D2.2 - Functional Specification of the Backend

Part 2: Evaluation of Performance Monitoring Frameworks

Work Package 2

December 2017

ProfiT-HPC Consortium

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

Abstract

This document will deliver a functional, yet abstract specification for the backend of the toolkit, which will be developed in the course of this project.

In the part two of the document, an evaluation of several performance monitoring frameworks is presented. These tools, in comparison to those discussed in the first part, are more suitable to be used in a system wide manner. Here, we give our assessments regarding the toolkit's use cases for the usage of Telegraf, PerSyst, TACC Stats, Collectl, and others. These tools (and their corresponding daemons) support collection of extensive performance information, and are compatible with the design of the backend of the toolkit.

Contents

1 Introduction and Rationale 3

2 Telegraf 4
2.1 Design Architecture 4
2.2 Installation 5
2.3 Usage 5
2.3.1 System Plugin 5
2.3.2 Lustre2 Plugin 5
2.3.3 Procstat Plugin 6
2.4 Interfaces and Interoperability 6
2.5 Deployment and Maintenance 6
2.6 Proof of Concept and Example Output 6
2.7 Conclusion 7

3 PerSyst 8
3.1 Design Architecture 8
3.2 Installation 9
3.3 Usage 9
3.4 Interfaces and Interoperability 10
3.5 Deployment and Maintenance 10
3.6 Proof of Concept and Example Output 10
3.7 Conclusion 11

4 TACC Stats 12
4.1 Design Architecture 12
4.2 Installation 13
4.3 Usage 13
4.4 Interfaces and Interoperability 13
4.5 Deployment and Maintenance 14
4.6 Proof of Concept and Example Output 14
4.7 Conclusion 14

5 Collectl 15
5.1 Design Architecture 15
5.2 Installation 16
5.3 Usage 16
5.4 Interfaces and Interoperability 17
5.5 Deployment and Maintenance 17
5.6 Proof of Concept and Example Output 17
5.7 Conclusion 18

6 Related Work 20
6.1 collectd 20
6.2 Ganglia 20
6.3 Diamond 20
6.4 Cray Tools 21
6.5 Further Monitoring Frameworks 21

7 Conclusion 22

Bibliography 23

Introduction and Rationale

This document presents the second part of deliverable D.2.2 – "Functional Specification of the Backend". After part one has already given an overview of metric collection tools usable for the back end of the ProfiT-HPC toolkit, this document assesses different performance metrics collector daemons and monitoring frameworks. Most of the tools evaluated in this document provide a complete framework for (online) collection and (offline) analysis of the system-wide performance metrics.

All evaluations and statements made about the different performance monitoring frameworks in this document are strongly influenced by the question of suitability for the ProfiT-HPC project and toolkit. With this in mind, a strong focus was set on the the usability for job-based performance monitoring and profiling.

In Chapter 2, the Telegraf agent (part of the *TICK* stack from InfluxData) is described. Chapter 3 describes PerSyst, the in-house developed tool for performance monitoring of the Leibniz Supercomputing Centre (LRZ) systems. Chapter 4 depicts the tool developed by the Texas Advanced Computing Center (TACC), and finally, Chapter 5 outlines the usage of the tool Collectl in accordance to the profiling toolkit. Chapter 6 briefly lists several other related works and projects that have not been considered in depth for the purpose of the developed toolkit, before Chapter 7 summarises the document and gives an overview of their usability for the ProfiT-HPC project.

Telegraf

Telegraf [1] is a plugin-based agent for collecting, aggregating, processing, and publishing metrics. Via its various plugins, a wide range of information about the system it is deployed on can be collected. The collected metrics can then be published to different databases or file formats. Telegraf is part of InfluxData's open source TICK stack [2]. In this stack, it is optimised to be used in combination with the InfluxDB time-series database [3], the Kapacitor streaming engine [4] and the Chronograf visualisation front end [5].

Telegraf is developed as an open-source project on GitHub [6] under the MIT license. It is well documented and also has clear guidelines for writing of plugins and contributing to the project.

Telegraf is continuously upgraded, with four release versions since the start of the GitHub project in 2015.

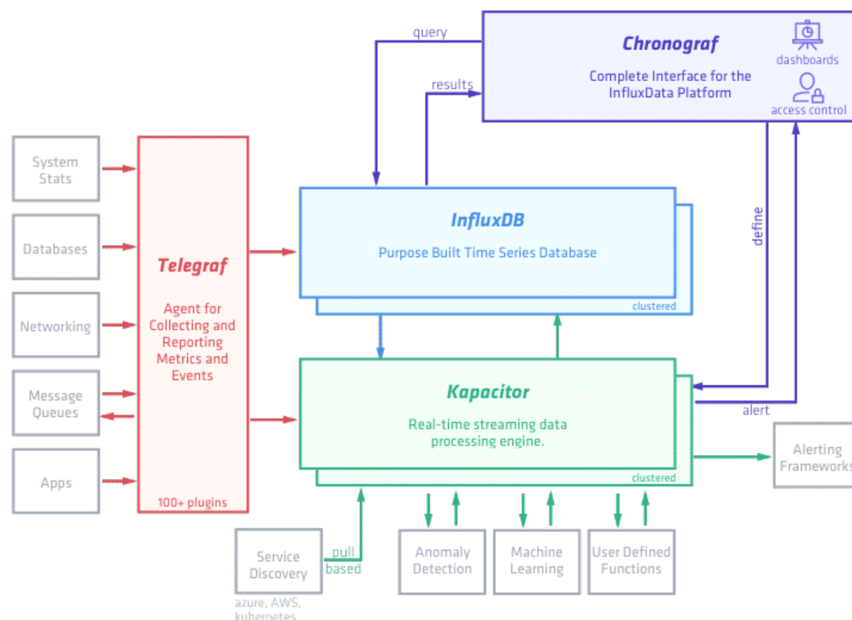

Figure 1: Overview of the TICK stack (figure from [2] by courtesy of InfluxData).

Design Architecture

Telegraf is independent, but is designed to be used within the TICK stack. For normal use cases, a time series database (in the TICK stack: InfluxDB) is needed to store the results to handle the many measurements collected by Telegraf. However, alternative products can be used for all default InfluxData TICK stack elements, e.g., Grafana [7] instead of Chronograf for visualisation, OpenTSDB [8] instead of InfluxDB for storing the time-series data and Grafana, Icinga [9] and others instead of Kapacitor for analysis or error detection.

Installation

Telegraf is provided either via a single package without further dependencies for various Linux distributions or as source code. For building Telegraf from source, a Go version of 1.8 or newer, as well as GNU make is required. The build process as taken from the documentation [10] is as follows:

- Dependencies are managed by gdm, which is installed by the Makefile if you don't have it already.
- Install Go
- Setup your `GOPATH`
- Run `go get -d github.com/influxdata/telegraf`
- Run `cd $GOPATH/src/github.com/influxdata/telegraf`
- Run `make`

Usage

Telegraf provides several plugins to collect metrics. A full list can be found on the GitHub page [6]. Here we will only present a few plugins collecting metrics that are relevant for ProfiT-HPC. Only minimal overhead could be observed for all plugins we tested so far.

