DOI: 10.14529/jsfi210304

A Review of Supercomputer Performance Monitoring Systems

Konstantin S. Stefanov[1], *Sucheta Pawar*[2], *Ashish Ranjan*[2], *Sanjay Wandhekar*[2], *Vladimir V. Voevodin*[1]

High Performance Computing is now one of the emerging fields in computer science and its applications. Top HPC facilities, supercomputers, offer great opportunities in modeling diverse processes thus allowing to create more and greater products without full-scale experiments. Current supercomputers and applications for them are very complex and thus are hard to use efficiently. Performance monitoring systems are the tools that help to understand the efficiency of supercomputing applications and overall supercomputer functioning. These systems collect data on what happens on a supercomputer (performance data, performance metrics) and present them in a way allowing to make conclusions about performance issues in programs running on the supercomputer. In this paper we give an overview of existing performance monitoring systems designed for or used on supercomputers. We give a comparison of performance monitoring systems found in literature, describe problems emerging in monitoring large scale HPC systems, and outline our vision on future direction of HPC monitoring systems development.

Keywords: monitoring, supercomputers, performance monitoring, review.

Introduction

High Performance Computing is now one of the emerging fields in computer science and its applications. Top HPC facilities, supercomputers, offer great opportunities in modeling diverse processes thus allowing to create more and greater products without full-scale experiments. Current supercomputers are very complex, and their efficient usage is very complicated. One of the tools helping to understand the efficiency of supercomputing applications is the performance monitoring systems. They collect data on what happens on a supercomputer (performance data, performance metrics) and present them in a way allowing to make conclusions about performance issues in programs running on the supercomputer.

In this paper, a performance monitoring system is a software package that continuously gathers performance metrics for at least compute nodes of an HPC compute system. Data from other sources may be collected as well. Those data are then used to provide an insight into what is happening in a supercomputer from a performance viewpoint, for individual nodes, parts or the whole supercomputer or specific jobs. In this paper we do not consider as a performance monitoring system a system which is aimed at monitoring the health of the supercomputer, i.e. that the components of a supercomputer are in a state suitable for running jobs. Nagios [13] is an example of such a system. We also do not consider systems aimed at profiling and tuning the specific jobs and which collect performance metrics only for the job in consideration. PTP [53], Tau [46], and HPCTOOLKIT [23] are the examples.

This paper is organized as follows. In section 1 we give a brief overview of historic and current performance monitoring systems. In section 2 we compare the features of the systems described in section 1. In section 3 we try to outline the performance impact of large scale monitoring systems. In section 4 we focus on design directions of large scale monitoring systems. And then we give a conclusion.

[1]Lomonosov Moscow State University, Moscow, Russian Federation
[2]HPC-Tech Group, Centre for Development of Advanced Computing, Pune, India

1. An Overview of Performance Monitoring Systems

Let us introduce some terms to describe different monitoring systems in a consistent way.

A *node agent* is a part of a monitoring system that runs on compute nodes of an HPC cluster and gets data related to that node. If the data related to other parts of the supercomputer (like network equipment, storage systems, central servers, etc.) are needed, they may be obtained by some node agents or by some dedicated agents running not on a compute node. Sometimes a monitoring system is claimed to be *agentless*. Generally, it means that instead of a custom agent some standard part of an OS or a hardware being monitored is used, e.g. SNMP agent.

Data from node agents and agents collecting other data flow to a *central*, or *server part* of a performance monitoring system. In the server part, the data are processed and presented to a user via some interface (not necessarily a GUI).

Another important part of a performance monitoring system is data storage. It is used to store the performance data and to process them in a case when some past data are needed.

Performance metrics are data about performance. The most frequent type of data is numeric data. A *primary metric* is a metric obtained directly from a data source: Operating System, hardware, etc. CPU loads, free RAM are the examples. A *secondary* or *derived*, metric is a metric which values are calculated based on values of primary and/or other secondary metrics. For example, a maximum value of free RAM over some period of time is an example of a derived metric.

Performance data from agents can be transferred to the server part of performance monitoring systems in *push* or *pull* modes. A *push* mode is a mode when agents send data to the server part without a command (by their own schedule). A *pull* mode is when agents are queried for data by the server part. There may be mixed cases: for example, node agents work in push mode while data from network equipment are obtained via SNMP in pull mode. A more elaborate example of mixing the modes is given in [22].

Concrete performance monitoring systems will be considered further. The performance monitoring systems we selected for this section were chosen with the following guideline in mind. We selected a specific tool if there is a scientific paper describing it or it is referenced in a paper describing other tool and has an active website. We do not consider stale systems like CLUMON although it is referenced in other papers as we did not manage to find a paper describing it and its website is only available as a copy in the Internet Archive [1].

1.1. PARMON

PARMON [28] was introduced in 1998. It is a tool for monitoring a cluster of UNIX workstations. It was used in Center for Development of Advanced Computing (C-DAC) in India for PARAM 10000, a HPC cluster consisting of nodes with UltraSPARC CPUs working with Solaris OS.

PARMON includes two main components.

Parmon-server runs on every node (a node agent) and provides information about node performance metrics on request in a client-server fashion (pull mode).

Parmon-client is a custom Java-based GUI client which connects to *parmon-servers* running on nodes, retrieves information and presents it to a user. It can collect and present information on one specific node or on a set of nodes.

All performance data processing is done by the *parmon-client* online while running a GUI session for the user. There is no database for storing data for subsequent analysis.

1.2. SuperMon

SuperMon [41, 50], introduced in 2001, is described as 'a flexible set of tools for high speed, scalable cluster monitoring'. It can be considered as one of the first monitoring systems designed for HPC clusters, or at least the first which gained wide visibility. It was created in the Advanced Computing Laboratory of Los Alamos National Laboratory and tested on 128 nodes Alpha Linux cluster. It is not a full performance monitoring system as it provides only the way to obtain and aggregate performance data; no processing of those data is described.

SuperMon includes a Linux kernel module that produces performance monitoring data, *mon* – node level data server and *Supermon* – several node data concentrators. These components form a tree-like hierarchical structure in which *mon* retrieves data from the kernel module on the same node, *Supermon* retrieves data from several *mons* or *Supermons*.

SuperMon has some interesting features.

All its components (kernel module, *mon* and *Supermon*) speak the same text-based protocol. It uses s-expressions introduced in LISP programming language.

Its components form a hierarchy that can be used for multi-level aggregation of monitoring data thus making SuperMon scalable.

