A Resource Utilization Analytics Platform Using Grafana and Telegraf for the Savio Supercluster

Nicolas Chan*
nicolaschan@berkeley.edu
University of California, Berkeley

ABSTRACT

Understanding high performance computing cluster utilization patterns is key for decision making and efficient resource allocation. Cluster utilization statistics are useful for both cluster administrators and users, providing the ability to identify potential issues, plan jobs to minimize time spent in the queue, and identify constrained resources where funds might be focused in the future. Much of the existing data is accessible by the command line, but presenting the statistics visually allows for easier identification of trends and interactive exploration. Programs such as XDMoD exist to perform a similar function, but are not as well-suited towards use-cases of smaller clusters. Instead, we use a stack of open source software, allowing for a high degree of flexibility, including multiple database backends and more user-friendly querying and visualization. Based on open source software, similar software stacks can be deployed at other clusters to fit their existing infrastructure and needs. We present the workflow used by Berkeley Research Computing to consolidate existing data, collect additional utilization information, and display the relevant charts for different use-cases on the Savio supercluster. We have begun collecting, integrating, and displaying job information from Slurm; account association information; and CPU metrics collected with Telegraf. With this data, the Grafana visualization framework is able to present broad summary statistics, such as aggregated usage by campus department, all the way down to the CPU usage on a single node over the course of a job, and all with flexibility for a wide variety of possible needs and use-cases.

CCS CONCEPTS

• **Human-centered computing** → **Visual analytics**; **Information visualization**; • **General and reference** → *Metrics*.

KEYWORDS

visualization, high performance computing, metrics, utilization, analysis

ACM Reference Format:
Nicolas Chan. 2019. A Resource Utilization Analytics Platform Using Grafana and Telegraf for the Savio Supercluster. In *Practice and Experience in Advanced Research Computing (PEARC '19), July 28-August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3332186.3333053

PEARC '19, July 28-August 1, 2019, Chicago, IL, USA

ACM ISBN 978-1-4503-7227-5/19/07…$15.00
https://doi.org/10.1145/3332186.3333053

1 INTRODUCTION

High performance computing (HPC) clusters are highly complex systems and continue to grow in complexity over time. As users require new types of compute resources, additional storage and networking capabilities, and the use of new software, understanding how the cluster is actually being used and making strategic and planning decisions becomes increasingly difficult over time.

To effectively make decisions in this complex environment, the data about the cluster state and utilization needs to be easily visible and presented in a way that makes sense for the use-cases of users, system administrators, and managers. Overall, the data collected and displayed by this project must allow its users to make better decisions about how to more efficiently use the cluster resources. System administrators also need to be alerted of problematic situations, such as in the cases of storage space reaching capacity or unusually high CPU activity on the login nodes.

The available metrics revolve around four main areas: compute utilization, storage utilization, network utilization, and hardware state. We focused initially on compute utilization, particularly collecting job data from the Slurm scheduler, augmenting it with additional data from other sources, and combining this data to create appropriate visualizations.

Users need to be able to visualize how their jobs are actually using compute resources in order to use the requested resources most efficiently. Debugging issues such as low performance over the course of a job becomes easier with a visualization of CPU usage over the time the job ran on the relevant nodes. Managers need to understand more broadly how the system is being used, such as which departments use which compute types. To display this information, we must monitor the requested utilization from Slurm (by node type) and actual resources used (such as CPU and memory usage) at any given time. For administrators, visualization of compute utilization allows for identification of problematic usage, such as users running compute-intensive work on the login nodes.

Much of the data necessary to answer these questions is already collected and recorded by Slurm and is accessible through its command line interface using commands such as `sinfo` and `sacct` [9]. Therefore, the bulk of the work for this project is aggregating this data and displaying it a visual way that is tuned to the various use-cases described earlier.