System Plugin

The system plugin collects system-wide metrics from the procfs. Far more than 100 different metrics can be collected in the following categories:

- cpu
- mem
- net
- netstat
- disk
- diskio
- swap
- processes
- kernel (/proc/stat)
- kernel (/proc/vmstat)
- linux_sysctl_fs (/proc/sys/fs)

Lustre2 Plugin

The lustre2 plugin collects metrics from the procfs entries of a Lustre file system procfs. It can also collect job I/O statistics via Lustre jobstats, if they are activated in the Lustre file system.

Procstat Plugin

Another interesting plugin to use is the procstat plugin. It provides detailed information on individual processes by collecting more than 50 metrics, including I/O, CPU usage and memory information. For I/O and file descriptor related measurements, the Telegraf agent needs to be run either as root, or from the same user space as the process to be monitored.

The plugin can monitor processes of individual users, but will need to be customised to automatically choose the correct user or sub-processes of a running job.

Interfaces and Interoperability

Telegraf can provide its measurements to a text file, but is usually configured to write directly to one of many possible consumers such as InfluxDB [3], OpenTSDB [8], Elasticsearch [11], AWS CloudWatch [12] and many others. An up-to-date list can be found in the Telegraf documentation [13].

As graphical output, the collected metrics can be visualised by using e.g. Grafana [7], Chronograf [5] or other tools processing the previously published data.

There is currently no integration into a job scheduler or batch system. Job information can, however, be easily added via custom processor plugins. In general, Telegraf is very easily extensible with additional plugins, by editing existing plugins or by running a custom binary with the exec plugin.

Deployment and Maintenance

Telegraf can be installed from a single package. The official Telegraf version is available prepackaged for most Linux distributions. Configuration of Telegraf is realised via a single configuration file. For better readability, multiple configuration files may be used. Upgrades can be performed by upgrading to a newer Telegraf package. For a customised Telegraf, packages need to be built from source for each update, to include the custom changes.

The Telegraf agent is normally started by the init system during the boot process and does not interfere with user processes or batch job output. The users do not need to adjust their jobs or executables in any way to enable metric collection. Upon failure, Telegraf is automatically restarted. Depending on the duration of the failure, no metrics might be collected, but the system should otherwise not be affected. In our tests, we could not detect any failures of the agent.

Proof of Concept and Example Output

To test Telegraf and the capabilities of the TICK stack a proof of concept installation was made. The procstat plugin as a promising candidate to be used within the ProfiT-HPC toolkit was chosen to be activated.

Telegraf and the shipped procstat plugin was configured in the following way via the `telegraf.conf` file to monitor processes of a specific user ID and to add a job ID tag to its measurements - similar to a possible scenario within a ProfiT-HPC toolkit to monitor processes of user batch jobs:

```
[global_tags]
    jobid = "6666.testjobid"
[[inputs.procstat]]
```

```
user = "bzadmgla"
```

To be able to collect I/O related metrics Telegraf was additionally run as *root*.

The proof of concept worked as expected (all processes and their resource usage running under the configured userid were recorded and the measurements were tagged by the configured `jobid` value), the setup was impressively easy and done within minutes.

An example Chronograf dashboard showing visualisations of the collected metrics is shown in Figure 2.

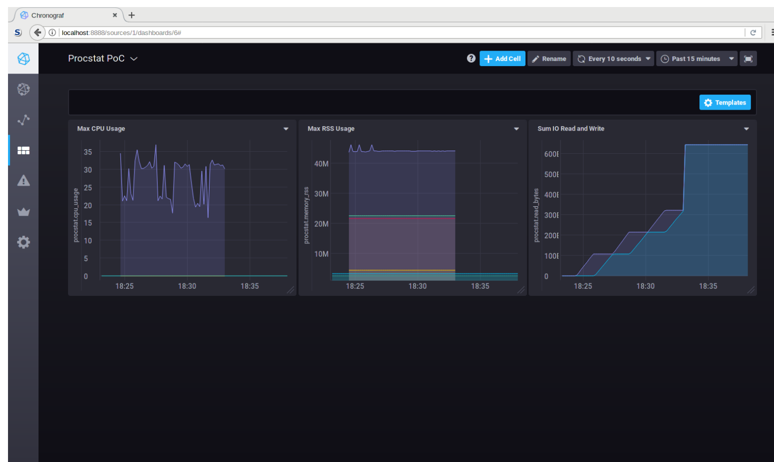

Figure 2: Visualisation of metrics collected by Telegraf's procstat plugin: Screen shot of a Chronograf dashboard (from left to right: maximum CPU usage in percent, maximum memory usage and the sum of I/O reads and writes of the monitored process).

Conclusion

Telegraf is a highly modular, plugin-driven metric collection agent, that can be used with a variety of output options, depending on the specific needs. This makes it very easy to deploy it in various scenarios, e.g. monitoring system usage and status, sensor data and more.

For the ProfiT-HPC project it provides a good alternative to implementing a completely new metric collection approach. It ships with plugins for many metrics that are of interest and also provides a framework with clearly defined configuration management, separate collection, aggregation and processing pathways via individual plugins. For missing metrics, plugins can be added either directly in Go by following the guidelines for new plugins in the documentation or by use of the exec plugin, where a custom binary or script can be called to provide further measurements. Due to Telegraf's many output options, it could also be integrated into existing infrastructure with only some small tweaks.

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

PerSyst

The PerSyst monitoring tool [14], [15] was originally developed (and in production) on the SGI Altix 4700 as part of the Höchstleistungsrechner Bayern II (HLRB-II) at the LRZ. It was further developed and ported to the current LRZ supercomputer “SuperMUC”, mainly by Dr. Carla Guillen, Dr. Wolfram Hesse and Dr. Matthias Brehm. The development of PerSyst was done only at the LRZ and is not publicly available, as for example the TICK-Stack (see Chapter 2). The implementation language of this non-commercial product is done in C++ and a documentation of the important concepts of PerSyst can be found in [15], the PhD thesis of Dr. Carla Guillen. Furthermore there is an online user documentation about how to use the PerSyst report via the web interface (dynamic report) or the console (static report). This documentation can be found under [14]. There is no further public documentation available.

Design Architecture

The PerSyst tool is an independent and non-modular monitoring tool, which is not part of a software stack (although this was planned as part of the FEPA project) and uses no source code instrumentation. Nevertheless, it depends on the following tools collecting the metrics on the nodes:

- Likwid to collect hardware events,
- `mmpmon` to collect IO GPFS events,
- `perfquery` to collect InfiniBand events,
- `/proc/meminfo` to collect information about the memory usage, and
- SAR (system activity report events).

