Iuhasz, G.; Petcu, D. Monitoring of Exascale data processing, in *2019 IEEE International Conference on Advanced Scientific Computing (ICASC), 12-14, September 2019, Sinaia, Romania. Proceedings*
DOI: https://doi.org/10.1109/ICASC48083.2019.8946279

Monitoring of Exascale data processing

Gabriel Iuhasz
Institute eAustria Timisoara and West University of Timisoara
iuhasz.gabriel@e-uvt.ro

Dana Petcu
Institute eAustria Timisoara and West University of Timisoara
dana.petcu@e-uvt.ro

Abstract

Exascale systems are a hot topic of research in computer science. These systems in contrast to current Cloud, Big Data and HPC systems will routinely contain hundreds of thousand of nodes generating millions of events. At this scale of hardware fault and anomalous behaviour is not only more likely but to be expected.

In this paper we describe the architecture of and Exascale monitoring solution coupled with an event detection component. The latter component is extremely important in order to handle the multitude of potential events. We describe the major lacking research that needs to be done, which will make event detection freezable in real world Exascale systems.

Keywords Exascale, H2020, monitoring, event processing, anomaly detection
Formal publication `https://doi.org/10.1109/ICASC48083.2019.8946279`

I. Introduction

System monitoring for large-scale HPC platforms is a challenging task, which becomes more difficult as the scale and complexity of the infrastructure increases. This is true both in the case of hardware as well as the underlying software stack. In such cases, the system mean time between failures tends to decrease inversely proportional to the number of components. Therefore, the applications can experience interruptions in service due to hardware failures or misconfigurations. As the HPC systems' size increases, application failures become a critical issue, which could have profound effect on the overall system performance. In addition to hardware failures, novel system software stacks coupled with legacy parallel scientific applications deployed on modern cluster platforms push the envelope of reliability. The DARPA Exascale Computing Study [1, 11] predicts that at Exascale the predicted failure rates could be as low as 35-39 minutes.

One important challenge in Exascale computing consists of developing scalable components that are able to monitor in a coordinated and efficient manner the use of hardware resources and the behaviour of applications. However, monitoring Exascale system is a challenge due to the large number of components and the tight requirements, such as sub-optimal period scheduling [7]. Moreover, in HPC systems the schedulers assign compute nodes in a static way, trying to run different applications in different nodes to avoid interferences. They use CPU availability as main criteria on the scheduler side and HPC system statistics, as TACC Stats in [5], to improve the system utilization. However, as nodes scale-in, a single failure could affect multiple applications in the system simultaneously. Thus, efficient system monitoring plays an important role in three aspects: 1) tuning the computing infrastructure;2) optimizing scheduling in order to share resources and to provide high efficiency; and 3) detect faulty components/nodes in the system.

II. Current monitoring solutions

Essential problem in extreme scale systems is their scalability due to large numbers of resource and huge data amounts to be transferred, stored and accessed. These facts impose a challenge on monitoring system to provide insightful data allowing to avoid or remove bottlenecks related to data congestion, and in this way, improve performance and efficiency of large scale applications. On the other hand, the scale and complexity of such systems put important constraints on the monitoring system itself.

There are a large variety of monitoring solutions available for large scale systems. In particular the most popular monitoring solutions are external services at least in the case of Cloud and Big Data framework monitoring.

NewRelic[1] provides a solution for monitoring both the infrastructure in a traditional, hybrid or Cloud setup as well as the applications running on them. It provides "serverless" solution capable of handling a wide variety of metrics including those from code instrumentation.In this situation, code refers both to actual code that runs as well as the VM's on which the code is running. This means that users can observe the effects of deploying new features on preexisting deployments and infrastructure.

Honeycomb[2] is designed as a platform driven by events intended to debug systems, applications and databases. Aggregation is done in real-time for use in fast analytics and data analysis. It is designed to hide as much as possible of the underlying schemas or data indexes. It also features integration with various communication media, in particular with Slack which aids in communication and alerting of detected events.

DataDog[3] is a service based monitoring solution that provides full stack monitoring. It handles infrastructure monitoring, application monitoring as well as log management. It is designed to be used in DevOps type workloads in particular for aggregating metrics and events.

[1] https://newrelic.com/
[2] https://honeycomb-analytics.com/
[3] https://www.datadoghq.com/

Although these service based solutions fit many large scale system requirements they have 2 major drawbacks. First, almost all of them are paid services. Payment can be done on a subscription basis or on the number of incoming metrics and events. The latter being particularly bad scenario for Exascale systems where hundreds of thousands of nodes will generate millions if not billions of messages and events.