SuperMon was tested on a heterogeneous cluster. The cluster was composed of nodes with 1, 2 and 4 Alpha CPUs [50]. But this point is not discussed in SuperMon paper. One of the possible reasons is that the paper does not describe the part which processes monitoring data, only the means to get the data are described. And the specifics of monitoring a heterogeneous cluster should be in a data processing part.

SuperMon was tested for extremely high (even for today) data sampling rates up to 3500 Hz. Somewhat counterintuitive, SuperMon authors claim that the higher the sampling rate the less the perturbation it causes to user jobs. Unfortunately only the theoretical support for this claim is given and no experiments have been done.

Despite all those features SuperMon is just a tool for retrieving and aggregating data, no database that may store the data for future analysis is not described.

1.3. Ganglia

Ganglia is one of the oldest but still widely used cluster performance monitoring systems. Its description [37] was published in 2004 when Ganglia was quite mature. The oldest release announcement (`http://ganglia.info/?m=200202`) found on Ganglia website is the announcement of version 2.0.3 in February 2002. The website [4] shows an impressive list of organizations using Ganglia.

Ganglia can be used for clusters and federation of clusters. Its components are *gmond*, agent running on nodes, and *gmetad*, which aggregates data from *gmonds* or other *gmetads*.

Communication between *gmetad* and *gmond* is done via multicast UDP. This allows several data receivers to operate on data coming from a single data producer or a single set of producers from several nodes. *Gmetad* may poll for data and *gmonds* respond, or *gmonds* send data by themselves. Multiple *gmetads* from a tree hierarchy and communicate via TCP. All data are transmitted in XDR encoding. A special crafted client library communicates to *gmetad*. *gmetad* store the collected data using RRDtool [20], a tool for storing and visualizing time series data.

Ganglia also has a web front-end that can show collected data for all or part of the nodes.

Ganglia authors provide experimental data for the overhead (they call it local overhead) incurred by *gmond* running on compute nodes. They measure the overhead as a percentage of CPU load consumed by *gmond* and its size in physical and virtual memory. The reported numbers are up to 0.4 % of CPU load, 16 MB of physical RAM and 16.7 MB of virtual memory (maximum values for different clusters).

1.4. NWPerf

NWPerf [42] was introduced in 2004. At that time it was used on MPP2 [10], 977-node Linux 11.8 TFLOPS cluster located at Pacific Northwest National Laboratory (PNNL).

Generally, NWPerf data collecting part looks very similar to that of Ganglia. NWPerf uses agents running on compute nodes that send XDR-encoded data via multicast UDP to data receivers. Data are stored in PostgreSQL [18] relational database thus allowing using SQL for data processing.

NWPerf adds data about a job being run on a cluster to that database. By combining performance data and data about jobs it becomes possible to analyze the specific jobs performance and calculate performance metrics of jobs.

One of NWPerf design goals was to lessen the overhead (perturbation in time of jobs) caused by the monitoring system. NWPerf authors did quite an extensive experimental evaluation of such overhead. They used MPI collective operations slowdown as a measure of the influence of the monitoring system on user jobs. To reduce the overhead they synchronized the moment when all node agents wake up to collect and transmit performance data. As a result they claim that they can collect performance data once per minute causing not more than a 1 % slowdown of user jobs on a 1000+ node cluster.

1.5. Ovis-2

Ovis-2 [26], introduced in 2008, is a redesign of Ovis [16]. Ovis is a monitoring system with sophisticated analytic tools allowing setting complex conditions on collected metrics as a trigger for notification and reaction. Ovis-2 is a framework for performance cluster monitoring and analysis of performance data. It addresses scalability and fault-tolerance.

Ovis-2 contains *sheep* processes (node agents) that run on components and collect data. *Shepherd* is an aggregator which gets data from *sheep* and stores them in a database.

Sheep, when run, search for a *shepherd* in mDNS (local multicast name-resolving system) and register themselves with a random *shepherd*, which, in turn, may redirect to another *shepherd*. After registering *sheep* begin to push performance data. Each *sheep* has all possible data source library compiled-in and on startup tries to instantiate as many data sources as possible. *Shepherds*, in turn, aggregate data and store them into a replicated MySQL [12] database.

Scalability is achieved by distributing the load from many *sheep* between *shepherds*. Fault-tolerance is achieved by many *shepherds* which may take over other failed *shepherd*, and by replicated database storage.

GUI and analytical tools for collected data analysis are mentioned in the paper. They are oriented on analysis of the whole cluster health. The job-centric view is mentioned in the Future Work section of [26].

Ovis and Ovis-2 do not seem to be developed further, but one of their component, Lightweight Distributed Metrics Service (LDMS) became an independent tool (section 1.8).

1.6. TACC Stats and SUPReMM

TACC Stats [31, 32], introduced in 2011 by Texas Advanced Computing Center, is a package that collects performance-related data from compute nodes of a cluster and presents them in a job-centric view. TACC stats consists of four components: *monitor*, *pickler*, *analysis*, and *site*.

Monitor is a small modular executable aimed to collect performance data. It is run in the job prolog, every ten minutes, and the job epilog. *Monitor* stores the collected data locally on every node.

Pickler runs every 24 hours to collect the data saved by *monitors* on compute nodes and stores them in central storage in per-job files. The data are stored in Python pickle format.

Analysis is a set of tests and plotting routines that can be run on a set of jobs to show possible performance issues in tested jobs.

Site module is a web interface to the data provided by TACC stats.

As a part of SUPReMM project [27], TACC stats was integrated with Open XDMoD [15, 43], a tool aimed at providing per-job data from cluster scheduler. With such integration, Open XDMoD is able to enrich its data with the performance data from TACC stats.

To map jobs to software packages and other properties of the executable files used for the job, a XALT [25] tool is used. XALT can collect the data like the libraries used for build the binary used for the job, the compiler used to compile it, tries to determine the exact version of the software package used, etc.

1.7. Dataheap

Dataheap, presented in 2012 paper [33], is focused on combining performance monitoring data from compute nodes and other, mostly I/O-related, sources like RAID controllers, storage area networks (SAN) switches, parallel file systems, etc.

It has compute node agents which send performance data from nodes and agents which send performance data from other sources. All those data are processed to calculate secondary data and then stored in a database. MySQL and SQLite are mentioned as possible choices. Additional tools exist for access to stored data like standalone GUI application, PHP-based web interface, and a command-line tool.

Dataheap authors pay special attention to the calculating of secondary values (or derived metrics). These are values that are not received directly from some sources like OS and hardware (primary values) but are calculated based on them. Such secondary metrics can be aggregated like average over a time interval or updated when a single new value of primary metric arrives. The latter case is specifically considered by Dataheap authors. Dataheap can calculate such secondary metrics on-the-fly. A scheme is proposed for updating secondary values based on interpolation of primary values.