Furthermore, PerSyst is based on the following distributed three component hierarchical agent architecture (see figure 3):

- Sync agent (to synchronize the measurement),
- Collector agent (collects performance data) and
- PerSyst agent (performs the measurement).

In the case of SuperMUC one Sync agent exists at the top of the agent hierarchy as the front end to the system, which triggers and synchronises the measurements on the nodes. In the middle layers, twelve additional Sync agents and 216 Collector agents are implemented. On the lowest hierarchy level, a PerSyst agent listens to the tools and collects the measured data on every of the 9216 nodes of the SuperMUC.

The total time of a measurement cycle lies at 10 minutes, i.e., 600 seconds, in which the measuring and analysis phase take 120 seconds and the idle phase takes 480 seconds (see Ref. [15], Section 4.4). In the measuring and analysis phase the initialisation and restart of the tools which have crashed are included. Furthermore, the possible repetition of measurements and re-sending of the data is also part of this phase. The actual measurement time amounts to only 10 seconds. In the idle phase the middle and front end layers aggregate and collect tasks and store them.

At LRZ it was tested whether the total time of the measurement cycle could be reduced to seconds instead of minutes. The quality of the data grew, but the monitoring overhead also grew

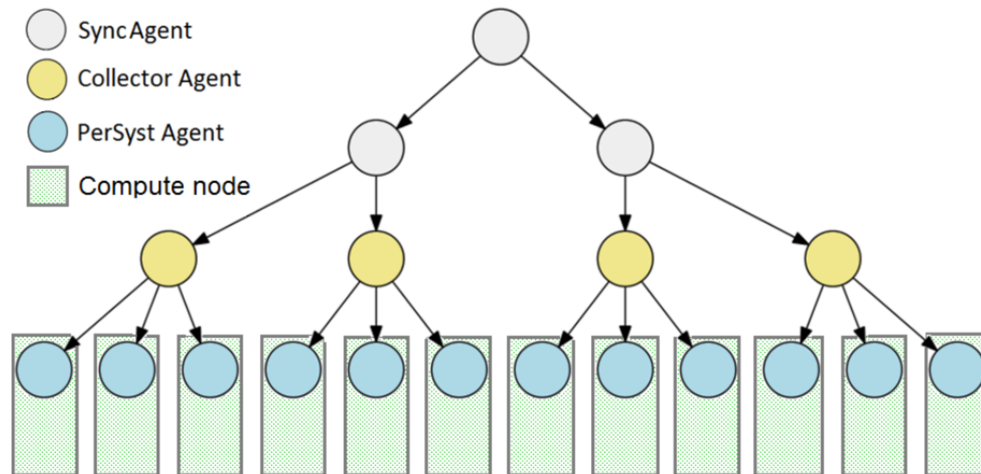

Figure 3: Hierarchy of the agents (Source: [15], p. 69, with friendly permission of C. Guillen).

immoderately. Since a ten second measurement every 10 minutes delivers a sufficient amount of good-quality measurements, fulfilling all needs at the LRZ, this interval length was chosen.

Installation

First of all, no local installation of PerSyst was done, because of the non trivial porting steps from the SuperMUC to a Tier 3 cluster (see Section 3.5). In addition, PerSyst was not developed as a portable framework but rather for the local installation and monitoring of LRZ systems. Thus, the installation description that can be given here is very limited.

When installing PerSyst, the metric collecting tools `procfs`, Likwid and SAR (described in [16], Sections 2, 5 and 7) have to be installed as well as `mmpmon` and `perfquery`, which are shipped with the parallel file system GPFS, respectively with the Infiniband installation.

Usage

The above mentioned metric collection tools collect the following metrics every ten minutes:

- FLOPS, vectorised FLOPS,
- single precision to double precision ratio,
- count of instructions, CPI,
- core frequency,
- L3 misses to Instructions Ratio, L3 bandwidth, L3 hits to misses,
- Load to misses ratio, Load to store ratio,
- user / system CPU load, IO wait (collected with SAR and all in percent),
- Transmitted and received Bytes.

In addition to these, further or other metrics can be configured for collection as well. Since the PerSyst tool was not installed on any ProfiT-HPC cluster, a statement in terms of measurement

overhead cannot be given. According to [15], section 4.4, the communication and computational overhead of PerSyst is negligible in the case of a measurement cycle of ten minutes.

Interfaces and Interoperability

As described before, the PerSyst agents collect the values which are measured by the metric collecting tools and there is no plugin concept to add further metric collection tools at the time of writing. The collected data is then stored in a database, from where two different outputs are generated: a dynamic and a static one. In the dynamic version, a web API is used to visualise the data in a timeline view and a data/performance view. The timeline view shows the severity distribution over a selected domain (e.g. CPUs or nodes) while the data/performance view shows comparison graphs between properties. A static view of the data set can be generated via the command line, which can then be visualised with a common web browser (see [14]). An examplary output is shown in Section 3.6.

Deployment and Maintenance

Although PerSyst has some very interesting features (for example the way of performance analysis), PerSyst was not installed on one of the project clusters yet, because of some shortcomings for our purposes:

- The total measurement time of ten minutes is too large for the purposes defined in the project, since a fine grained measurement to identify e.g. short MPI bursts is envisaged. It is possible, that this time interval length could be reduced on Tier 3 systems to one minute thirty seconds, but this time interval is also too long for the projects needs.
- The second reason was the fact, that the porting to another cluster is not straight forward, because several non trivial adjustments have to be done.

Proof of Concept and Example Output

Because of the above mentioned problems, the PerSyst tool was not installed yet and a Proof of Concept was not carried out. Nevertheless we will present an example output of a user report by PerSyst, which can be found online (see [14]).

The user gets from the PerSyst tool an aggregated view of the performance information of its *own* running or executed jobs in the following two ways of output:

- dynamic report via the web API,
- static report via a command line tool.

For brevity in this section only the dynamic report and the timeline mode to visualise the performance data is shown. A screen shot of the timeline view is shown in figure 4. In the upper area of the view the user can find the *Job View table*, where specific performance information about its jobs and the values of the specific metrics are listed in a table (e.g. start and end time of the job, number of cores and nodes used and the energy consumption). In the *Average View* below, the values of performace counter and metrics of the jobs are listed and rated, so that the user can see, if the averaged metric value for this job is good or not so good or bad, which is visualized by colours (green = good until violet = very problematic). This rating is based on the built in PerSyst performance analysis, which collects the raw data and evaluates the derived metrics on the basis of properties and strategies / strategy maps. The result is a severeness

Figure 4: Dynamic Report of PerSyst (figure was taken from [14]).

indicator, which is a normalized value from zero (no problems) until one (severe problematic value of the metric) and of which the coulouring is derived from. Especially this performance analysis feature, which is based on the ideas of `Periscope` (see [17]), could be helpful for the implementation of the report generator in the ProfiT-HPC project.