In the following sections, we present related work in the area of HPC utilization visualization, provide a high level overview of the system architecture, describe implementation details of the Slurm data collection and visualization, and finally propose additional areas of interest for expanding the capabilities of the system.

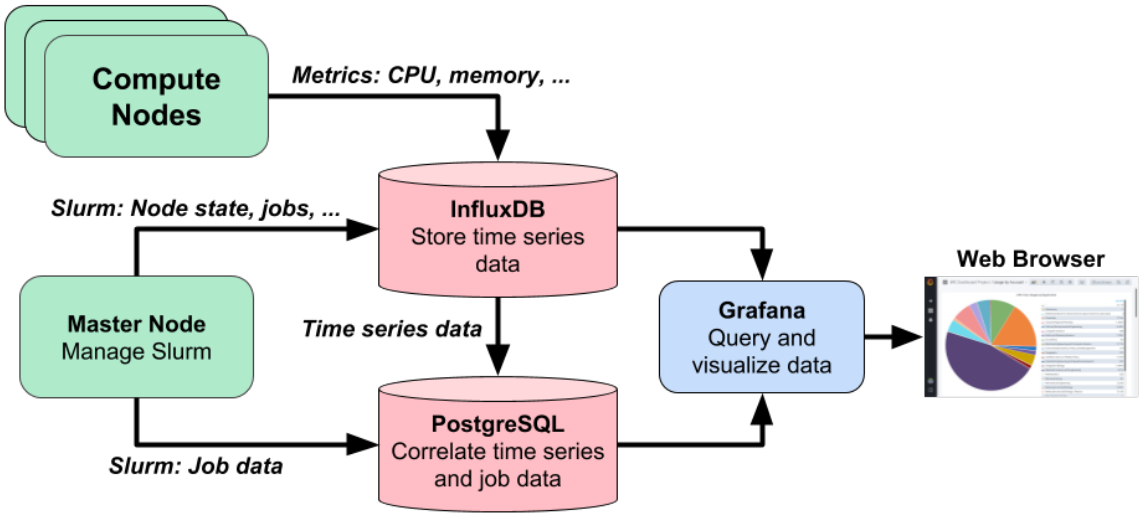

Figure 1: Data flows from Telegraf and Slurm on the nodes and is stored in either InfluxDB or PostgreSQL. Grafana uses both databases as backends and queries and visualizes the data for users in a web browser.

2 RELATED WORK

Monitoring cluster utilization is not a new idea, and various tools already exist to do this job with different areas of focus.

XDMoD [7] provides a comprehensive ability to collect usage data from Slurm and various sources and present this information visually according to different use-cases [2]. XDMoD has many of the capabilities we need and is deployed at clusters that are part of XSEDE. However, XDMoD would require additional configuration to integrate with data from sources besides Slurm, such as the account name to department mappings. While this would likely be possible to do, we decided not to go this route, preferring to build upon our existing usage of Telegraf [11].

Prometheus is another software project that focuses on providing precise alerting capabilities [8], but does not provide visualization capabilities and instead recommends using Grafana as a front-end if high quality visualization is necessary. We preferred to use Grafana directly since adding another application would add to the complexity of maintenance.

Grafana is a visualization tool for the web browser which focuses on analytics and monitoring, supports data from many different backend sources, and provides alerting capabilities [3]. It has been used before in HPC, such as for monitoring I/O performance [1].

We also considered Metabase [6] for the visualization role, but it did not support InfluxDB [4] (the high-performance time series database preferred by Telegraf) at the time of this project and does not focus on time series data. It does provide the ability to make public dashboards, but these lack the interactive abilities of Grafana to allow users to select time ranges.

After looking over alternatives, we settled on Grafana and Telegraf so that we could utilize existing infrastructure and provide the visualization and alerting capabilities of Grafana as a single application, allowing easier long term maintenance compared to maintaining an additional software stack. Additionally, we preferred Grafana's interface to that of XDMoD because of its user-friendly graph interaction, allowing users to drill-down and select the time range of interest, and ease of embedding the charts in existing documentation. We could also create highly customized queries and dashboards. System administrators require alerting capabilities, and Grafana provides many alerting channel types, including email and Slack, which match our organization's preferred communication channels.

