iotop	I/O utilization	Yes	[36]	
10	iostat	IOPS count, storage utilization, IO-transfer per sec, req response latency	Yes	[37]	
11	hpcmd	GFLOP/sec, memory bandwidth, algorithmic intensity, network avg transfer rates, utilization and memory usage of GPUs	Yes	[38]	
12	fio	_	Yes	[39]	It is a benchmark utility.
13	smartctl	Prediction of storage device failure, throughput performance	Yes	[40]	It checks the SMART status of the

					storage devices.

Table 2. Various I/O Performance Monitoring tools available and discussed in the paper.

A. hpcmd

hpcmd is an HPC performance monitoring system. It is primarily used in the Max Planck Computing and Data Facility (MPCDF) to keep an eye on the performance metrics of all the jobs on HPC systems [6]. It is a lightweight middleware that runs as a daemon in each Compute Node background, performs measurements at regular intervals, and if necessary, calculates derived metrics [6].

hpcmd is written in plain Python and configurable via a YAML file. The Performance Metrics collected from each node are stored in Local Syslog, and from there, send to Central rsylog. Rsyslog collects Performance Metrics from each and every node in the cluster. Splunk, written in XML, is suitable for Data analysis and visualization [6]. It is supreme for crunching numbers in Performance Data collected from multiple nodes over long periods of time [6].

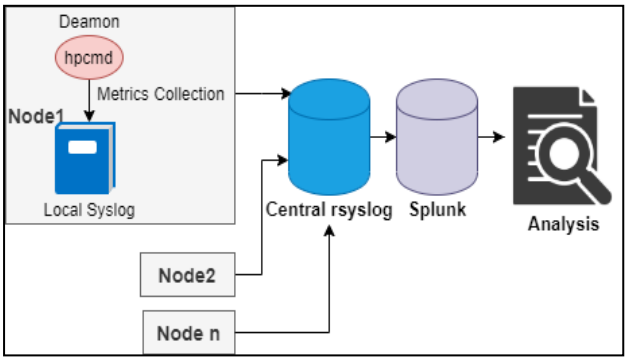

Fig 3. The MPCDF HPC Monitoring System architecture

Performance Metrics in hpcmd*:*

- CPU core events: CPUs provide performance monitoring units (PMUs) for each core. PMUs are programmable hardware units, and it induces only minimal overhead.
- CPU uncore events: Traffic monitoring between different sockets is achieved with uncore PMUs. These are usually present in modern-day CPU. To access the core and uncore counters, we use Linux perf subsystem.
- I/O: Parallel file systems are pivotal components of any HPC system. These are shared resources, and the wrong usage may affect the whole system. Monitoring the I/O operations and disk usage per node can give valuable hints at the usage patterns for that particular hardware system.
- Network: High-speed and High-bandwidth networks are mainly how HPC clusters manage to perform the task at hand.
- Software: Complemented by numerous system tools, a huge variety of application-based metrics can be accessed using Linux Kernel, e.g., the number of processes and threads started by a job, pinning of these tasks, memory usage, job's environment variables, and many more. hpcmd utilizes the ps and numastat under the hood.

Data Collection:

This involves an instance of the hpcmd middleware running as a system service in the background on each compute node, measuring at regular intervals and sequentially collecting data from the sources as mentioned above. Key-value pairs of measured values and derived metrics are written to the local Syslog file as log lines [6].

Data Aggregation:

Each node delivers hpcmd log lines via rsyslog. These are monitored, collected and finally fed into a central Splunk system.Log traffic can be transported via a separate ethernet link, not putting a load on the high-performance network reserved for applications.

A presentation report can be produced to make information available to the clients, for each activity, and given as a PDF file to download through a web server after login.

B. iosnoop

iosnoop is a tracing tool made for disk I/O events. Initially created for MacOS X and Solaris, later ported to Linux. It harnesses **ftrace** capabilities to provide a more straightforward set of human-readable metrics. Ftrace is an internal tracer, typically considered a function tracer, it is a framework for several tracing utilities. As can be seen,

iosnoop is just another building block for a whole stack of performance utilities.

Performance Metrics in `iosnoop`:

- Process IDs
  - PIDs on CPU when I/O was issued or inserted.
- BLOCK I/O
  - Disk block for the operation. It is the location relative to the device which is being monitored.
- Latency
  - Latency, meaning time taken for the I/O operation in milliseconds.
- Type of I/O
  - Read
  - Write
  - Metadata
  - Sync
  - Read-ahead
  - Flush
  - Force unit access
  - Discard
  - Secure
  - Null (not RWFD)
- COMM
  - Process name for the PID
- I/O queuing and completion timestamps.
- I/O start and completion timestamps.