Conclusion

PerSyst is a performance monitoring tool with many interesting features, e.g.:

- the kind of performance analysis, which rates the derived metrics,
- the intelligent reduction of the network traffic on the base of the performance analysis,
- the proven stability of the tool and the long experiences which were made with it,
- extensible to further metrics.

In contrast there are some severe disadvantages for the project needs in comparison to other tools:

- relatively long total time interval length,
- non trivial porting procedure.

Because of the above described problems, this tool will not be considered for the usage in the ProfiT-HPC toolkit for now. Nonetheless, different aspects, i.e., the different output possibilities and the aggregation of metrics might influence the design of the ProfiT-HPC toolkit.

TACC Stats

TACC Stats [18, 19] is a framework for continuous and low-overhead collection of system-wide performance data. It is developed by the Texas Advanced Computing Center and has been collecting job-level performance data since September 2013 on two of the largest TACC systems, Lonestar and Stampede, as well being deployed on various chip architectures on multiple HPC systems around the world, including all of the major HPC systems of the NSF XSEDE consortium [20]. TACC Stats also provides a web-based interface for analyzing jobs and system-level reports.

The package is split into two sub-packages, `monitor` and `tacc_stats`, with the former unifying and extending measurements taken by Linux monitoring utilities such as Systat (SAR) or iostat, and the latter performing the data collection and analysis in the production environment. The source code of the package is hosted on Github [21], but at the time of writing its community is limited and the documentation for the project is seemingly narrow.

TACC Stats is licensed under the GNU Lesser General Public License Version 2.1. The latest version (as of October 2017) is 2.3.1, released in April 2017. This version adds the support for Intel Knights Landing, Intel Omnipath, and NVIDIA GPU architectures.

TACC Stats is funded by the National Science Foundation and is related to an ongoing HPC systems analytics project, SUPReMM [22]. SUPReMM is a comprehensive open-source tool chain that provides resource monitoring capabilities to users and managers of HPC systems. The SUPReMM processing tools package is under active development and is released under the GNU Lesser General Public License Version 3.0. Contributions are accepted via standard Github pull requests [23].

The SUPReMM architecture has three major components:

- a software that runs directly on the HPC compute nodes and periodically collects performance information,
- a software that uses the node-level performance data to generate job-level data,
- an Open XDMoD module that enables the job-level information to be viewed and analysed.

Open XDMoD (eXtrem Digital Metrics on Demand framework) [24] was originally targeted at managers and providers of computational resources of the XSEDE organization [25, 26], but is now available as an open source product. Specifically, it is a tool to facilitate the management of HPC resources and incorporates TACC Stats as a principal source of data. So far, it has had more than 200 downloads in 2017 and 74 confirmed installations/upgrades in 2017 [27]. Open XDMoD is released under the GNU Lesser General Public License Version 3.0. Support for the tool is available by email and contributions can only be made after contacting the developers.

Design Architecture

TACC Stats is split into two sub-packages. The `monitor` sub-package is written in C and performs the online data collection and transmission to a RabbitMQ server over the administrative Ethernet network[1]. It is possible to not use RabbitMQ, which will result in a legacy build that relies on the shared file system to transmit data, but this is not recommended by the developers [21]. In this mode, cron jobs are used to store the collected metrics on the compute nodes, in addition to daily rsync commands to agglomerate the data.

[1] As a side note, InfluxDB supports reading data from RabbitMQ via an input plugin [28], making it theoretically possible to import the data independently to a time-series database.

The `monitor` sub-package includes a System V daemon, `tacc_statsd`, which collects CPU usage, socket level memory usage, swapping and paging statistics, system load and process statistics, system and block device counters, inter-process communication, file systems usage (NFS, Lustre, Panasas), interconnect fabric traffic, and CPU counters and Uncore counters (e.g. counters from the Memory Controller, Cache and NUMA Coherence Agents, Power Control Unit) [19]. After collecting the data, it is possible to find out the under-performing or misconfigured jobs. Jobs are flagged when they leave nodes idle, use the wrong network, experience a drastic drop in performance, or show evidence of low efficiency. The associated web interface allows for browsing all jobs associated with a cluster, identifying flagged jobs, and plotting basic job characteristics.

The `tacc_stats` module is a pure Python package. It installs and sets up a Django-based web application (running for example on an Apache web server) along with tools for extracting the data from the RabbitMQ server in order to feed them into a PostgreSQL database.

Further details regarding design architectures of TACC Stats can be found in Ref. [20].

Installation

The installation steps are described on the Github page of the project [21]. Although TACC Stats is seemingly intended for open-source software based HPC clusters, our installation attempts of the last two releases in an Ubuntu 16.04 environment were unsuccessful. Several dependencies had to be addressed manually, particularly the XALT package [29, 30] (corresponding with a MySQL server host). The necessary installation steps are not entirely listed for a fresh install (on an Ubuntu Xenial in our case).

Usage

According to the developers, the daemon has an overhead of 3-9% on a single core when configured to sample at a frequency of 1 Hz [21]. The typical configuration is to sample at 10 minutes interval, with additional samples taken at the start and end of every job. The overhead for this case is estimated to be 0.02%. An example snapshot of the web interface is illustrated in Refs. [19, 20]. For the SUPReMM module, the recommended collection period is to sample every 30 seconds.

A Python script in the `tacc_stats` module is present to process the node-level data into job-level data. The module (`job_pickles.py`) stores each job's data in a Python pickle[2] file. Running `job_pickles.py` without arguments for example every 24 hours as a cron job can be used to pickle and store all jobs from the previous day to a predefined directory.

Interfaces and Interoperability

TACC Stats does not have a flexible modular design, and its extensibility is limited to directly modifying the code base. The daemons running on the compute nodes send the data directly to the RabbitMQ server (and a prespecified queue). The accompanying web portal is a straightforward (and currently only) way to explore the historical data. System administrators or consultants can search for a sublist of jobs that have metric values exceeding chosen thresholds. The web portal searches require no knowledge of SQL or the underlying database structure, making generation of more complex or customized queries possible [20].

When using TACC Stats in conjunction with a batch system and in order to correctly label job IDs, it is required to modify the prologue and epilogue scripts of the job scheduler. At

[2] "Pickling" is a builtin Python language process in order to convert objects or lists into a character stream. The serialization (deserialization) enables Python objects to be saved to (restored from) disk.

the beginning and end of every job, TACC Stats should be made aware of the job ID. The aforementioned pickling works natively for SGE and Slrum job schedulers.

The performance data can be accessed via the web portal which is relying on a PostgreSQL database server. The job data can be browsed by date or searched through a metadata query. The current public implementation (i.e., the Github repository [21]) is targeted only to administrators and consultants, and not directly to the users.