1.8. Lightweight Distributed Metrics Service (LDMS)

Lightweight Distributed Metrics Service (LDMS) [24] is the data collection, transport, and storage component of Ovis (section 1.5). LDMS, as a separate component, was introduced in 2014. At the time of publication it was used on The National Center for Supercomputing Applications (NCSA) Cray XE6/XK7 Blue Waters and Sandia National Laboratories Linux cluster Chama. Blue Waters consists of 24 648 nodes and Chama consists of 1296 nodes.

LDMS is comprised of *ldmsd* daemons which can be configured to run in either *sampler* or *aggregator* modes.

A *sampler* includes sampling plugins each combing a specific set of metrics. *Samplers* can run several sampling plugins. The sampling frequency is used-defined and can be reconfigured on-the-fly.

Aggregators collect data from *samplers* and/or other *aggregators* in pull mode. They can use TCP, InfiniBand RDMA or Crays Gemini RDMA transports. Aggregation frequency is set on startup and cannot be changed without a restart. *Aggregators* support failover connection to samplers to take over data pulling if another *aggregator* is down.

The collected data are stored by *aggregators* with storage plugins. Possible storage formats are MySQL, Comma Separated Value (CSV) files and Scalable Object Store (SOS) [5].

1.9. LIKWID Monitoring Stack

Originally, LIKWID [52] was a set of tools for getting hardware performance counters for a specific job running on an x86-based compute node. After some development by 2017, it was transformed to a LIKWID monitoring stack [44] (LMS) which collects performance data from nodes and presents them in a per-job view.

LMS contains a host agent which uses a component from the original LIKWID to obtain performance data. The data from nodes or other sources like servers are sent to a central router. The router receives a signal about job start and finish from the cluster resource management system. The router is responsible for tagging the performance data with job id tags and storing them in a database. Another function of the router is to pull performance data from the sources which do not support push mode of operations like Ganglia *gmond* and mix that data with the data received from sources using push mode.

All data are saved in InfluxDB [7] time-series database. The protocol used in communication between LMS components is the InfluxDB line protocol. The protocol is based on HTTP and is widely used. Hence adding a filter into the data processing chain should be quite easy.

The performance data stored in InfluxDB are tagged with hostnames and job ids. These data are used to represent the metrics on all or part of the cluster or specific jobs. Grafana [6] toolkit is used to create charts and dashboards and show them to a user.

1.10. Performance Co-Pilot

Performance Co-Pilot (PCP) [17, 40] is a performance monitoring toolkit tracing its history since 1999 and is still being actively developed. While not targeted specifically for HPC clusters and supercomputers, it is often compared to while describing HPC performance monitoring systems – that is the reason for including it in this review.

PCP includes a Performance Metrics Collector Daemon (*pmcd*), an agent running on hosts to be monitored. *Pmcd* includes several Performance Metrics Domain Agents (*PMDA*), which are dynamically loaded libraries with a specified API. Each PMDA is responsible for collecting metrics from a performance metric domain like a kernel, a database server, etc. PCP uses pull mode: a client application connects to *pmcd* and requests some metrics. *Pmcd* routes the request to the appropriate *PMDA* and returns the response.

One of the client applications included in PCP is *pmlogger*, which can create performance metrics archive. One interesting feature of PCP is the ability to replay the archive created by *pmlogger*, thus enabling to reproduce the events from the past.

Other modules of PCP include *pmie* which can make notification on rules for metric values; *pmie* to generate periodic reports and a PCP GUI package.

1.11. Examon

Examon [3, 29], which stands for Exascale Monitoring, is a framework for the performance monitoring of HPC systems. Its distinctive feature is that it uses MQTT [11] protocol as a transport for performance data.

Its main executable *pmu_pub* collects performance data and publishes it to the MQTT broker. Then the data are exported to KairosDB [8] (a NoSQL time-series database built on top of Apache Cassandra [35]) and can be retrieved or analyzed by other tools which are not part of the project.

As MQTT uses publish/subscribe model, other clients can use the data in parallel to storing them in a database to make other services like alerting.

1.12. DiMMon and TASC

Distributed Modular Monitoring (DiMMon) [51], introduced in 2015, is a framework aimed at creating performance monitoring configurations. It is designed to be modular in all aspects. It has several types of modules: sensors that collect performance data, processing modules that make some calculations with the data, communication modules that send or receive data to components of the system working on other nodes. But this distinction is purely logical: from the frameworks point of view all modules are the same and developed using the same API. This modular design allows to create different paths for different data or having several databases for data aggregated on different periods of time.

DiMMon is used as a data source in TASC [47, 48]. TASC (Tuning Application for Supercomputers) is a system for visualizing performance monitoring data and produce advice to users if there are some performance issues in their jobs. DiMMon pushes data to TASC, which uses PostgreSQL and MongoDB [21] to store performance data. Redash [19] is used for data visualization.

1.13. C-CHAKSHU

C-CHAKSHU, a multi-cluster monitoring platform, was introduced as a part of the software stack for the National Supercomputing Mission (NSM) [14], which is implemented by C-DAC and supported by the Ministry of Electronics and Information Technology (MeitY) and Department of Science and Technology (DST), Government of India, in the year 2019. It monitors multiple HPC/Supercomputing systems, which are geographically separated, from a single dashboard. Currently, this tool is deployed on HPC systems installed at different scientific institutions and research organizations across India under the NSM project. C-CHAKSHU designed intelligently to address and cater to the needs of different users ranging from system administrators, application developers, domain scientists, and system architects. It has scalability for the pre-exascale/ Petascale system.

This tool collects real-time system-wide performance metrics, compute nodes health assessment-related metrics with action, verification of essential service/daemons. In addition, it also collects job/application execution snapshots and related performance counters, information from resource managers, and presents them in a Dashboard.

In C-CHAKSHU, two major components are used, one is on the server node to pull data whenever needed. The second component runs on all compute nodes all the time which collects different system-wide metrics and other subsystems over a network. Furthermore, Compute Nodes process data on their own and insert it into a NoSQL database whereas the aggregation of data for system-wide monitoring is carried out by the single server. C-CHAKSHU only causes a negligible network overhead on the cluster management node, incurs no overhead on the computing nodes, and reveals insightful knowledge of how HPC components interact with each other.

C-CHAKSHU has a loosely coupled architecture that supports the integration of any third-party tool and different back-ends easily. This tool has more capability of analyzing system performance and identification of bottlenecks.