C. iolatency

`iolatency` is a tool that summarizes the block device I/O as a histogram. It shows the distribution of latency, which in turn could be used to identify the latency outliers. This tool uses trace buffers in the kernel resulting in substantial overhead. A visual example is shown in Fig 4.

Performance Metrics in `iolatency`:

- Latency
  - Latency higher than & less than a value in milliseconds
- I/O Type
  - Only specific I/O operations measured according to the type (r, w)
- Block I/O queue time
- Visual Distribution
  - ASCII histogram

D. bitesize

`bitesize` shows I/O distribution for requested block sizes according to the process name. It outputs a histogram summarizing block I/O size. It works by tracing (dynamic) block I/O functions within the kernel, which are asynchronously copied into the user-space. The overhead for IOPS smaller than 10k is relatively negligible. Visual example in Fig 5.

Performance Metrics in `bitesize`:

- Kbytes
  - A range of Kbytes correlating to I/O operations
- Count of I/O

```
 >=(ms) .. <(ms)   : I/O      |Distribution                          |
      0 -> 1       : 4381     |######################################|
      1 -> 2       : 9        |#                                     |
      2 -> 4       : 5        |#                                     |
      4 -> 8       : 0        |                                      |
      8 -> 16      : 1        |#                                     |
```

Fig 4: iolatency

```
    Kbytes          : I/O      Distribution
      -> 0.9        : 0        |                                      |
  1.0 -> 7.9        : 38       |#                                     |
  8.0 -> 63.9       : 10108    |######################################|
 64.0 -> 127.9      : 13       |#                                     |
128.0 ->            : 1        |#                                     |
```

Fig 5: bitesize

    - Number of I/O operations in the given range
- Visual Distribution
    - ASCII histogram

E. iorate

iorate is a Dell EMC created tool, designed for storage benchmarking. This tool can play out a multitude of I/O tests on devices with different loads. A combination of config files is used to define each I / O test. The configuration files may be user-designed or default patterns. Multiple tests can be performed, each one running one after the other. iorate produces a comprehensive log file of the checks performed and a report file containing accurate statistics on each system test run [7].

Preparing for Testing:

A test is a group of I/O patterns applied to a device. For testing, certain things should be determined:

- Devices or filesystems need to test. It becomes a part of Devices File (devices.ior).
- Type of I/O test (Read/Write, Random/Sequential) to run. This information forms a Patterns File(patterns.ior).
- The sequencing of tests is stored in Test File(test.ior).

The three required configuration files are [7]:

- Devices.ior: Lists the devices to test. It defines the devices or files against which I/O we perform I/O.
- Patterns.ior: Types of I/O to run. It defines the different I/O types that can be performed on a device. We can use default I/O patterns or edit and choose our own. Patterns file describes each block size and type of I/O running in the test. Patterns can be read or write and random and sequential.
- Tests.ior: Lists the tests to run. I/O patterns performed on a device or group of devices are defined by this. Each test uses one or more patterns and is specified as a percentage of the I/O's to be performed.

Running the Tests:

iorate needs all of the configuration files described above to be configured and formatted without any errors. Testing is initiated by invoking iorate with optional configuration files names and parameters. If no configuration files are specified, iorate uses the default as mentioned above configuration files and iorate.perf and iorate.log as default output files [7].

Interpreting Test Results:

- Log File: iorate outputs a log file of the test we run. iorate.log is the default file name of a generated log file. The log file wraps around copy of config file(device_file,pattern_file,test_file). If we split these files into separate input files, they will produce the same test, i.e., it enables the test to be reproduced later.
- Performance File: It includes comprehensive statistics concerning each test run on each unit. Every line in the file represents data on a single computer/system for a single analysis. The first line is a header containing metadata concerning every test line. The default file name is iorate.pref.

F. sysstat

sysstat package is a collection of many tools/utilities to monitor system performance and usage-activity. sysstat is open source and freely available.

Several utilities make up the sysstat software application. Every utility has its particular function [8]:

- Iostat: Records CPU statistics and system input/output statistics.
- Mpstat: Records individual or combined statistics relevant to the processor.
- Pidstat: Statistical reports for Linux tasks (processes): CPU, I/O, memory.
- Sar: Gathers, records and stores information about system activity (CPU, kernel tables, NFS, interrupts, network interfaces, memory, disks TTY, sockets)
- Sadc: It stands for system activity data collector. It acts as a backend for sar.
- Sa1: Collect and store binary data in a routine data file of the system activity
- Sa2: Writes a summarized activity report daily.
- Sadf: displays multi-format sar collected data (CSV, XML), which helps to load and import performance data into a database.