Deployment and Maintenance

After compiling the daemon from source, it is possible to build an RPM package with specifying the RabbitMQ server (and the queue name) to deploy on the compute nodes. Moreover, the developers have addressed the case of a shared usage of nodes among multiple users [20]. While being impossible to definitively attribute all the collected performance data to specific jobs on shared nodes, a scheme to disentangle data has been suggested using `LD_PRELOAD` environmental variable for signalling the daemon. However, this method is prune to a race condition and has to be done synchronously. Besides, collecting Lustre and MPI performance in a shared environment is not possible without code instrumentation, which is beyond the scope of the tool.

Proof of Concept and Example Output

It appears that so far, TACC Stats has been developed to be used mostly in particular to TACC HPC systems. As mentioned before, many of the required setup steps has to be done manually, and a general automatic installation procedure seems to be missing (since the dependencies have been most probably already existing on TACC systems). This point, besides the ties to the XALT project restrained us to implement an in depth proof of concept in a time frame acceptable for our use cases.

Snapshots of the web interface as well as details of the performance collected can be found in Refs. [20, 31].

Conclusion

TACC Stats design goals are similar to the ProfiT-HPC toolkit. However, the current implementation is tailored to the needs of TACC, and the target audience has not been initially the users of the system. The web interface and the analysis is more suited to system administrators and consultants, or at least advanced users. Additionally, the integration of XALT with TACC Stats, although insightful in theory, would impede the applicability of the tool to Tier-2 and 3 systems in Germany.

The future roadmap of TACC Stats is towards implementing a time-series database in order to aggregate and correlate the job data more efficiently, as well as enabling automatic real-time analysis of for instance problematic jobs [20].

Collectl

Collectl [32] is a CLI based tool written in Perl with a monolithic structure. It is designed as a comprehensive and fine-grained performance monitoring tool for Linux, originating from the HPC world. Collectl enables real-time viewing of the performance data directly in the console, and monitors a broad set of subsystems. Its primary source of information is the proc filesystem.

The latest version of Collectl, 4.3.0, was released on October 3rd, 2017 under the GNU General Public License Version 2 licence. The source code is hosted on SourceForge [33] and a comprehensive documentation for the tool is available from the website or the man page. Support for the tool is available either through the corresponding forums and mailing lists or contacting the developer directly. A Lustre data collector for Collectl is developed independently and can be found in [34]. The plugins supports Lustre 2.x systems (and is tested for versions up to 2.5.x).

While Collectl is focused on efficient performance data collection, a web-based tool is also available (named Colplot) from the same developer that uses Gnuplot for generating plots from Collectl generated files. The latest version of this tool is 5.2.0 and is released on January 23, 2017. It can be downloaded from the same SourceForge repository as Collectl.

Design Architecture

Collectl can be operated in different modes, as listed below.

- **Interactive Mode**
  This is the default mode with data read from proc file system. In this mode simply the change between the current and previous values are recorded.
- **Record Mode**
  The data is analysed after being read similar to the interactive mode, and is written to a file. This is the most efficient path of operation, and can be used for example in per job basis situation.
- **Playback Mode**
  This mode is nearly identical to the interactive mode except that the data is read from a file.
- **Plot Format**
  Selecting the `-P` flag leads to creating output in a format suitable to be plotted.
- **Socket**
  The flag `-A` followed by an address and an optional port number (2655 being the default port) sends data directly to a socket to the address specified.
- **Custom Output**
  In this mode, Collectl can bypass its standard routines for formatting interactive data and call a custom formatting routine instead. This can be used for example to send the data directly to Graphite. Further help on this is provided using `collectl --export graphite,h`.

A crucial difference between Collectl and similar tools such as SAR is the ability of synchronised (among different systems) and sub-second sampling ability of the former. Also, SAR does not provide NFS, Lustre, or interconnect statistics and its output cannot be directly fed into a plotting tool. Moreover, SAR is limited to collecting data from one thread, as well as being limited to 256 processes in interactive mode. Furthermore, SAR cannot select processes to monitor other than all or by PID (hence cannot selectively discover new processes). All these limitations are

lifted using Collectl as the performance data collection tool. Furthermore, Collectl can collect data locally at one rate (finer-grained) and export at a different rate.

A separate utility called Colmux acts a multiplexer for the Collectl data streams from multiple streams [35]. It has been tested on clusters of over 1000 nodes, although the server running the tool then should have the capacity for a relatively higher load. This helper utility can run in two distinct modes of Real-Time and Playback. In the real-time mode, Colmux communicates with remote instances of Collectl, which are collecting real-time performance metrics. In the playback mode, Colmux communicates with remote copies of Collectl playing back historical data files.

Installation

Collectl is a Perl script, hence very easy to install, deploy, and use. No root privilege is required for running or installing the tool. An accompanying script named `INSTALL` installs the tool into the same location as the RPM package. It copies the script by default to `/usr/bin/collectl` and all the other runtime components into `/usr/share/collectl`.

Usage

Using the `-top` flag, the output of Collectl will be similar to the Linux `top` command. Collectl can run interactively, as a daemon, or both. Overhead measured to be <0.1% when run as a daemon using the default sampling interval of 60 seconds for process and slab data and 10 seconds for everything else. According to the web page of the tool, the overhead of the collection of 8640 samples with 10 seconds samples in a day is about 0.01% for CPU, disk, network, and memory data. Obtaining the same data per process nearly doubles the overhead [36]. Users can simply measure the overhead of Collectl by using a builtin mechanism, using `time collectl -scdnm -i0 -c8640 -f /tmp` for the mentioned sampling experiment.

Running the `collectl` command without any arguments would show data with the following header, followed by the corresponding measured values:

```
$ collectl
waiting for 1 second sample...
#<--------CPU--------><----------Disks-----------><----------Network---------->
#cpu sys inter  ctxsw KBRead  Reads KBWrit Writes   KBIn  PktIn  KBOut  PktOut
```

For obtaining all available metrics, flag `--all` has to be specified. The following lists the header in this case:

```
$ collectl --all
waiting for 1 second sample...
#
#<--------CPU--------><--Int---><-----------------Memory----------------->
    ↪ <----------Disks-----------><----------Network----------><-------TCP
    ↪ --------><------Sockets-----><----Files---><------NFS Totals------>
#cpu sys inter ctxsw Cpu0 Cpu1 Free Buff Cach Inac Slab Map Fragments KBRead Reads
    ↪  KBWrit Writes KBIn PktIn KBOut PktOut IP Tcp Udp Icmp Tcp Udp Raw Frag
    ↪ Handle Inodes Reads Writes Meta Comm
```

Collectl output can also be saved in a rolling set of logs for later playback, or displayed interactively. There are also plugins that allow for reporting data in alternating formats or sending them over a socket to remote tools, such as Ganglia or Graphite. The output files can also be created in a space-separated format for plotting with external packages. Among available utilities, the `colplot` script provides a web-based interface to plots generating using Gnuplot.