3 HIGH LEVEL SUMMARY

To collect the metrics about each node (CPU, memory, disk usage), we deploy an instance of Telegraf to all the compute nodes and login nodes in the cluster. The resource utilization of Telegraf on the compute nodes appears to be negligible, using a similar amount of CPU time as the Slurm daemon. Telegraf running on the compute nodes feeds into an InfluxDB time series database [4] which is accessible through Grafana. It is simple to add additional Telegraf input plugins which run at various intervals.

The scripts we wrote for collecting Slurm data are available in a public repository under an open source license under the UC Berkeley Research IT organization on GitHub (https://github.com/ucb-rit/savio-analytics-dashboard). While the scripts we use are tuned to our specific needs, they can serve as a template that can be adapted to meet the needs and infrastructure of other clusters. The

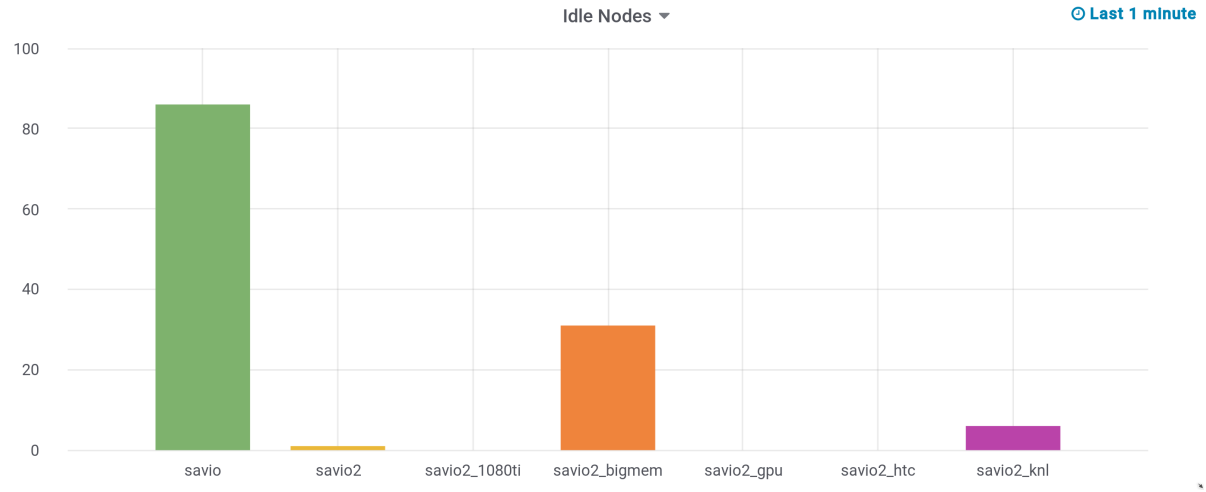

Figure 2: This chart shows the number of nodes currently idle in each partition. If a user's code could use either of the Savio or Savio2 partitions, the user might choose to use Savio since there are more nodes idle. This helps to minimize wait time in the queue, helping to more efficiently utilize cluster resources overall.

Table 1: InfluxDB Line Protocol Data Samples

Description	Data Sample
Allocation	`allocation,partition=savio2 allocated=132,idle=19,total=163,unavailable=12`
Node State	`node_state,hostname=n0039.savio2,state=allocated responding=1`
Queue (pending state)	`queue,account=ac_cdcal,partition=savio2,state=PENDING nodes=2,cpus=312`
Queue (running state)	`queue,account=co_planets,partition=savio2,state=RUNNING nodes=4,cpus=104`

Telegraf "exec" plugin on the master node runs the Bash scripts `main.sh` and `slow.sh` which run at a fast (30 seconds) and slow (1 day) interval, respectively. Using some Python to help with the formatting, these collect the Slurm data using the Slurm command line interface, tag the data appropriately, and format the data according to the InfluxDB Line Protocol [5].