The four major components used for the collection are sar, sa1, sa2, and corn. Sar is short for the system activity reporter, used to display results from data collected. Sa1 refers to the internal mechanism for carrying out the

statistical collection and writing the data at specified times to a binary file. Sa2 converts sa1 binary file to a human-readable format. After creating the binary file successfully, it is of utmost importance to set up a cron job; this will call the sa2 libraries to transform sa1 binary file to human-readable sar file [8].

G. collectl

`collectl` is a versatile performance monitoring tool. It can run as a daemon or in the form of an interactive session. The data captured using this tool can be stored in compressed or uncompressed form as per the user's demand. It includes options to format output in various ways. `Collectl's` complete data domain includes CPU interrupts, NFS shares, inodes, the Lustre file system, memory, sockets, TCP, and Infiniband statistics [9]. One thing to note — it only reports a snapshot of the data being recorded since the frequency at which `/proc` is written cannot be controlled by the tool.

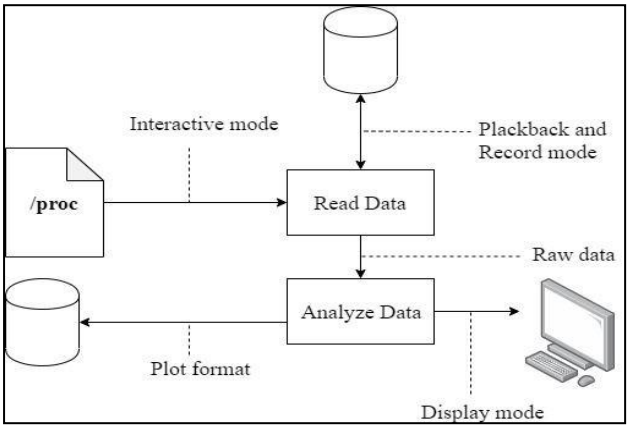

Figure 7. The architecture of the collectl Utility Tool.

The above block diagram shows that `collectl` reads data from /proc and writes its results into a file or terminal [10].

1. Interactive Mode: This is the default mode, and in this mode — the data is read from /proc and passes through *analyze*. It compares current and previous values. Also, it formats the outputs in a certain way.
2. Record Mode: Data passes from /proc the same way as in the interactive mode but is written to a file instead of going through the Analyze function. It is the most effective direction the data can take by removing the calculations and formatting.
3. Playback Mode: Here, collectl works identical to *Interactive Mode* except instead of reading data from /proc it reads it from a file.
4. Plot Format: When the -P switch is selected, it administers the formatting of the data to be delimiter separated (spaces by default) on a single line, starting with the date and time. -f switch is used to provide the location at which data is written. Default is terminal.
5. Socket: When running with -A and an address with an optional port number, `collectl` will send its data to the address specified.
6. Custom Output: collectl can bypass its standard routines for formatting interactive data and call a custom formatting routine instead.

The following are some of the `Collect's` features [11]:

1. Low overhead: `collectl` uses minimal CPU. When running as a daemon, its CPU utilization is below 0.1%. The overhead can increase when hundreds of processes are running on many disks.
2. Summary vs Detailed Data: We can report aggregated performance on many devices such as CPUs, Disks, interconnects such as Infiniband or Quadrics, Networks, and even the Luster File System. However, reports on individual devices can also be produced.
3. Brief vs Verbose Format: It is convenient to see fewer data, and for this, `Collectctl` provides a *brief format* as the default interactive display format, this helps one to understand what a variety of subsystems are doing on a single line, making it much easier to identify anomalies in the data by checking a column of numbers. If the user requires more details and is willing to look at multiple lines per sample, the *verbose* format is what they want.
4. Top-anything across a cluster: With colmux, `collectl` can run on multiple machines in the cluster and present an integrated view from all nodes, sorted by the column.

H. syscount

`syscount`, as the name suggests, counts the number of system calls performed. It can trace syscalls by

- Process name
- Syscall name

Different modes can be set by passing specific arguments (flags). The methods which only report syscall names generally have minimal overhead. These modes use in-kernel counts. Other modes create a data file which results in additional overhead.