Interfaces and Interoperability

The output of Collectl is text, and it can generate human readable tables or plottable data. Because of that, it is easily extensible with further utilities to pre- or post-process the data. There are currently 5 different custom exports available that are part of a standard Collectl distribution:

- `gexpr` for sending Collectl data to Ganglia,
- `graphite` for sending data to Graphite,
- `lexpr` generates output in an easy to parse format,
- `proctree` provides an alternative representation of process data,
- `vmstat` outputs data in the same format as the `vmstat` command.

It is possible to build custom modules for importing or exporting the data. The API is callable using Perl scripts, similar to the export modules mentioned above.

Users can select the type of data Collectl reports as either summary or detailed. Furthermore, summary data can be generated in either brief or verbose formats, with the former being the default, and the latter containing the most complete data.

Deployment and Maintenance

As previously mentioned, it is effortless to install Collectl on the target systems. The `apt` or `yum` package managers can also be used to install the tool from the standard repositories.

Collectl can run as a daemon in the background, and it supports a quite mode via the `--quiet` flag. Installing Collectl using the package manager configures the tool to run as a service, and is disabled by default from starting automatically. Per default configuration, Collectl writes its date and time named log files to `/var/log/collectl` (as well as a message log), rolling them every day just after midnight and retaining a week of log files. The file `/etc/collectl.conf` contains various configurations for running in the daemon mode. This usage mode plus the Colmux utility serves as a way for automatic collection of system-wide performance data, in connection with the plotting tool to (automatically) generate performance plots.

Proof of Concept and Example Output

After installing Collectl, it is helpful to see what subsystems are present for generating the performance data. Examplarily, the output of `collectl --showsubsys` is shown below.

```
$ collectl --showsubsys
The following subsystems can be specified in any combinations with -s or
--subsys in both record and playbackmode.  [default=bcdfijmnstx]

These generate summary, which is the total of ALL data for a particular type
  b - buddy info (memory fragmentation)
  c - cpu
  d - disk
  f - nfs
  i - inodes
  j - interrupts by CPU
  m - memory
  n - network
  s - sockets
  t - tcp
  x - interconnect (currently supported: OFED/Infiniband)
```

```
  y - slabs

These generate detail data, typically but not limited to the device level

  C -  individual CPUs, including interrupts if -sj or -sJ
  D -  individual Disks
  E -  environmental (fan, power, temp) [requires ipmitool]
  F -  nfs data
  J -  interrupts by CPU by interrupt number
  M -  memory numa/node
  N -  individual Networks
  T -  tcp details (lots of data!)
  X -  interconnect ports/rails (Infiniband/Quadrics)
  Y -  slabs/slubs
  Z -  processes
```

The output format suitable to be plotted with Colplot can be selected by the `-P` flag. For example, the command `collectl -s cdmCDFMNZ -P -c1000 -f.` (which can be run under normal user privileges) creates a compressed file in the current directory, containing 1000 samples each one second apart from the selected subsystems (i.e., summary for CPU, disk, and memory, as well as details for individual CPUs, disks, NFS, memory, network, and processes). Some of the generated plots using the above command is shown in Figs. 5 and 6. If the daemon mode is chosen instead, the corresponding settings can be configured in the `/etc/collectl.conf`. Consequently, in the Colplot web interface, a time frame can be selected in order to generate the desired plots. The generated plots can be saved as images in PNG format, or exported to PDF files.

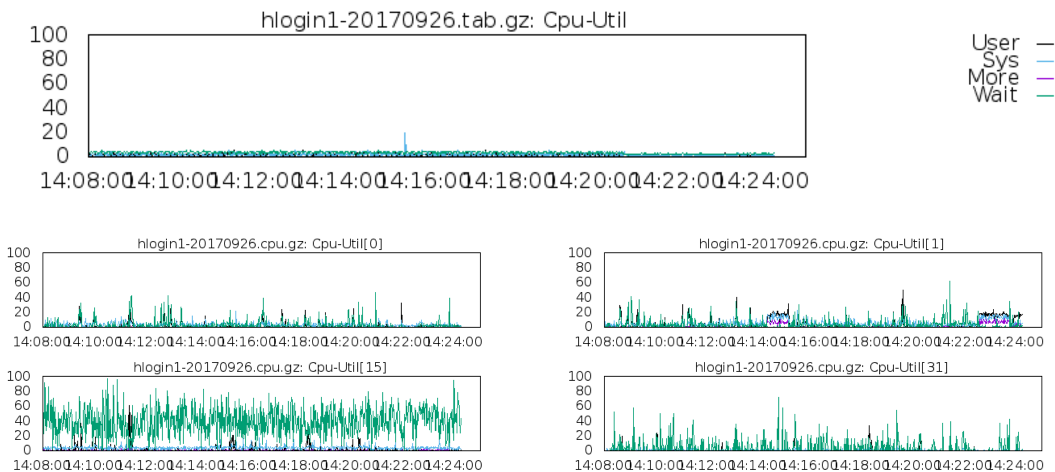

Figure 5: CPU utilization of the HLRN-III login server over 1000 seconds generated with Colplot using the data collected from Collectl. Top: total CPU utilization. Bottom: single CPU utilizations.

Conclusion

Collectl proves to be a powerful and comprehensive tool to collect performance data, in particular for HPC environments. Its flexibility and ease of use make it an attractive choice to be used as the backend of the toolkit. Unfortunately, there are some points that refrain us from really using it. Compared to, e.g., TICK Stack, the number of already available plugins is rather small. Potentially, too many plugins will have to be developed by ourselves. In addition to this, there seems to be only one developer behind Collectl, leaving it with an uncertain future.