2. Comparison of Performance Monitoring Systems

In this section we will compare the performance monitoring systems from several points of view.

First, we will outline data sets available in performance monitoring systems. Second, we will outline similarities and differences in data path inside monitoring systems and in the modes used to obtain the data. Next, we will compare the methods used to measure an overhead incurred by performance monitoring systems. And finally, we will give a table summarizing the comparison.

2.1. Metrics Used for HPC Performance Monitoring

The full list of metrics collected by a monitoring system is generally not easy to obtain: it is not usually included in a paper. Frequently, the list can be mined from the documentation if it is available. And still the exact metrics often depend on OS, on the hardware, etc. So we will not try to provide the full exact lists of metrics. Instead, we will outline the general groups of metrics used for performance monitoring.

The first group is what NWPerf authors call 'vmstat like information': 'percent of time spent on kernel processes, percent of time spent on user space processes, swap utilization and swap blocks in and out, and block device KB in and out' [42]. In modern Linux-based OS and some UNIX variants these metrics are available via `/proc` interface. Generally these metrics include CPU usage parameters (up to 10 metrics); network interface counters (bytes in/out, packet in/out, errors), load average for 1, 5, and 15 minutes; various types of memory usage (free RAM, RAM active, RAM inactive, swap usage, etc.), block device statistics. This group is basic for any performance analysis, and all performance monitoring systems mentioned in section 1 provide such data.

The second group is metrics provided by Performance Monitoring Units (PMU), Performance Monitoring Counters (PMC), Hardware Monitoring Counters (HMC), or simply hardware counters. All these are the names for basically the same thing: a set of hardware resources in modern CPUs that can count hardware events which occur while the program is being executed. Examples of these metrics are: a number of CPU cycles, a number of executed operations, a number of floating-point operations, a number of last level cache misses, etc. While a CPU can support a large number of events (several hundreds) that can be counted in general, a number of events that can be counted simultaneously is small. For example, CPUs based on Intel Haswell microarchitecture have three fixed-purpose counters (an event that is tied to that counters cannot be changed) per thread and four general purpose counters (an event to count on them can be programmed) per thread. So a maximum of seven events per thread can be counted simultaneously or 11 if HyperThreading is switched off. This limitation can be relaxed with so-called multiplexing [39]: measuring metrics in a round-robin way and extrapolating the results. This leads to a loss of accuracy, which is usually tolerable. There are techniques aimed at improving the accuracy, e.g. [38].

The first mention of metrics from this group in context of performance monitoring we found is for NWPerf [42]. The mentioned metrics are 'CPU Performance Counters including: percent of peak flops (for CPU 0 and 1), memory bytes per cycle (for CPU 0 and 1)'. Then we can mention TACC stats [32] which collects 'core and socket perf. counters'. Since that we may assume that every performance monitoring system can collect such data. With regard to PMCs we must pay special attention to LIKWID. Originally, LIKWID [52] was developed as a set of tools to deal with hardware performance counters on x86_64 CPUs. Later it was transformed to a full monitoring system [44].

Taking into account a relatively small number of performance metrics that can be gathered simultaneously even with multiplexing, it is important not to waste valuable resources and wisely choose the appropriate metrics to measure. Unfortunately, there is no common view of what metrics to collect. Hence, a decision should be made for every setup. For example, some ideas may be borrowed from top-down analysis methodology [54].

And the last group of performance metrics is data from other sources. It may be data from compute nodes gathered via IPMI. We place IPMI data in this group due to two reasons. First, it is usually gathered by out-of-band means, not via compute node agent. Second, these data are more relevant to health monitoring, not the performance monitoring per se. Another source for data in this group may be sources common for several components of a supercomputer system: network equipment (usually gathered via SNMP or InfiniBand-specific tools), shared storage, etc. As these metrics are influenced by several (or all) jobs simultaneously, a separate problem of correlating these data to jobs behavior arises, see, for example [34].

2.2. Data Paths in Performance Monitoring Systems

In this section we will outline data paths used in performance monitoring systems. A data path is a way that the performance data traverse in a performance monitoring system. This includes data being communicated from node agent to a server part, aggregating data, storing data for subsequent use and presenting the results to a user.

The first step in a data path in a performance monitoring system is passing data from compute nodes for subsequent processing. As we have said earlier, basically there are two modes for doing this: push, when the data are sent by compute node agents on its own, without a request, and pull, when the data are sent as a reply to a request for the data. Generally, push mode is more efficient and leads to more scalable solutions, as making a request incurs additional overhead compared to just sending the data. Due to this reason, most of the performance systems considered in section 1 use push mode. In fact, of the modern systems, only LDMS uses pull mode. LDMS pulls the data from compute nodes by means of RDMA, which allows to pull the data with very low overhead and without request-reply type interaction with the agent on compute nodes. This is achieved by the means of special interconnect hardware supporting such features like InfiniBand or Cray Gemini. LDMS can also be used in mixed modes [22]. Ganglia supports both modes. Its main mode is push, the data is sent by gmond working on compute nodes. Gmetad (server part of Ganglia) can request the data by sending a request. It is done to get recent data held in memory by gmonds and missed for some reason. When gmetad form a tree for serving large-scale clusters, the data between gmetad is transferred with pull mode: the upper-level gmetad requests the data from lower-level gmetad.

What happens after the data are collected from their sources (compute nodes and others) differs. PARMON was designed to make online analysis only. Its GUI can show only online data, no means for data storage and historical data analysis was provided. SuperMon is a set of tools for retrieving and aggregating data, no data storage is provided, and no tools for presenting collected data to users are mentioned in its description. All newer performance monitoring systems or analysis tools built on top of them provide data storage option. Gagnlia storage is based on RRDtool, which is a library for working with time-series data but cant be called a full database. Some monitoring systems use relational database management systems (RDBMS) like MySQL, PostgreSQL or SQLite. While RDMS are not very performant when dealing with time series data, which are the most part of monitoring data, use of RDBMS is justified because of their widespread and their familiarity to many developers. Other databases are used as well: TASC and C-CHAKSHU use MongoDB, LIKWID monitoring stack uses InfluxDB, Examon uses KairosDB. InfluxDB and KairosDB are specialized databases for time-series data.