Performance Metrics in `syscount`:

- PID
  - Process ids
- COMM
  - Process name
- SYSCALL
  - System call name
- COUNT
  - Number of syscalls

The primary reason to include this tool is to keep track of **open, read, write, close** syscalls, and the number of times they were invoked. By having a clear count of syscalls related to the I/O operations and extracting other metrics from I/O targeted tools, we can draw certain conclusions about the usage of particular storage systems [12].

I. smartctl

Storage devices include a function called **SMART** (Self-Monitoring Analysis and Reporting Technology). It is used to monitor disk status by using various methods and sensors present on the hardware. IBM designed it. It contains multiple attributes (measured values), which directly or indirectly affect Hard Disk health status. A range of onboard sensors continuously monitors hard disks.

Some of the parameters include:

- The raw Read error rate
  - It indicates a problem with hard drive surface, actuator, or the head: high RER, the higher chance of disk failure.
- Seek Error rate
  - It is not considered a critical error by most vendors.
- Write Error rate
  - It indicates issues with recording data to the disk.
- Current Pending Sector
  - Points out the number of disk sectors to be remapped since they can no longer be read.

SMART metrics determine if there are *Bad Sectors*, and with enough bad sectors, SMART predicts a possible failure.

`smartctl` utilizes SMART to detect errors and determine drive health and reports back to the user [13].

J. collectd

`collectd` runs in the background, periodically collects performance metrics from system and applications, and stores these values in various ways, such as RRD files. Collectd gathers multiple parameters that can be used to monitor the system and find performance bottlenecks.

Performance Metrics Collected:

- CPU utilization: Time spent within the system, user, idle, and connected states.
- Disk utilization: Sectors read/written, a variety of read/write operations, the average time taken up to complete an I/O operation.
- Memory utilization: It is memory occupied by running processes, page cache, buffer cache, and free
- CPU, disk, network, and memory I/O statistics from virtual machines.
- Network latency: Time required to reach the default gateway or host.

K. grafana

The tools listed above can gather a considerable amount of metrics. To make the full use of the collected data and extrapolate, *meaning* out of them is our primary goal. For this, `grafana` is used. It allows us to query, visualize, alert, and understand our metrics. It encompasses creating custom dashboards that suit our unique needs and enables them to share with other teams if needed [14].

Some key features include:

- Visualization of data
  - Client-side graphs
  - Panel plugins
- Dynamic Dashboard
  - Reusable dashboard configurations
- Exploring Metrics
  - Ad-hoc queries
  - Compare different time ranges, queries, and data sources in a split view.
- Exploring Logs
  - Searching through logs & live streaming
- Alerts
  - `grafana` provides the ability to structure alert rules for our essential metrics.
  - Notification forwarding to systems like Slack
- Mixed Data Sources
  - Ability to mix various data sources in the same graph

    - Specifying a data source per query
- Seamless integration with Databases
    - Graphite
    - InfluxDB
    - Prometheus
    - MySQL

These features make `grafana` a promising tool to visualize and analyze our metric data and draw conclusions from it reliably.

V. CONCLUSION & FUTURE WORK

This paper listed many tools that could ultimately be used to form a cohesive system granting us the ability to monitor finer details in a High-Performance Computing infrastructure. More so, the tools we discussed are targeting the I/O subsystem; using the data captured — one could gain more in-depth insight on how the system responds to a variety of workloads. The tools are employed to create an all-in-one monitoring tool, giving a visual representation of what is happening throughout the Input-Output stack. Using the information gathered, one could pin-point *points of failure*, identify bottlenecks, ease troubleshooting and also predict possible future anomalies using time series forecasting and classification.