For running more complex queries, such as correlating CPU data with data from a job, we need to use a relational database, and we chose PostgreSQL for this role. Using a script running daily in a crontab, PostgreSQL ingests CPU data from Telegraf and also collects Slurm job data for the previous day.

On the front-end, Grafana allows us to create graphs and dashboards based on the data in InfluxDB and PostgreSQL, as well as set up alert rules and communication channels. Figure 1 provides a diagram of the overall data flow.

4 IMPLEMENTATION DETAILS

This section focuses on the implementation details for collecting and visualizing the information of interest on the UC Berkeley Savio supercluster. As discussed in the high level overview, most of the data is collected from Slurm and processed using a collection of Bash and Python scripts.

4.1 Job Planning

When deciding between partitions during job submission, users need to know the relative number of idle nodes in order to request the less congested partition. To provide this information, we need to collect and visualize the current node status and queue state.

4.1.1 Collecting Node States. The current node states are available through the Slurm command line interface with the `sinfo` command. The `main.sh` Bash script, which is called at frequent intervals, collects the node states and partition allocation data, and formats it according to the InfluxDB Line Protocol, which is then ingested by Telegraf and stored by InfluxDB. `allocation` allows displaying statistics by partition over time and `node_state` allows tracking the state of individual nodes.

Sample allocation and node state data samples, formatted according to the InfluxDB Line Protocol are presented in Table 1. This format consists of key value pairs (separated by an equals sign). The pairs on the left are for tags which allows filtering based on attributes, while the pairs on the right are for numerical data.

4.1.2 Visualizing the Node State. The simplest way to visualize the node state is with a bar chart showing the number of nodes currently idle, presented in Figure 2. This allows users to decide between partitions and estimate wait times.

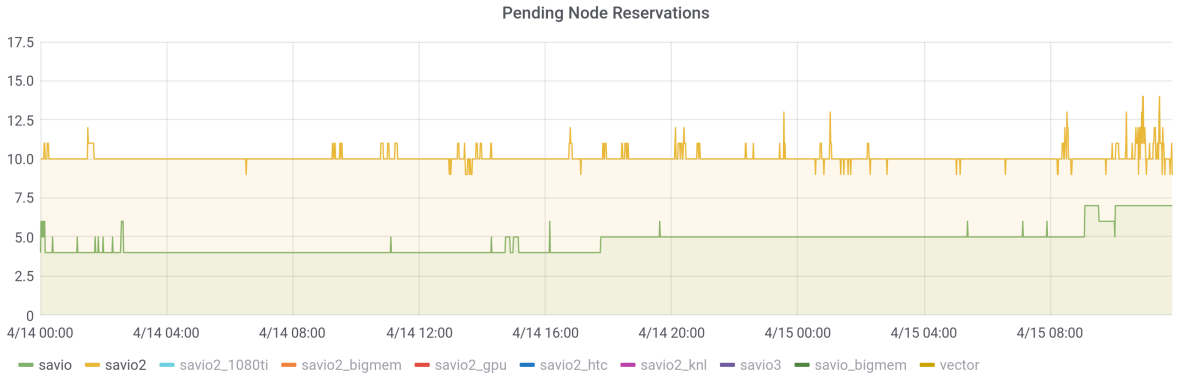

Figure 3: This chart shows the number of nodes requested in the queue for the Savio (green) and Savio2 (yellow) partitions over time. A lower number of nodes in the queue for a partition would indicate the job would likely run sooner. While it is not possible to perfectly predict queue wait time, estimates like the one provided here can help users decide between possible choices of partitions.