Another step in processing monitoring data is calculating derived metrics. The most common type of derived metrics is one metric aggregated data. The data can be aggregated over a period of time for reducing data storage volume. Or the data from several nodes can be aggregated to get an overview of some part of the compute system. This may include a partition with a specific hardware, e.g. accelerators, or a part in a specific room, rack row, etc. An aggregation may be done over all nodes occupied by one compute job. In this case the goal of aggregation is to get the data related to one job or a set of jobs. Most common type of aggregation is the calculation of average, minimum, and maximum. Another type of metric that is derived from a single other metric is transforming counters to its derivative with time. Some metrics are counters, i.e. they only increase when some event happens. An example of such a metric is a counter of bytes received or sent via a network interface. Absolute values of such metrics are generally not interesting. What is useful for performance analysis is a time derivative of such a counter. In case of network interface bytes counter such a derivative will produce an amount of traffic passing via an interface in a time period (bytes/s).

Other derived metrics are calculated from several primary metrics. An example of such a metric is an Instructions Per Cycle (IPC) metric. It represents an average number of instructions a CPU executes in one cycle. It is quite a complex metric. To calculate it, one needs a time derivative of executed instructions counter, a time derivative of CPU cycles counters and to divide the former by the latter. In modern CPUs cycle frequency is not constant due to power saving settings, waiting for RAM access, etc., so the time derivative of CPU cycles counters is not constant. If both counters are retrieved at the same moment, calculating derivatives is not needed, one can simply divide differences of counter values. Other examples of metrics derived from several other metrics are an average network packet size, and FLOPS per Watt ratio.

Another difference between performance monitoring systems is where on the data path the derived metrics are calculated. We see three distinct places where it can be done. We also should note that not every derived metric can be calculated at any of these places.

The first possible solution is to calculate derived metrics directly at node agents immediately after the metrics have been obtained. This is feasible for derived metrics which can be calculated from primary metrics available at a single node. Such derived metrics include an average over a specific period of time or an average over all similar metrics from a node (e.g. percentage of time a core spent in user mode averaged over all cores). This approach is criticized for producing greater overhead and thus affecting compute jobs. Still this is used at least while calculating time derivatives from counters. DiMMon has modules that can calculate derivatives and averages on node agents.

The second possibility is to calculate derived metrics in a server part of a monitoring system after receiving data from node agents. This is done by the majority of monitoring systems. Note that for a big supercomputer this produces a serious requirement in compute power of the server part. As monitoring systems usually run on dedicated servers, this does not incur overhead visible to a main, computing part of a supercomputer, this requirement can be fulfilled with a required number of separate servers and generally is tolerable from a hardware requirements perspective. As we have already mentioned, sometimes sophisticated techniques are used to account for different data obtained at different timestamps [33, section 4]. Ganglia, which is built on RRDtool as data storage, uses RRDtool's built-in features to calculate derived metrics while storing data.

The last possibility is to calculate the derived metrics after the data are stored in a permanent storage. In this scenario, the data are retrieved from the database, the derived metrics are calculated, and the results are saved back to the database (or to another database). This approach is commonly used when calculating metrics related to compute jobs or long periods of time. C-CHAKSHU uses this approach for derived metrics and uses MongoDB non-relational database for scaling. From the systems described in section 1, the first paper that proposed such an approach is the description of NWPerf [42]. After that, all performance systems that produce per job data use this approach. This solution has a drawback that it incurs much load on a data storage thus requiring quite a capable storage system to be used for performance monitoring data.

MRNet [45], a tool for creating overlay networks operating in multicast/reduction manner and aimed at performing scalable calculations, is often mentioned as a possible solution to calculating derived metrics in performance monitoring systems. Still we are not aware of any real monitoring system that uses MRNet.

2.3. An Overhead Caused by Performance Monitoring Systems

In this subsection we will not directly compare the overhead of different performance monitoring systems. It is a very difficult task as no methodology exists to make such a comparison. We will rather try to describe methods which performance monitoring system authors use to measure overhead of their systems.

First, we have to admit that most of the monitoring system authors do not give any measurements regarding an overhead of a system. More common is to give general description like 'insignificant' or 'negligible' or not to say anything at all. Exceptions are the object of this description.

SuperMon authors make a theoretical study of what they call 'perturbation' from a monitoring system [50, section 4.1]. They make a conclusion that a node agent with a high peak sampling rate is preferable as it causes less perturbation when used with sampling rates much less than the peak rate. Hence peak sampling rate may be viewed as an indirect metric of monitoring system overhead. Unfortunately, SuperMon authors do not provide experimental data to support their claim.

Ganglia authors measure several value as overhead metrics [37, section 5.2]. They measure the percentage of CPU time consumed by Ganglia processes and their memory footprint.

They also provide data on network bandwidth consumed by communications between Ganglia components. They present data for different clusters and confederations of clusters.

NWPerf authors try to measure the overhead (‘perturbation effects’) directly [42, section 4]. They run a test which executes a cycle of MPI collective operations (All-to-All and AllReduce were used) and measure its execution time without NWPerf agent or with it running with different sampling rates. The conclusion made from these results is that in its normal configuration (data are obtained at 1-minute intervals) NWPerfs effects are ‘immeasurably small’.

The authors of LDMS provide extensive data on influence of LDMS on various benchmarks [24, section V]. They run more than 10 different benchmarks (artificial tests as well as real applications) and measure time changes with different LDMS modes. Overall conclusion is that LDMS obtaining data once per second does not really affect running jobs.

2.4. Comparison Summary

In Tab. 1 we provide a short comparison of different features of performance monitoring systems. The columns denote the following properties of the performance monitoring system:

- Year – when the system was introduced or when it was mentioned for the first time.
- Cluster size, nodes – a maximum number of cluster the system was tested on (per available information).
- Heterogeneous – if a system was tested on a heterogeneous cluster.
- PMC – if the system can collect Performance Monitoring Counters data.
- Data Storage – if the system has storage for performance metrics and what type of storage is used (if the information is available).
- GUI – if the system offers a Graphical User Interface.
- Per-job – can the system correlate the performance data to the jobs executed on the cluster.
- Push/pull – which mode for transmitting data does the system use.
- Reconfig on-the-fly – can the system be reconfigured without restart.
- Overhead measured – if the overhead produced by the system is measured and the results are available.

An empty cell means that no information was found. Plus and minus signs mean ‘Yes’ and ‘No’, respectively.

3. Problems with Monitoring Large Scale HPC Systems

This section describes the challenges with the current monitoring software.

3.1. Requirement of Multiple Tools

Currently, we are working with many monitoring tools like Ganglia, Nagios, XDMoD, C-CHAKSHU along with custom scripts in a single HPC facility.

Existing monitoring software can be categorized broadly into the following types.