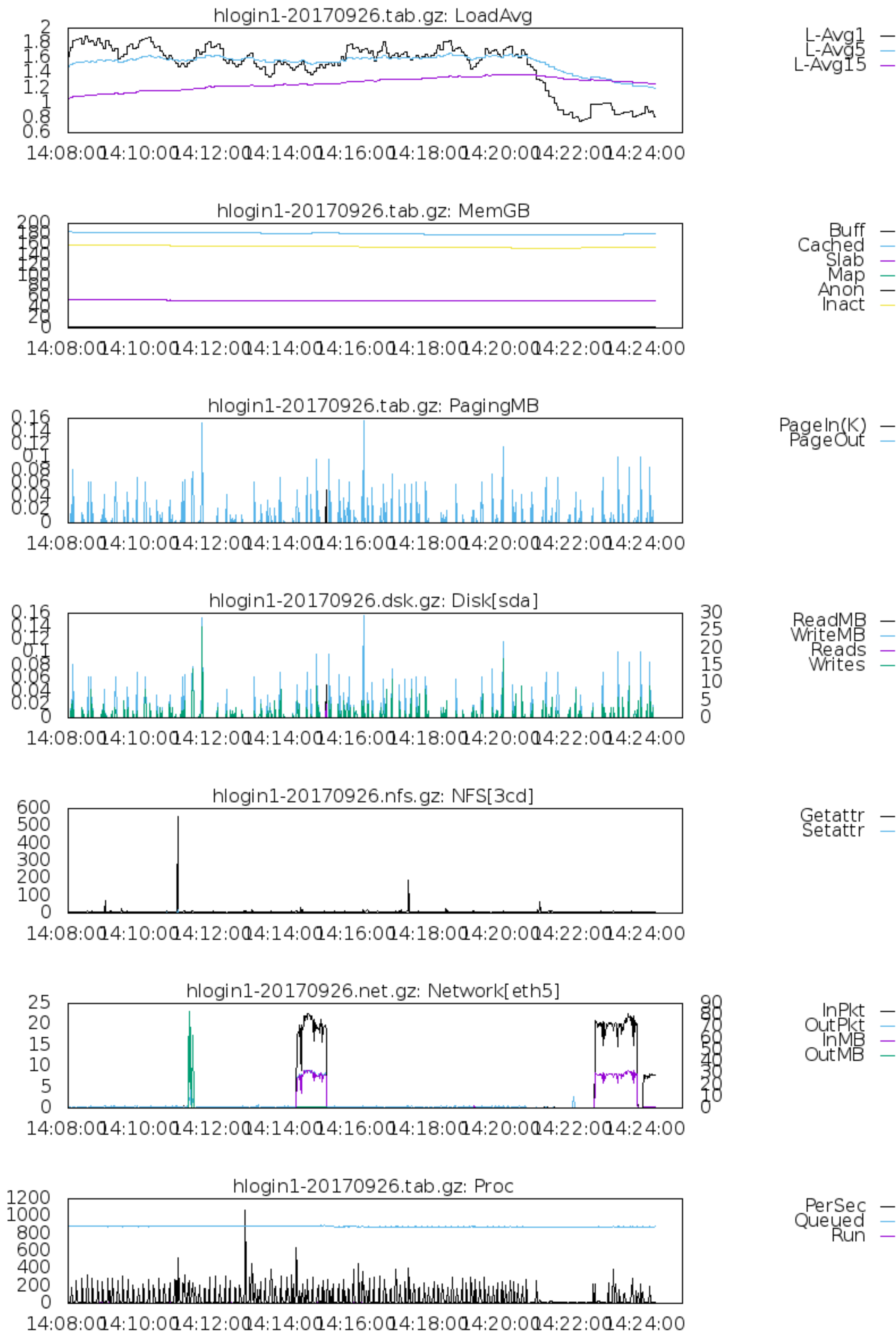

Figure 6: Various performance data of the HLRN-III login server over 1000 seconds generated with Colplot using the data collected from Collectl. From top to bottom: load averages, memory, paging, disk I/O, NFS, network I/O, processes.

Related Work

This section describes further tools and frameworks that could theoretically be considered for usage in the ProfiT-HPC toolkit. Some of the following tools are very similar to those previously described in this document and only small nuances have driven the decision to the one or the other tool. Other tools listed here are widely used but their usage not feasible in the ProfiT-HPC toolkit. A short description and explanation of why these frameworks or tools have not been chosen follows in the next subsections.

collectd

The system statistics collection daemon collectd [37] is widely used and has been available for more than twelve years now. While it has been implemented by only one person in the beginning, it has been developed by more than 300 people according to the GitHub project statistics [38] by the time of writing. The daemon is written in C, to ensure high performance and light-weight collection of data. According to their own description, collectd is *not* a monitoring tool, but this seems to be mainly in relation to notifications and the detection of threshold exceedance. Nonetheless, it can collect metrics in 10 second intervals. Different plugins have been developed to add further metrics to the collection or to add consumers like InfluxDB.

One major drawback in collectd is the lack of tags. While there is the possibility to add global tags, the addition of tags on a per metric basis needs some additional effort. This is already included in other frameworks and tools we have assessed. In addition, collectd can *only* be started as a daemon, whereas other frameworks often give the opportunity to either start a daemon, which runs permanently, or to start the agent on a per job basis.

Ganglia

Ganglia [39] is a scalable monitoring tool, designed for high-performance computer systems and clusters. It was once developed by Berkeley for the monitoring of their own systems but is now widely used - not least due to the bundling in Red Hat Enterprise Level Linux distributions. Also Ganglia operates with daemons, one of which (`gmond`) is a multi-threaded daemon running on every compute node. In addition, a meta daemon `gmetad` runs on each tree node of the hierarchical architecture design. In general, the metrics collected by Ganglia are focused on the general system state and cannot be broken down to a per process, per user or per job level. This does not pose a problem on system which have non-shared node usage, but does indeed pose a problem when trying to monitor single jobs on shared nodes.

Diamond

The daemon-based, metric collection framework Diamond is written in Python, easily extensible through a given API and publishes the collected metrics to diverse consumers. Initially, Diamond was developed to publish the collected metrics to Graphite, but now also supports other consumers like RRDs and time-series databases. It is an actively developed open-source project with bursts of activity from very different contributors. This is also one major drawback of Diamond - many of the additional collectors and handlers have not been touched for two years, leaving open questions about the maintenance of the user additions.

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

Cray Tools

The Cray Resource Utilization Reporting (RUR) tool is developed for administrators to gather statistics on application resource usage. It is easily extensible through plugins and a well-established tool on Cray systems. But since RUR is only available on Cray systems, this tool is not of interest for our toolkit.

Further Monitoring Frameworks

There are many system monitoring frameworks like Snap Telemetry, netdata, Nagios Core or Performance Co-Pilot (PCP). All are implemented in a scalable fashion and intended for usage on large, distributed systems. Most are extensible through either plugins or add-ons and only some require changes in the main code. All of the mentioned frameworks are designed for system-based monitoring and thus require additional development work from the project partners. Considering the goals of out project, there is no definite advantage of one of these tools over the other, thus the decision was based on the ease of installation, quality of documentation and flexibility. In addition, it was taken into account how much additional programming overhead the use of a specific tool or framework would impose on the ProfiT-HPC consortium.

Conclusion

This document has given an overview of monitoring frameworks considered for the first stage of the ProfiT-HPC toolkit. This overview is in no means complete, as there are far more frameworks available and continuously new ones are emerging. Many of these frameworks follow the same intent and have the same goals, leaving only small nuances to make a decision for one or against another for our aspired toolkit. Nonetheless, we have decided to use Telegraf and with this the InfluxDB as a starting point for our toolkit. This is mainly due to the large amount of already available plugins, making it easy to adapt the complete toolkit to the needs of different data centers. This should be beneficial for the future roll-out of the toolkit, as we do not need to formulate stringent requirements in terms of used hard- and software. At the same time this mitigates the development efforts needed to enable a potentially nationwide roll-out.