VI. REFERENCES

1. Ministry of Electronics & Information Technology. 2017. *High Performance Computing (HPC)*. [online] Available at: <https://meity.gov.in/content/high-performance-computinghpc#:~:text=Performance%20Computing(HPC)-,High%20Performance%20Computing(HPC),science%2C%20engineering%2C%20or%20business.>
2. Netapp. 2020. *What Is High-Performance Computing?* [online] Available at: <https://www.netapp.com/us/info/what-is-high-performance-computing.aspx>
3. Shuibing He, Xian-He Sun, Yanlong Yin "BPS: A Performance Metric of I/O System" 2013 IEEE 27th International Symposium.
4. J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach: Morgan Kaufmann Pub, 4th edition, 2011.
5. D. Bitton, M. Brown, R. Catell, S. Ceri, T. Chou, D. DeWitt, D. Gawlick, H. Garcia-Molina, B. Good, and J. Gray, "A Measure of Transaction Processing Power," Datamation, vol. 31, pp. 112-118, 1985.
6. Stanisic L., Reuter K. (2020) MPCDF HPC Performance Monitoring System: Enabling Insight via Job-Specific Analysis. In: Schwardmann U. et al. (eds) Euro-Par 2019: Parallel Processing Workshops. Euro-Par 2019. Lecture Notes in Computer Science, vol 11997. Springer, Cham
7. 2005. *Iorate*. Open Systems I/O Driver and Measurement Tool. EMC CORPORATION.
8. Peck, J., 2009. *SYSSTAT Howto: A Deployment And Configuration Guide For Linux Servers - Linux.Com*. [online] Linux.com. Available at: <https://www.linux.com/training-tutorials/sysstat-howto-deployment-and-configuration-guide-linux-servers/>
9. Scott Helvick-Survey of Hardware Performance Analysis Tools
10. Collectl. 2018. *Collectl Architecture*. [online] Available at: <http://collectl.sourceforge.net/Architecture.html>
11. Collectl. 2018. *Collectl Features*. [online] Available at: <http://collectl.sourceforge.net/Features.html>
12. Courses.engr.illinois.edu. 2013. *System Calls And I/O*. [online] Available at: <https://courses.engr.illinois.edu/cs241/sp2014/lecture/05-syscalls_sol.pdf>
13. Linux.die.net. 2012. *Smartctl(8) - Linux Man Page*. [online] Available at: <https://linux.die.net/man/8/smartctl>
14. GrafanaLabs. n.d. [online] Available at: <https://grafana.com/docs/grafana/latest/getting-started/what-is-grafana/>
15. Gregg, B., 2014. *Iosnoop For Linux*. [online] Brendangregg.com. Available at: <http://www.brendangregg.com/blog/2014-07-16/iosnoop-for-linux.html>
16. Godard, S., n.d. *Iostat(1) - Linux Man Page*. [online] Linux.die.net. Available at: <https://linux.die.net/man/1/iostat>
17. Forster, F., n.d. *Documentation – Collectd – The System Statistics Collection Daemon*. [online] Collectd.org. Available at: <https://collectd.org/documentation.shtml>
18. P. M. Chen and D. A. Patterson, "Storage performance-metrics and benchmarks," in Proceedings of the IEEE, vol. 81, no. 8, pp. 1151- 1165, Aug. 1993, doi: 10.1109/5.236192.
19. J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach.
20. J. S. Vetter and A. Yoo, "An Empirical Performance Evaluation of Scalable Scientific Applications," in Proceedings of the 2002 ACM/IEEE conference on Supercomputing, 2002, pp. 1-18.
21. H. Song, Y. Yin, Y. Chen, and X.-H. Sun, "A Cost-Intelligent Application-Specific Data Layout Scheme for Parallel File Systems," in Proceedings of the 20th international symposium on High performance distributed computing, San Jose, California, USA, 2011, pp. 37-48
22. EMC.com. 2011. *EMC Unified Storage System Fundamentals For Performance And Availability*. [online] Available at: <https://www.dell.com/community/s/vjauj58549/attachments/vjauj58549/vnx/29570/5/EMC%20Unified%20Storage%20System%20Fundamentals%20for%20Performance%20and%20Availability.pdf>
23. P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang and K. Riley, "24/7 Characterization of petascale I/O workloads," 2009 IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, 2009, pp. 1-10, doi: 10.1109/CLUSTR.2009.5289150.
24. H. Luu, B. Behzad, R. Aydt and M. Winslett, "A multi-level approach for understanding I/O activity in HPC applications," 2013 IEEE International Conference on Cluster Computing (CLUSTER), Indianapolis, IN, 2013, pp. 1-5, doi: 10.1109/CLUSTER.2013.6702690.
25. Chen, P. and Patterson, D., 1994. A new approach to I/O performance evaluation. *ACM Transactions on Computer Systems (TOCS)*, 12(4), pp.308-339.

26. S. A. Wright, S. D. Hammond, S. J. Pennycook, R. F. Bird, 1. A. Herdman, I. Miller, A. Vadgama, A. Bhalerao, and S. A. Jarvis. Parallel File System AnaJysis Through Application I/O Tracing. Comput. J., 56(2):141-155, 2013.
27. 2014. *Storage Performance Fundamentals*. [online] EMC CORPORATION. Available at: <https://virtualizationandstorage.files.wordpress.com /2016/01/student-guide2.pdf>