- Tools that provide job level details through resource manager like XDMoD [15, 43], TACC-stats [31, 32], VisQueue [49].
- Tools that provide only node level stats that use third-party API/tools to get the metrics like Ganglia [37], PARMON [28], SuperMON [41, 50]. It may include proprietary tools like NVIDIA DCGM [2] for monitoring sub-system components like graphics card/storage/ High-performance networks like Infiniband.

Table 1. A comparison of performance monitoring features

System	Year	Cluster size, nodes	Heterogeneous	PMC	Data storage	GUI	Per-job	Push/pull	Reconfig on-the-fly	Overhead measured
PARMON	1998	48+	-	-	-	+	-	Pull	-	-
SuperMon	2001	128	+	-	-	-	-	Pull	-	±[1]
Ganglia	2002[2]		+		RRDTool	+	-	Mixed	-	+
NWPerf	2004	1000+	-	+	PostgreSQL	-	+	Push	-	+
Ovis-2	2008	920			MySQL	+	-	Push		-
TACC Stats & SUPReMM	2011	6400	+	+	Pickle	+	+	Pull	-	-
Dataheap	2012			+	MySQL SQLite	+	+	Push	-	-
LDMS	2014	24648	+	+	MySQL CSV SOS			Pull[3]	+	+
LIKWID Monitoring Stack	2017			+	InfluxDB	+	+	Push		
PCP	1999			+	Custom	+	+	Pull		
Examon		1022	+	+	KairosDB	+	+	Push		
DiMMon & TASC	2015	6300	+	+	PostgreSQL MongoDB	+	+	Push	+	
C-CHAKSHU	2019		+	+	MongoDB	+	-	Pull	-	-

[1] Only theoretical study given
[2] The first found release announcement date
[3] Generally, LDMS uses pull mode, but in some cases the mixed mode can also be used

- Tools like Nagios [13] that provide node component level health metrics and also provide alerting services and event-based recovery.

As per the study mentioned in section 1 and the broad category mentioned above, there is no single tool to achieve all aspects of monitoring, analysis, and alerting. Currently, we require multiple tools along with custom scripts to get comprehensive monitoring of the HPC systems.

3.2. Performance

The monitoring software should not introduce any substantial performance penalty and unpredictable noises to the HPC system. In-band communication increases overhead on compute nodes.

- A higher sampling rate for collecting metrics has a performance penalty on the overall monitoring system in addition to an effect on application execution and also poses a challenge on scalability. Many times out-of-band measurements do not enable high-frequency sampling [30] and we cannot increase the frequency of the measurement.
- Reading metrics from OS and temp files should not introduce any lock to user applications.

3.3. Large Data Storage

Storing the generated log/metrics requires a separate infrastructure. An enormous amount of data from a few hundreds to thousands nodes generating logs and various performance metrics.

If we want to capture comprehensive metrics and logs data can be of size to the tune of TBs per day. If we want to save it for off-line analysis and correlations for a few months, data storage requirements will be huge. Managing multiple databases/structures/formats as every monitoring tool requires different types. It is challenging to handle multiple databases in a single monitoring environment.

3.4. Non-dedicated Monitoring Infrastructure

Typically monitoring software deployment is an afterthought. We provision monitoring software on master or management servers already having dedicated workloads. As we evolve in our monitoring requirements, existing infrastructure becomes insufficient for dedicated monitoring systems. Traditional HPC facilities do not have dedicated hardware clusters with big data analytics capability along with deep learning for processing real-time events for monitoring infrastructure.

3.5. Data Duplication and Redundancy

Multiple monitoring software in the current HPC environment collect similar kinds of data and store them in different data formats/databases. Similarly, we collect the same kind of logs from multiple nodes/sources without any incremental information. This redundant data makes our monitoring infrastructure slow for querying, analyzing and processing. It also consumes unnecessary space in the storage.

3.6. Data Correlation

Monitoring systems retrieve data from all hardware subsystems and system software. As observed in the study presented in section 1, there is no correlation provided among different metrics or logs to capture the HPC system behavior in totality. There are also requirements for data analysis to be done to find out trends in system behavior over time.

3.7. Limited Metrics Set

Due to multiple limitations like out of band management interface [36], processor [38], or network, we are monitoring limited metrics. There is no universal set of metrics or standardization to collect specific metrics set from different subsystems of the HPC. Every HPC facility has its own custom set of metrics as per its need. A large-scale monitoring system requires new capabilities in metric-gathering.

4. Future Directions of Monitoring Systems Development

Going forward beyond the petascale systems, we will have thousands of compute nodes and many more components and subcomponents in a single HPC system. To monitor overall efficiency and get deep insight into the complete system will be a must.

4.1. Comprehensive Monitoring

Large supercomputing sites require a monitoring framework to get all kinds of monitoring information through simple API-based queries. The visualization of monitored data over a dashboard is more of a drag and drop type along with the integration of alerting mechanisms. It should be able to perform proactive actions based on monitoring events to improve system efficiency and failure. Currently, we are using multiple tools like Nagios, Ganglia, resource manager stats, and many custom scripts to monitor the HPC systems. Typically the same or similar kind of data across multiple monitoring tools are kept in different data formats/databases. The framework must remove the need for maintaining multiple copies of the same metric data and should be available to all the consumers of data through a single provider with redundancy. This has been further discussed in the data management section below.

We are monitoring many metrics in the isolation. We will see the deployment of methods for correlating different metrics in real-time to understand the holistic behavior of the system. It will also help us classify the HPC workload based on similar correlated behaviors.

4.2. Long Term Trend Analysis

As HPC systems are operational for multiple years, trend analysis over a longer period is necessary to understand behavioral changes in the overall system. They still do not give long-term statistics for drawing perceptions for policy decisions. Further, almost all of them lack the decision-making capability based on dynamic system events.

4.3. Scalable and Modular Framework

Creation of a scalable and modular framework for accommodating a large number of nodes and metrics. We keep on increasing parameters to be monitored as more sensors are available in the next-gen server and sub-systems.

- Hardware Infrastructure: dedicated cluster for monitoring infrastructure for real and offline monitoring.
- High Memory servers for In-memory real-time data analysis.
- Broker services: Message broker services for allowing multiple consumers to access the same data and avoiding data duplications.
- Ability to over-provision and failover capability.
- Lightweight collector daemons at compute side with a minimal performance penalty and use maximum out-of-band capabilities.
- Support for ingestion of data at variable time-interval with techniques to enable only meaningful and incremental information flow.
- We need to provision monitoring infrastructure before the establishment of the HPC system.

4.4. Data Management

Furthermore, data mining and reduction techniques will become a necessity in exascale to perform the on-the-fly information reduction that will be a requirement to deliver scalable, automated online performance analysis. As the system grows, it generates a large volume of data from various sub-systems. Identification of data storage and defining optimal data path in addition to data processing before storing.