An alternative to these types of service based solutions are usually made up of different components each having a well defined scope. One interesting an well used and liked monitoring stack is the so called ELK[4] stack made up of 3 main components; ElasticSearch, Logstash and Kibana.

Logstash is a tool developed in order to collect, process and forward events and log messages. Basically, it handles Extract, Transform and Load (ETL) operations. It uses configurable plugins for input, output and filters in order to collect, process and load data. The input plugins then send the data for processing to the filter workers. Finally, the processed data is routed to one or more output plugins such as Elasticsearch, Kafka, InfluxDB and many more types of data stores. It is an extremely useful tool in any monitoring setup as its configurability means that it can fill many roles at the same time.

ElasticSearch is an open-source, RESTful search engine based on top of Apache Lucene[5]. It is horizontally scalable n which can perform near real-time processing. It also provides support for multi-tenancy, streamlined backup procedures as well as insuring data integrity. One of the most important capabilities of Elasticsearch is its ability to handle high throughput of tens or even hundreds of thousands of messages per second. Each node, which makes up an Elasticsearch cluster can be configured to have a specific role from serving as data store node, to load balancer nodes and coordination or master node.

Ganglia[6] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. The design of Ganglia is hierarchical and targets federation formation for clusters. Because of the use of hierarchical approach, it manages preserve a low per-node overhead and high concurrency. By design it is robust and easy to be ported on various operating systems. One of its major benefits is that it can easily scale-up to handle thousands of nodes. Nowadays it is being used in lots of setups around the world.

Nagios[7] monitoring platform supports multi-layer monitoring. Due to its plugin based architecture, Nagios provides a way of monitoring both Cloud based resources as well as in-house infrastructure. In order to achieve this it uses SNMP monitoring network resources. The architecture of Nagios requires a centralized server in order to collect monitoring data. However, it is possible to create a hierarchy of Nagios servers that mitigate the disadvantages of a centralized server.

D-Mon[4], is a monitoring tool developed during the H2020 DICE research projects. It was designed for the monitoring of data intensive applications during the development phase focusing on Big Data frameworks. In contrast to other solutions it provides a detailed snapshot for different versions of the same application. It is based on the ELK stack with a custom REST API which enables easy querying of monitoring data. It is able to respond in a comprehensive variety of formats (JSON, CSV, Plain, XML). Analysis tools are able to easily query D-Mon for monitoring data. To further increase the usability and reduce the workload on the analysis tools D-Mon also supports some data pre-processing steps during interrogation. Furthermore, it also supports most major big data technologies; HDFS, YARN. Spark, HBase, Storm, Flink, MongoDB, Cassandra, Solr, Elasticsearch.

Figure 1: ASPIDE Monitoring

III. ASPIDE Monitoring and event detection

We propose a distributed event processing platform that is capable of analyzing the incoming monitoring data. This analysis allows end users to have an up to date cross-section of Exascale applications internal state and performance. One important distinction of large scale systems is that there is a substantial quantity of data available. However, this data is unusable by many of the available analysis methods. This is due in large part to the fact that semantically labeled data is very hard to obtain or create, requiring a lot of effort which in many cases is not transferable from one large scale system to another. In this work we aim to present a comprehensive overview of the different technologies used that enable our event detection system to tackle this type of analysis. We describe the requirements for identifying events and anomalies as well as methods of utilizing unsupervised methods in conjunction with continual user feedback.

We can see in figure 1 the overall monitoring architecture of the ASPIDE monitoring solution. The figure has been taken from the technical deliverable describing it in detail [8]. The following paragraphs describe in short the aforementioned architecture.

Node collectors should be lightweight and highly configurable, collection from several layers of the underlying software stack. In our solution we use Collectd[8]. This metrics collection daemon is widely used in HPC, Cloud and even in DevOps use-cases. There is an active user and developer community. Being plugin based means that it is highly adaptable. Users can create custom metrics (even

[4]https://www.elastic.co/
[5]https://lucene.apache.org
[6]http://ganglia.info
[7]https://www.nagios.org/
[8]https://collectd.org

code instrumentation) and metrics forwarding plugins with relative ease in a variety of programming languages (C, Python, Java etc.).

Event processors, perform pre-processing of incoming raw monitoring data as well as computationally tractable analysis tasks. In our solution we use a combination of Logstash and Kafka. This setup allows us to instantiate several Logstash deployments serving as both pre-processing nodes and potentially metrics load balancer. Kafka allows us to push incoming events into topics which can then be fed into long term storage and even into other tools that require online analysis.