References

[1] "Telegraf from InfluxData | Agent for Collecting & Reporting Metrics & Data" [Online]. URL: https://www.influxdata.com/time-series-platform/telegraf/ (Retrieved: 01 October 2017).

[2] "TICK Stack Based Open Source Platform | InfluxData" [Online]. URL: https://www.influxdata.com/time-series-platform/ (Retrieved: 01 October 2017).

[3] "InfluxDB | The Time Series Database in the TICK Stack" [Online]. URL: https://www.influxdata.com/time-series-platform/influxdb/ (Retrieved: 01 October 2017).

[4] "Kapacitor | Real time stream Processing Engine for InfluxData" [Online]. URL: https://www.influxdata.com/time-series-platform/kapacitor/ (Retrieved: 01 October 2017).

[5] "Chronograf | Complete Interface for the InfluxData Platform" [Online]. URL: https://www.influxdata.com/time-series-platform/chronograf/ (Retrieved: 01 October 2017).

[6] "GitHub - influxdata/telegraf" [Online]. URL: https://github.com/influxdata/telegraf (Retrieved: 01 October 2017).

[7] "Grafana - The open platform for analytics and monitoring" [Online]. URL: https://grafana.com/ (Retrieved: 01 October 2017).

[8] "OpenTSDB - A Distributed, Scalable Monitoring System" [Online]. URL: http://opentsdb.net/ (Retrieved: 01 October 2017).

[9] "Icinga - Open Source Monitoring" [Online]. URL: https://www.icinga.com/ (Retrieved: 01 October 2017).

[10] "InfluxData | Documentation | Telegraf Version 1.4 Documentation" [Online]. URL: https://docs.influxdata.com/telegraf/v1.4/ (Retrieved: 01 October 2017).

[11] "Open Source Search & Analytics · Elasticsearch" [Online]. URL: https://www.elastic.co (Retrieved: 01 October 2017).

[12] "Amazon CloudWatch - Cloud & Network Monitoring Services" [Online]. URL: https://aws.amazon.com/cloudwatch/ (Retrieved: 01 October 2017).

[13] "GitHub - telegraf/plugins/outputs · influxdata/telegraf" [Online]. URL: https://github.com/influxdata/telegraf/tree/master/plugins/outputs (Retrieved: 01 October 2017).

[14] Leibniz Rechenzentrum. "PerSyst Report" [Online]. URL: https://www.webapps.lrz.de/persystreport_demo_fat/ (Retrieved: 01 October 2017).

[15] Carla Beatriz Guillén Carías. "Knowledge-based Performance Monitoring for Large Scale HPC Architectures". Dissertation. Technische Universität München, 2015. URL: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20150610-1237547-1-7.

[16] ProfiT-HPC Consortium. "Deliverable 2.2: Functional Specification of the Backend - Part 1: Pre-Selection of the Metric Collection Tools" [Online]. URL: https://profit-hpc.de/wp-content/uploads/2017/09/profithpc_deliverable_2-2-1_final-1.pdf (Retrieved: 01 October 2017).

[17] "Periscope Tuning Framework" [Online]. URL: http://periscope.in.tum.de/ (Retrieved: 27 September 2017).

[18] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra. "Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats". In: *2014 First International Workshop on HPC User Support Tools*. 2014, pp. 13–21. DOI: 10.1109/HUST.2014.7.

[19] "TACC Stats - Texas Advanced Computing Center" [Online]. URL: https://www.tacc.utexas.edu/research-development/tacc-projects/tacc-stats (Retrieved: 01 October 2017).

[20] R. T. Evans, J. C. Browne, and W. L. Barth. "Understanding Application and System Performance Through System-Wide Monitoring". In: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2016, pp. 1702–1710. DOI: 10.1109/IPDPSW.2016.145.

[21] "tacc_stats" [Online]. URL: https://github.com/TACC/tacc_stats (Retrieved: 01 October 2017).

[22] "Supremm by ubccr" [Online]. URL: https://ubccr.github.io/supremm/ (Retrieved: 01 October 2017).

[23] Center for Computational Research University at Buffalo. "SUPReMM" [Online]. URL: https://github.com/ubccr/supremm (Retrieved: 01 October 2017).

[24] "Open XDMoD" [Online]. URL: http://open.xdmod.org/ (Retrieved: 01 October 2017).

[25] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr. "XSEDE: Accelerating Scientific Discovery". In: *Computing in Science Engineering* 16.5 (2014), pp. 62–74. ISSN: 1521-9615. DOI: 10.1109/MCSE.2014.80.

[26] "Home - XSEDE" [Online]. URL: https://www.xsede.org/ (Retrieved: 01 October 2017).

[27] Robert DeLeon Matt Jones Steve Gallo. "XD Net Metrics Service (XMS)". XSEDE Quarterly Meeting. 2017. URL: https://confluence.xsede.org/display/XT/XSEDE+Quarterly+Meeting+-+March+2017 (Retrieved: 01 October 2017).

[28] InfluxData. "RabbitMQ Input Plugin" [Online]. URL: https://github.com/influxdata/telegraf/tree/master/plugins/inputs/rabbitmq (Retrieved: 01 October 2017).

[29] "XALT - Texas Advanced Computing Center" [Online]. URL: https://www.tacc.utexas.edu/research-development/tacc-projects/xalt (Retrieved: 01 October 2017).

[30] Fahey-McLay. "XALT" [Online]. URL: https://github.com/Fahey-McLay/xalt (Retrieved: 01 October 2017).

[31] Todd Evans, William L. Barth, James C. Browne, Robert L. DeLeon, Thomas R. Furlani, Steven M. Gallo, Matthew D. Jones, and Abani K. Patra. "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: IEEE Press, 2014, pp. 13–21. ISBN: 978-1-4673-6755-4. DOI: 10.1109/HUST.2014.7. URL: http://dx.doi.org/10.1109/HUST.2014.7.

[32] "collectl" [Online]. URL: http://collectl.sourceforge.net/ (Retrieved: 01 October 2017).

[33] "collectl download | SourceForge.net" [Online]. URL: https://sourceforge.net/projects/collectl/ (Retrieved: 01 October 2017).

[34] Peter Piela. "collectl-lustre" [Online]. URL: https://github.com/pcpiela/collectl-lustre (Retrieved: 01 October 2017).

[35] "Colmux" [Online]. URL: http://collectl-utils.sourceforge.net/colmux.html (Retrieved: 01 October 2017).

[36] "collectl - Performance" [Online]. URL: http://collectl.sourceforge.net/Performance.html (Retrieved: 01 October 2017).

[37] "collectd website" [Online]. URL: https://collectd.org/ (Retrieved: 01 October 2017).

[38] "collectd GitHub project page" [Online]. URL: https://github.com/collectd/collectd (Retrieved: 01 October 2017).

[39] "Ganglia Homepage" [Online]. URL: http://ganglia.sourceforge.net/ (Retrieved: 26 September 2017).