4.1.3 Collecting Queue State. The current state of the Slurm queue is already easily accessible through the Slurm command line interface with the `squeue` command. The `main.sh` Bash script, which is called at frequent intervals, collects the state of the queue and currently running jobs. It includes the necessary information for filtering the queue, including job ID, job state, and type of resources requested. Another script takes the raw queue data and groups the data by account and partition type. Finally, the data is formatted according to the InfluxDB Line Protocol which is then ingested by Telegraf and stored in InfluxDB.

Queue data samples, formatted according to the InfluxDB Line Protocol, are presented in Table 1.

4.1.4 Visualizing the Queue. A simple InfluxDB query is able to filter the queue data to only the nodes that are in the queue by requiring the reported state to be `PENDING`. Grouping the nodes by partition and summing over the time interval provides a historical view of the number of requested nodes in the queue (Figure 3).

4.2 Job Analysis

Analysis of jobs, both on an individual and aggregate level, is provided by ingesting data from the Slurm command line interface into the database backends. Grafana allows us to use this data to gain insight into the type of utilization going on in the cluster, as well as drill down to the individual job level to diagnose potential problems, such as low CPU utilization.

4.2.1 Collecting Slurm Job Data. Slurm records all job data for accounting purposes, which can be queried using the `sacct` command. Job information is augmented with the department name and run at longer intervals (e.g., daily) because the Slurm data can take a few minutes to be processed. The department name is determined by a script called `lookup.sh` which serves as an abstraction for looking up the department name based on the account name returned by Slurm. If installed on a different cluster, `lookup.sh` would have to be modified accordingly. For Savio, we use a simple SQLite database [10] backend populated with the account data from Google sheets.

The `slow.sh` and `postgres-daily.sh` scripts provide the job data to InfluxDB and PostgreSQL respectively. At the same time, `postgres-daily.sh` sends the CPU data stored in InfluxDB to PostgreSQL. PostgreSQL is necessary for performing more complex cross-table queries, such as correlating CPU usage with job information, which InfluxDB is not capable of doing. Since the data is ingested at a slow interval, PostgreSQL does not need to be optimized for ingesting the time series data in the same way that InfluxDB is.

InfluxDB receives data formatted according to the InfluxDB Line Protocol, containing the following data as tags: job ID, account name, account type, department, CPUs requested, partition, and final job state (such as `CANCELLED` or `COMPLETED`). The number of requested nodes, number of allocated nodes, raw time spent on the job, and CPU time spent on the job are stored as numerical fields.

The data is loaded into PostgreSQL according to the following schemas. The `jobs` table holds the raw job data, containing all of the fields like InfluxDB (plus the start and end times explicitly):

```
CREATE TABLE IF NOT EXISTS jobs (
  job_id TEXT PRIMARY KEY,
  account TEXT,
  type TEXT,
  department TEXT,
  cpus INTEGER,
  partition TEXT,
  state TEXT,
  req_nodes INTEGER,
  alloc_nodes INTEGER,
  raw_time DOUBLE PRECISION,
  cpu_time DOUBLE PRECISION,
  start_time TIMESTAMP WITH TIME ZONE,
  end_time TIMESTAMP WITH TIME ZONE);
```

Figure 4: Job information is correlated with the relevant CPU data to give a visual representation of the job over time, allowing for easier debugging. The job shown here has low overall utilization with many spikes, indicating it might be limited by I/O.

The job_nodes table allows a one-to-many mapping of a job to nodes:

```
CREATE TABLE IF NOT EXISTS job_nodes (
  job_id TEXT,
  hostname TEXT);
```

Finally, the CPU data is recorded as a direct translation of the fields present in InfluxDB:

```
CREATE TABLE IF NOT EXISTS cpu (
  timestamp TIMESTAMP WITH TIME ZONE,
  host TEXT,
  cpu TEXT,
  usage_guest DOUBLE PRECISION,
  usage_guest_nice DOUBLE PRECISION,
  usage_idle DOUBLE PRECISION,
  usage_iowait DOUBLE PRECISION,
  usage_irq DOUBLE PRECISION,
  usage_nice DOUBLE PRECISION,
  usage_softirq DOUBLE PRECISION,
  usage_steal DOUBLE PRECISION,
  usage_system DOUBLE PRECISION,
  usage_user DOUBLE PRECISION);
```