Time series DB must be capable of being distributed and handle large input and query size. As mentioned before Elasticsearch is one of the most popular solution for persistent storage of monitoring data. However it can be quite difficult to scale. If the number of shards is fixed at a lower limit than is required for scaling past a certain point the entire data store has to be re-indexed into the increased number of shards. Furthermore, the index typically takes up 25 to 30% of the total stored data.

Graphite[9] is designed to store numeric time-series data and to render graphs and other visualization in near real-time. It is one of the most widely used open source tools for monitoring computer system performance. It is comprised of 3 main components; carbon responsible for receiving incoming data, whisper which stores the incoming data similar to the industry standard RRD toolset and finally a graphing web interface. It can be used with a wide array of collection and visualization components such as Grafana. One issue when dealing with Graphite is that it's data store is of fixed size and has to be sized upfront. It stores recent data at high resolution (seconds per point) which gradually degrades for long term retention. This can be an issue for Exascale systems.

InfluxDB[10] is a time-series database written in the Go programming language. It is optimized for time-series data with high-availability. It is frequently used for monitoring infrastructure as well as IoT and real-time analytics. It has built in support for processing Graphite data. It has an SQL-like language which is used to gather data composed of measurements, series and points. In InfluxDB a point is defined as a key-value pair and a timestep. It supports several datatype such as 64 bit Int, Float as well as string and boolean. It also has a new data scripting language called Flux. One major issue when using InfluxDB is that the components which enabled horizontal scaling are no longer open-source as of version 0.12.0.

Prometheus[11] is another open-source software application used for event monitoring and alerting. Alerting is statically configured with a YAML file. Prometheus records real-time metrics in a time series database. Contrary to Graphite, Prometheus applies polling to collect metrics values instead of subscription to approaching events.

We can see that none of the persistent storing solutions is a perfect fit for the needs of Exascale systems. In ASPIDE we have decided to use Prometheus in combination with Elasticsearch for the initial implementation. Automatic scalability and resilience of this part of the monitoring system is still an open question and is an issue which requires further research.

Distribution and Optimization Manager handles the controlling of collectors and aggregation points. It also handles also optimization during deployment. Because of the sheer scale of Exascale system manual installation and configuration is not freezable. There are configuration management solutions that can handle this task such as Puppet and Chef, however they are not enough in our case. Not only will this system have to handle fault event and new nodes being added to the system but also with configuring the aggregation nodes. In short it must decide how many nodes are assigned to and aggregation point, what operations it applies to the incoming data and finally which node from the Exascale system will be assigned the role of aggregation node.

Decision Support system (Auto tuner) handles intelligent configuration (data centrist) of Exascale application and tasks. It will receive data from both the runtime, monitoring and event detection and use this to generate new configurations. Basically creating a feedback loop which optimizes application execution.

IV. Event and metrics

The event detection sub system will handle the processing of raw monitoring data in order to identify events and anomalies during application execution. The event processing and detection components architecture (called EDE) is described in full in [6].

Figure 2 show how EDE is integrated into the exascale monitoring. It collects historical data from the monitoring which it then applies pre-processing and formatting tasks to, so that it can be finally uses to train predictive models. The events and anomalies detected are then fed back into the decision support system to use it in fine tuning.

The question at this point is what are the events or anomalies and what metrics are required? This is a key issue which to the best of our knowledge has no clear response at this time. What can be inferred is that there are two types of methods which can be used in this scenario; supervised and unsupervised. Arguably the easiest one to use are the unsupervised methods where we work under the assumption that normal events are grouped together in relatively dense clusters while anomalies are sparse. The major downside of these methods is that even if we successfully detect an event we can't always clearly identify what type of the event or anomaly is nor can we easily analyze the root cause.

In the case of supervised methods root cause analysis is made simpler. However, contextual anomalies root cause analysis is still difficult. By contextual anomalies we mean those anomalies for which each metric taken by itself is not anomalous but in a particular context the event can be. The context is given by the metrics collected at a given time.

Defining context (in other words creating labeled training data) is a quite difficult to do, and usually requires human intervention. There are semi automated methods of doing this (injecting anomalies via predefined rules) do exist however this synthetic data is of limited practical use.

There are some available data sets designed for use in event and anomaly detection in the case of HPC and Cloud computing [3, 10], however they are few and far between. This can be traced back to the lack of access to the underlying system and the lack of interest by infrastructure providers to make their monitoring data publicly available.