4.5. Optimization

As the HPC system becomes bigger and the Monitoring system also becomes complex with multiple stages of collection, pre-processing, data segregation, aggregation, and multiple consumers of the data. We must perform an iterative way of optimization [36] of all the stages to remove any performance bottleneck in the overall monitoring systems. Understanding the overhead of each stage of the monitoring system and reduction of overhead in each stage will require optimization at each stage in an iterative fashion. Daemons or collectors installed at compute nodes should not introduce any noise or jitter for the application performance. Large data collection or spurt due to some event in the HPC system should not introduce any network overhead.

4.6. Resiliency Mitigation

Exascale systems are likely to experience even higher fault rates due to increased component count and density. Which component will fail and how it will impact the system is not known ahead of time which is important nowadays. Triggering resilience-mitigating techniques remains a challenge due to the absence of well-defined failure indicators. Examination of event logs can be part of a monitoring feature that can be used with a data mining framework for failure detection. Desh [30] framework for efficient failure prediction and enablement of proactive recovery mechanisms to increase reliability.

4.7. Visualization

Representation of meaningful data is also an important aspect to understand that gives performance statistics of HPC systems. Nowadays easily pluggable and drag and drop visualization technologies are adopted. Grafana [6] and Kibana [9] are tools that will be essential for the creation of a customized dashboard. Radar [36] kind of view will be useful to understand the complete picture of applications behavior during the execution.

Conclusion

Unlike other components of the HPC software stack monitoring is an afterthought in HPC cluster. Each HPC facility has its own list of components to be measured. Every supercomputing facility has expertise in different monitoring tools, and they use a lot of custom methods to measure cluster efficiency. There is a need for standardization and collaborative effort for monitoring tools to measure various hardware and different architectures in a similar technique. Data collection and store format should be compatible across HPC sites and future monitoring tools should be configurable on the HPC system without much effort.

Acknowledgements

The reported study was funded by the Russian Foundation for Basic Research and Department of Science and Technology, India, according to the research project No. 19-57-45020 (RFBR project number).

References

1. CLUMON. https://web.archive.org/web/20090517125016/http://clumon.ncsa.uiuc.edu/, accessed: 2021-06-16
2. Data Center GPU Manager Documentation. https://docs.nvidia.com/datacenter/dcgm/latest/index.html, accessed: 2021-07-19
3. ExaMon | Exascale Monitoring Framework for HPC. http://projects.eees.dei.unibo.it/monitoring/wordpress/, accessed: 2021-06-08
4. Ganglia Monitoring System. http://ganglia.info/, accessed: 2021-06-09
5. GitHub - ovis-hpc/sos: sos pre-release stable. https://github.com/ovis-hpc/sos, accessed: 2021-06-11
6. Grafana - The open platform for analytics and monitoring. https://grafana.com/, accessed: 2021-06-11
7. InfluxData (InfluxDB) | Time Series Database Monitoring & Analytics. https://www.influxdata.com/, accessed: 2021-03-17
8. KairosDB. https://kairosdb.github.io/, accessed: 2021-08-24
9. Kibana: Explore, Visualize, Discover Data | Elastic. https://www.elastic.co/kibana/, accessed: 2021-07-19
10. Mpp2 - Cluster Platform 6000 rx2600 Itanium2 1.5 GHz, Quadrics | TOP500. https://www.top500.org/system/173082/, accessed: 2021-05-28
11. MQTT - The Standard for IoT Messaging. https://mqtt.org/, accessed: 2021-06-17
12. MySQL. https://www.mysql.com/, accessed: 2021-06-11
13. Nagios - The Industry Standard in IT Infrastructure Monitoring. http://www.nagios.org/, accessed: 2021-06-18
14. National Supercomputing Mission. https://nsmindia.in/, accessed: 2021-07-19
15. Open XDMoD. https://open.xdmod.org/9.5/index.html, accessed: 2021-06-11
16. OVISWiki. https://ovis.ca.sandia.gov/index.php/Main_Page, accessed: 2021-06-11
17. Performance Co-Pilot. http://pcp.io/, accessed: 2021-06-11

18. PostgreSQL: The world's most advanced open source database. https://www.postgresql.org/, accessed: 2021-06-23

19. Redash helps you make sense of your data. https://redash.io/, accessed: 2021-06-17

20. RRDtool - About RRDtool. http://oss.oetiker.ch/rrdtool/, accessed: 2021-06-11

21. The most popular database for modern apps | MongoDB. https://www.mongodb.com/, accessed: 2021-06-17

22. Aaziz, O., Cook, J., Sharifi, H.: Push Me Pull You: Integrating Opposing Data Transport Modes for Efficient HPC Application Monitoring. In: 2015 IEEE International Conference on Cluster Computing. pp. 674–681. IEEE (2015). https://doi.org/10.1109/CLUSTER.2015.118

23. Adhianto, L., Banerjee, S., Fagan, M., et al.: HPCTOOLKIT: tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience 22(6), 685–701 (2010). https://doi.org/10.1002/cpe.1553

24. Agelastos, A., Allan, B., Brandt, J., et al.: The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014. pp. 154–165. IEEE (2014). https://doi.org/10.1109/SC.2014.18

25. Agrawal, K., Fahey, M.R., McLay, R., James, D.: User Environment Tracking and Problem Detection with XALT. In: 2014 First International Workshop on HPC User Support Tools. pp. 32–40. IEEE (2014). https://doi.org/10.1109/HUST.2014.6

26. Brandt, J.M., Debusschere, B.J., Gentile, A.C., et al.: Ovis-2: A robust distributed architecture for scalable RAS. In: 2008 IEEE International Symposium on Parallel and Distributed Processing. pp. 1–8. IEEE (2008). https://doi.org/10.1109/IPDPS.2008.4536549

27. Browne, J.C., DeLeon, R.L., Lu, C.D., et al.: Enabling comprehensive data-driven system management for large computational facilities. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 1–11. ACM, New York, NY, USA (2013). https://doi.org/10.1145/2503210.2503230

28. Buyya, R.: PARMON: a portable and scalable monitoring system for clusters. Software: Practice and Experience 30(7), 723–739 (2000). https://doi.org/10.1002/(SICI)1097-024X(200006)30:7<723::AID-SPE314>3.0.CO;2-5

29. Byford, N., Popov, S., Paterson, A.: Anomaly Detection in High Performance Computing Systems. In: Kos, L. (ed.) Summer of HPC 2020, pp. 12–14 (2020). https://summerofhpc.prace-ri.eu/wp-content/uploads/2020/12/SoHPC2020-reports.pdf