4.2.2 Visualizing Slurm Job Data. A PostgreSQL query takes the job information and CPU data and returns only the CPU data for the relevant time range from the nodes the job ran on (Figure 4). With the job correlation to the departments, managers can understand which groups are using particular cluster resources at a glance (Figure 5).

4.3 Evaluation

We have conducted an initial presentation of this analytics platform to users within our organization. The partition availability visualization is already useful when deciding which partitions will be the fastest to use when submitting test jobs. Visualizing node state over time has helped identify issues with nodes. Broad utilization by cluster department allows management to understand how the cluster is being used on a high level.

Additional data, such as memory and job efficiency, should be added to the job information dashboard alongside the existing CPU data. Presentation of trends based on days of the week will provide additional insight into cluster utilization patterns.

5 CONCLUSIONS AND FUTURE WORK

Presenting the job data from Slurm, especially when augmented with additional metrics and tagged with other metadata, allows several different types of users to interact with the cluster more efficiently. This analytics platform, even in its beginning stages, is already useful to users in our organization. Composed of open source software, the stack is highly flexible and can be deployed in other organizations and tailored to their existing infrastructure.

The Slurm data collection scripts require specific dependencies, and in future work we will explore improving portability to other clusters by utilizing tools such as containers to encapsulate all of the scripts and Python dependencies. Optimizations to the scripts and system design may allow for faster updating cycles, perhaps even showing job state as the job is running. We will also consider other database backends, such as the Timescale database [12], which could eliminate the need to run both InfluxDB and PostgreSQL.

However, collecting this data continues to raise more questions about how users are using the cluster and its performance. In future work, we will explore adding deeper integration into Slurm, including extracting data from Slurm job submission scripts to provide insight into what kinds of programs users use most. We will also

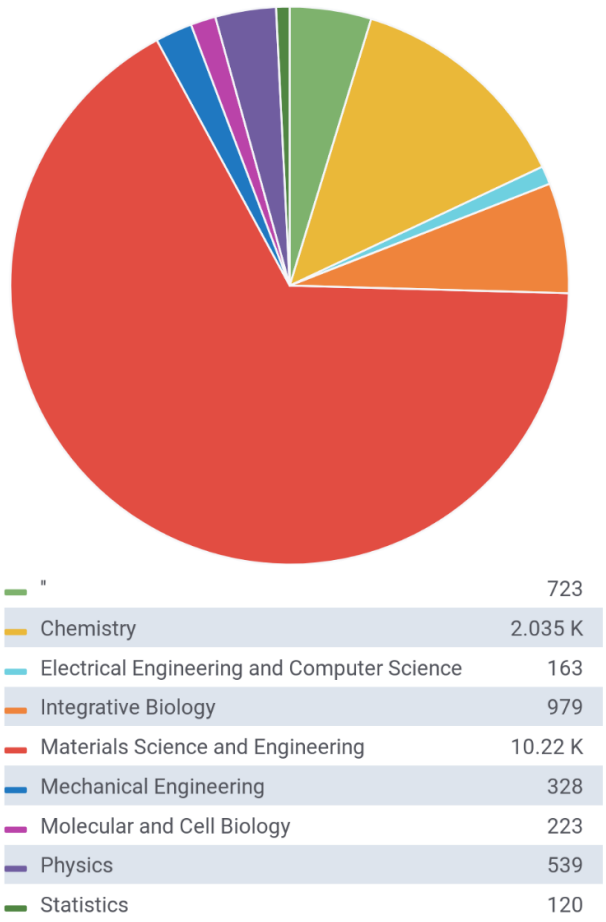

Figure 5: Aggregated cluster usage by department, presented as a sum of CPU time for the time range in question. In this chart, the empty string ('') represents unclassified accounts.

explore collecting additional metrics, such as local disk health or CPU temperatures, allowing proactive detection of node failures.