In order to define events and anomalies for Exascale systems we need to have input form the types of use cases which require these

[9] https://graphiteapp.org/
[10] www.influxdata.com
[11] https://prometheus.io/

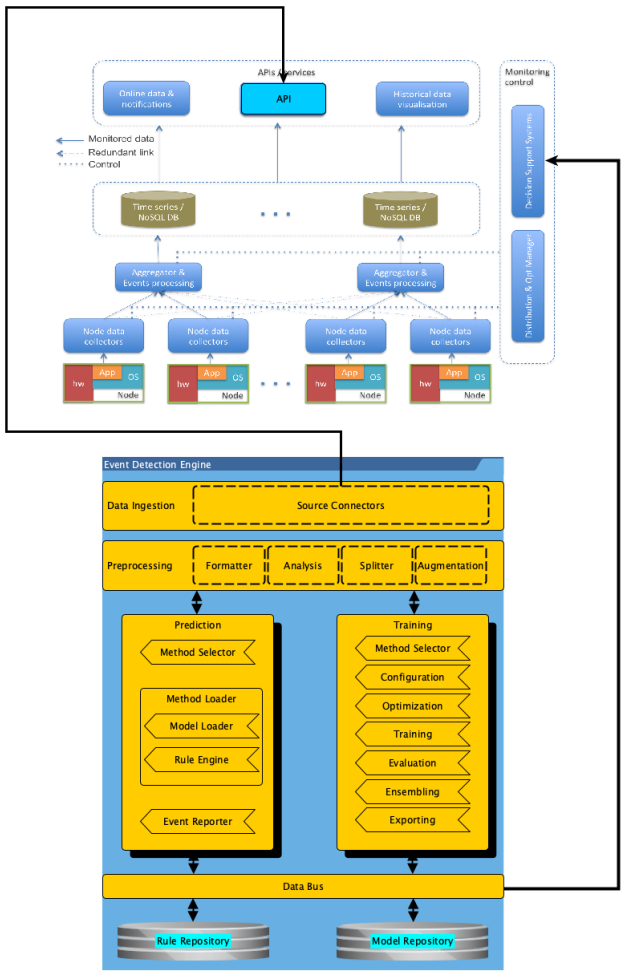

Figure 2: ASPIDE Monitoring and Event detection

types of systems. Without this it is difficult to gauge is the events and anomalies which are of interest in the case of current production HPC and Cloud/Big Data based system can be mapped on Exascale.

V. Conclusions

In this paper we have described currently available monitoring solutions and the technologies on which they are based. We have detailed both the positive and negative aspects of the technologies which could make up our Exascale monitoring system. As well as describing the current working architecture. Furthermore we described the requirements for a viable event detection system.

Future work will focus in testing the monitoring system implementation and checking for scalability and elasticity. In the upcoming weeks we will also focus on creating a comprehensive list and description of the potential events and anomalies which might effect any Exascale system. In order to accomplish this we will use the use cases from the ASPIDE H2020 project [2, 9].

Acknowledgment

This work has received funding from the EC-funded H2020 ASPIDE project (Agreement 801091: Exascale programming models for extreme data processing). This work was supported with hardware resources by the Romanian grant BID (PN-III-P1-PFE-28: Big Data Science).

References

[1] ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, 2008.

[2] L. Belcastro, F. Marozzo, D. Talia, and P. Trunfio. Parsoda: high-level parallel programming for social data mining. *Social Network Analysis and Mining*, 9(1):4, Dec 2018.

[3] A. Borghesi, A. Bartolini, and F. Beneventi. Data set for anomaly detection on a hpc system, June 2019.

[4] I. Drăgan, G. Iuhasz, and D. Petcu. A scalable platform for monitoring data intensive applications. *Journal of Grid Computing*, May 2019.

[5] 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*, pages 13–21, Nov 2014.

[6] G. Iuhasz and D. Petcu. Perspectives on anomaly and event detection in exascale systems. In *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pages 225–229, 2019.

[7] W. M. Jones, J. T. Daly, and N. DeBardeleben. Application monitoring and checkpointing in HPC: Looking towards exascale systems. In *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE '12, pages 262–267. ACM, 2012.

[8] A. O. D. E. S. D. K. G. I. J. G.-B. J. C. V. Kashansky. D3.2 extreme scale monitoring architecture. Technical report, H2020 ASPIDE, 2019.

[9] J. LÃpez-GÃmez, J. F. MuÃoz, D. del Rio Astorga, M. F. Dolz, and J. D. Garcia. Exploring stream parallel patterns in distributed mpi environments. *Parallel Computing*, 84:24 – 36, 2019.

[10] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sirbu, A. Bartolini, and A. Borghesi. Antarex hpc fault dataset, Oct. 2018.

[11] Vivek Sarkar et al. ExaScale Computing Software Study: Software Challenges in Extreme Scale Systems, September 2009.