30. Das, A., Mueller, F., Siegel, C., Vishnu, A.: Desh: deep learning for system health prediction of lead times to failure in HPC. In: HPDC'18: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing. pp. 40–51. ACM, New York, NY, USA (2018). https://doi.org/10.1145/3208040.3208051

31. Evans, T., Barth, W., Browne, J., et al.: Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats. In: Proceedings of the First International Workshop on HPC User Support Tools, HUST '14, New Orleans, Louisiana, USA, November 16-21, 2014. pp. 13–21. IEEE (2014). https://doi.org/10.1109/HUST.2014.7

32. Hammond, J.: Tacc stats: I/O performance monitoring for the instransigent. In: Invited Keynote for the 3rd IASDS Workshop. pp. 1–29. Austin, TX (2011)

33. Kluge, M., Hackenberg, D., Nagel, W.E.: Collecting Distributed Performance Data with Dataheap: Generating and Exploiting a Holistic System View. Procedia Computer Science 9, 1969–1978 (2012). https://doi.org/10.1016/j.procs.2012.04.215

34. Kluge, M., Hartung, M.: Mapping of RAID Controller Performance Data to the Job History on Large Computing Systems. In: 2014 International Workshop on Data Intensive Scalable Computing Systems. pp. 73–80. New Orleans, Louisiana, USA (2014). http://conferences.computer.org/discs/2014/papers/7038a073.pdf

35. Lakshman, A., Malik, P.: Cassandra. ACM SIGOPS Operating Systems Review 44(2), 35–40 (2010). https://doi.org/10.1145/1773912.1773922

36. Li, J., Ali, G., Nguyen, N., et al.: MonSTer: An Out-of-the-Box Monitoring Tool for High Performance Computing Systems. In: IEEE International Conference on Cluster Computing, CLUSTER 2020. pp. 119–129. IEEE (2020). https://doi.org/10.1109/CLUSTER49012.2020.00022

37. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing 30(7), 817–840 (2004). https://doi.org/10.1016/j.parco.2004.04.001

38. Mathur, W., Cook, J.: Improved Estimation for Software Multiplexing of Performance Counters. In: 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. vol. 2005, pp. 23–34. IEEE (2005). https://doi.org/10.1109/MASCOTS.2005.34

39. May, J.: MPX: Software for multiplexing hardware performance counters in multithreaded programs. In: Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001. p. 8. IEEE (2001). https://doi.org/10.1109/IPDPS.2001.924955

40. McDonnel, K.: System Level Performance Management (1999). http://mirror.linux.org.au/pub/linux.conf.au/1999/

41. Minnich, R.G.: Supermon: High-Performance Monitoring for Linux Clusters. In: 5th Annual Linux Showcase & Conference 2001, Oakland, California, USA, November 5-10, 2001. USENIX Association, USA (2001)

42. Mooney, R., Schmidt, K., Studham, R.: NWPerf: a system wide performance monitoring tool for large Linux clusters. In: 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No. 04EX935). pp. 379–389. IEEE (2004). https://doi.org/10.1109/CLUSTR.2004.1392637

43. Palmer, J.T., Gallo, S.M., Furlani, T.R., et al.: Open XDMoD: A Tool for the Comprehensive Management of High-Performance Computing Resources. Computing in Science & Engineering 17(4), 52–62 (2015). https://doi.org/10.1109/MCSE.2015.68

44. Rohl, T., Eitzinger, J., Hager, G., Wellein, G.: LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses. In: 2017 IEEE International Conference on Cluster Computing, CLUSTER 2017. pp. 781–784. IEEE (2017). https://doi.org/10.1109/CLUSTER.2017.115

45. Roth, P., Arnold, D., Miller, B.: MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools. In: Proceedings of the ACM/IEEE SC2003 Conference on High Performance Networking and Computing. p. 21. IEEE (2003). https://doi.org/10.1109/SC.2003.10039

46. Shende, S.S., Malony, A.D.: The Tau Parallel Performance System. International Journal of High Performance Computing Applications 20(2), 287–311 (2006). https://doi.org/10.1177/1094342006064482

47. Shvets, P., Voevodin, V., Zhumatiy, S.: Primary Automatic Analysis of the Entire Flow of Supercomputer Applications. In: Proceedings of the 4th Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists. pp. 20–32. CEUR Workshop Proceedings, Yekaterinburg (2018), http://ceur-ws.org/Vol-2281/

48. Shvets, P., Voevodin, V., Zhumatiy, S.: HPC Software for Massive Analysis of the Parallel Efficiency of Applications. In: Parallel Computational Technologies. PCT 2019. Communications in Computer and Information Science, vol. 1063, pp. 3–18. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28163-2_1

49. Solis, A.J., Foss, G., Jansen, C., Stelmaszek, M.: VisQueue. In: Practice and Experience in Advanced Research Computing. pp. 293–298. ACM, New York, NY, USA (2020). https://doi.org/10.1145/3311790.3396618

50. Sottile, M., Minnich, R.: Supermon: a high-speed cluster monitoring system. In: 2002 IEEE International Conference on Cluster Computing, CLUSTER 2002. pp. 39–46. IEEE (2002). https://doi.org/10.1109/CLUSTR.2002.1137727

51. Stefanov, K., Voevodin, V., Zhumatiy, S., Voevodin, V.: Dynamically Reconfigurable Distributed Modular Monitoring System for Supercomputers (DiMMon). In: Sloot, P., Boukhanovsky, A., Athanassoulis, G., Klimentov, A. (eds.) 4th International Young Scientist Conference on Computational Science. Procedia Computer Science, vol. 66, pp. 625–634. Elsevier B.V. (2015). https://doi.org/10.1016/j.procs.2015.11.071

52. Treibig, J., Hager, G., Wellein, G.: LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In: 2010 39th International Conference on Parallel Processing Workshops. pp. 207–216. IEEE (2010). https://doi.org/10.1109/ICPPW.2010.38

53. Watson, G.R., Frings, W., Knobloch, C., et al.: Scalable Control and Monitoring of Supercomputer Applications Using an Integrated Tool Framework. In: 2011 40th International Conference on Parallel Processing Workshops. pp. 457–466. IEEE (2011). https://doi.org/10.1109/ICPPW.2011.53

54. Yasin, A.: A Top-Down method for performance analysis and counters architecture. In: 2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014. pp. 35–44. IEEE (2014). https://doi.org/10.1109/ISPASS.2014.6844459