ACKNOWLEDGMENTS

To John White (Lawrence Berkeley National Lab HPCS Team), for configuring and deploying Grafana, Telegraf, InfluxDB, and PostgreSQL on the Savio supercluster. And to Patrick Schmitz (University of California, Berkeley), for guidance on project goals and feedback on technical writing.

This work is supported by an internship at Berkeley Research Computing (BRC) at the University of California, Berkeley. Members of the BRC group have also contributed their ideas for use-cases of the analytics dashboard.

REFERENCES

[1] Eugen Betke and Julian Kunkel. 2017. Real-Time I/O-Monitoring of HPC Applications with SIOX, Elasticsearch, Grafana and FUSE. In *High Performance Computing*, Julian M. Kunkel, Rio Yokota, Michela Taufer, and John Shalf (Eds.). Springer International Publishing, Cham, 174–186.

[2] Thomas Furlani, Barry Schneider, Matthew Jones, John Towns, David Hart, Steven Gallo, Robert Deleon, Charng-Da Lu, Amin Ghadersohi, Ryan J. Gentner, Abani Patra, Gregor von Laszewski, Fugang Wang, Jeffrey Palmer, and Nikolay Simakov. 2013. Using XDMoD to facilitate XSEDE operations, planning and analysis. *ACM International Conference Proceeding Series* (07 2013). https://doi.org/10.1145/2484762.2484763

[3] Grafana [n. d.]. Grafana - The open platform for analytics and monitoring. Retrieved April 14, 2019 from https://grafana.com

[4] InfluxData, Vlasta Hajek, Thomas Klapka, and Ivan Kudibal. 2018. *Benchmarking InfluxDB vs. MongoDB for Time Series Data, Metrics Management.* Technical Report. Retrieved April 14, 2019 from http://get.influxdata.com/rs/972-GDU-533/images/InfluxDB%201.4%20vs.%20MongoDB.pdf

[5] InfluxDB Line Protocol [n. d.]. InfluxDB Line Protocol reference | InfluxData Documentation. Retrieved April 14, 2019 from https://docs.influxdata.com/influxdb/v1.7/write_protocols/line_protocol_reference/

[6] Metabase [n. d.]. Metabase. Retrieved April 14, 2019 from https://www.metabase.com/

[7] J. T. Palmer, S. M. Gallo, T. R. Furlani, M. D. Jones, R. L. DeLeon, J. P. White, N. Simakov, A. K. Patra, J. Sperhac, T. Yearke, R. Rathsam, M. Innus, C. D. Cornelius, J. C. Browne, W. L. Barth, and R. T. Evans. 2015. Open XDMoD: A Tool for the Comprehensive Management of High-Performance Computing Resources. *Computing in Science Engineering* 17, 4 (July 2015), 52–62. https://doi.org/10.1109/MCSE.2015.68

[8] Prometheus [n. d.]. Prometheus - Monitoring system time series database. Retrieved April 14, 2019 from https://prometheus.io/

[9] Slurm Documentation [n. d.]. Slurm Workload Manager - Documentation. Retrieved April 14, 2019 from https://slurm.schedmd.com/documentation.html

[10] SQLite [n. d.]. SQLite Home Page. Retrieved April 14, 2019 from https://sqlite.org/index.html

[11] Telegraf [n. d.]. Telegraf | Agent for Collecting Reporting Metrics Data | InfluxData. Retrieved April 14, 2019 from https://www.influxdata.com/time-series-platform/telegraf/

[12] TimescaleDB [n. d.]. Time-series data simplified | Timescale. Retrieved April 14, 2019 from https://www.timescale.com/