Iuhasz, G.; Petcu, D. Perspectives on anomaly and event detection in exascale systems, in *2019 IEEE 5th International Conference on Big Data Security on Cloud (BigDataSecurity), 27-29 May 2019, Washington, USA*

DOI: https://doi.org/10.1109/BigDataSecurity-HPSC-IDS.2019.00051

Perspectives on Anomaly and Event Detection in Exascale Systems

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

The design and implementation of exascale system is nowadays an important challenge. Such a system is expected to combine HPC with Big Data methods and technologies to allow the execution of scientific workloads which are not tractable at this present time.

In this paper we focus on an event and anomaly detection framework which is crucial in giving a global overview of a exascale system (which in turn is necessary for the successful implementation and exploitation of the system). We propose an architecture for such a framework and show how it can be used to handle failures during job execution.

Keywords Exascale, H2020, machine learning, anomaly, distributed, monitoring

Formal publication `https://ieeexplore.ieee.org/document/8819496/`

I. Introduction

Exascale systems are envisioned to be a great leap in computational power in the not so distant future. Creating a system which has a 50 to 100 fold increase in power when compared with the current supercomputers is not an easy task to accomplish by any stretch of the imagination. These Exascale systems will have to have vastly superior processing units, memory and the ability to store and retrieve large quantities of data at high speed. The latest fact also raises another characteristic which is very commonly assigned to future exascale systems, namely that they are inherently heterogeneous in nature. Many type of computational nodes such as CPUs, GPGPUs and even solution based on many integrated cores (MIC) or even FPGA (Field programmable gate array) based solutions [4]. The vast expansion of computational power is worth while, as speaking from a historical point of view, these types of novel systems open new fields of research such as but not limited to: advances in medicine, material sciences, novel energy solutions, pollution reduction etc.

Besides the hardware advancements, the software stack required to bring into being an exascale system is equally if not more important. Because of the sheer size of these systems. Exascale systems will be in fact the combination of HPC and Big Data in many ways [13] and will require the combination of software frameworks and properties from both types of computation paradigms.

The need for components that are capable of managing and scheduling tasks on such large scale systems and posses mechanisms for autonomously handle events is evident. However, in order for a system to truly be autonomous there is a need for tools that can provide a clear and comprehensive overview of the current state of the exascale system. Monitoring of metrics can provide some of the information required however, components which are able to take historical data and identify not only normal events but also anomalous ones can provide the comprehensive overview of the current system state.

There is already a large amount of research reports being done in defining the challenges inherent in designing an exascale system capable of processing vast amount of data and/or executing scientific calculation which require substantial computational power [15, 3].

In this paper we focus on identifying open research issues and characteristics which are common to all exascale system research being done. We will identify several challenges, in particularly focusing on scalability and fault tolerance in section II. Furthermore, we will see how these challenges are tackled in the current research into exascale systems in section III. The main focus will be on the importance of event detection and how events are handled for scheduling and optimization of workflows. Section IV will focus on identifying the requirements for an event detection framework that is able not only to detect anomalous events but also provide event description and recommendations for root cause analysis and various other event handling methods. In section V we present an overview of the most suitable machine learning and data processing frameworks and libraries that are able to support the implementation of our proposed architecture. Finally, in section VI we draw our conclusions and future steps.

II. Main challenges of exascale systems

Ease of use is important in any complex system and exascale is no different. There is a clear need for the creation of systems and libraries which mask the underlying complexity , making exascale system more easy to use. This obfuscation can only happen if certain tasks are handled autonomously by the system instead of the user. Tasks that deal with data storage and retrieval, resource management and scheduling are excellent targets for complete or partial automation. In order for this to truly happen a system needs to be in place which can identify specific events and faults during job execution, be it at system or application layer.

One of the main issues when dealing with exascale systems is that,

when running jobs on millions of processing nodes, it will increase the occurrence of hardware faults significantly. New methods when dealing with these type of fault events have to be created not only to report the events to the end user, but also to the exascale system (resource management and scheduling subsystems, to be more precise) enabling autonomic handling of these events. In a nutshell, as the rate of fault events increases together with system size, it is feasible to expect that for every job execution any number of nodes have a high chance of failure. There are ways of handling these types of events and increase reliability such as: ensure the availability of critical application data even if global data becomes unavailable (replication), ensuring the correctness and consistency of the data at the start of recovery (metadata that records state transitions). All of the aforementioned methods for handling faults hinge on a fault tolerant management infrastructure that supports continuous execution [12]. The management layer has to have mechanisms for detecting when fault events occur and identifying when this actually occurs.

Energy efficiency is also a hot topic of research and an important metric when it comes to exascale systems. Tuning jobs with regard to energy efficiency not only for execution time is an important step in increasing the overall efficiency of exascale systems. These optimization can take many forms such as moving jobs (if possible) to specialized hardware such as FPGA or GPGPUs which have, in certain circumstances, a better computation to power consumption ratio then traditional CPUs. Again, these optimizations can only be made only if there is a thorough understanding of the entire system state and the underlying events.

Furthermore, there are many similarities between exascale and edge/fog systems. Both are massively distributed and require advanced scheduling and management components. Research done in this field for intrusion detection systems such as the ones detailed in [7] is of particular interest as it can provide a baseline for event detection methodologies. This also leads us to field of research which is extremely active in edge/fog scenarios and which is in our opinion underrepresented in exasscale system research. Namely the problem of privacy and encryption [6, 5].

III. Software support for exascale systems

Systems such as Argos [14] are designed to create an exascale operating system which aims to provide dynamic reconfiguration of nodes based on current work load, allowing for massive concurrency, a hierarchical node management scheme and a cross-layer communication.

Other systems such as ExaNest [11] aim to develop system level interconnect and non volatile storage based largely around low cost and low power components, ARM processors and FPGAs. This trend of designing for energy efficiency can be found in several research initiatives [16, 4]. For example the ATLAS library [10] introduces methods for tuning linear algebra operations for both energy efficiency and execution time. The HiPEAC[1] initiative aims to address the issues of data locality and energy efficiency.

Big Data technologies such as YARN, SPARK, Storm, Flink are all mainstays of today's computing world. A large percentage of research is devoted in making these types of frameworks more efficient, while other initiatives try to bridge the gap between HPC and Big Data such as the Extreme big data JST-CREST program [13].

All of these initiatives and ongoing efforts have one common thread, they require greater insight into the inner workings and states of not only the underlying exascale system but also of the jobs or applications themselves. This in turn enforces the need for a bespoken framework for identifying events and potential faults or anomalies.

IV. Event Detection Framework

The proposed architecture for our event detection engine can be seen in Figure 1. Why this architecture is feasible and appropriate for an exascale system are explained in what follows.

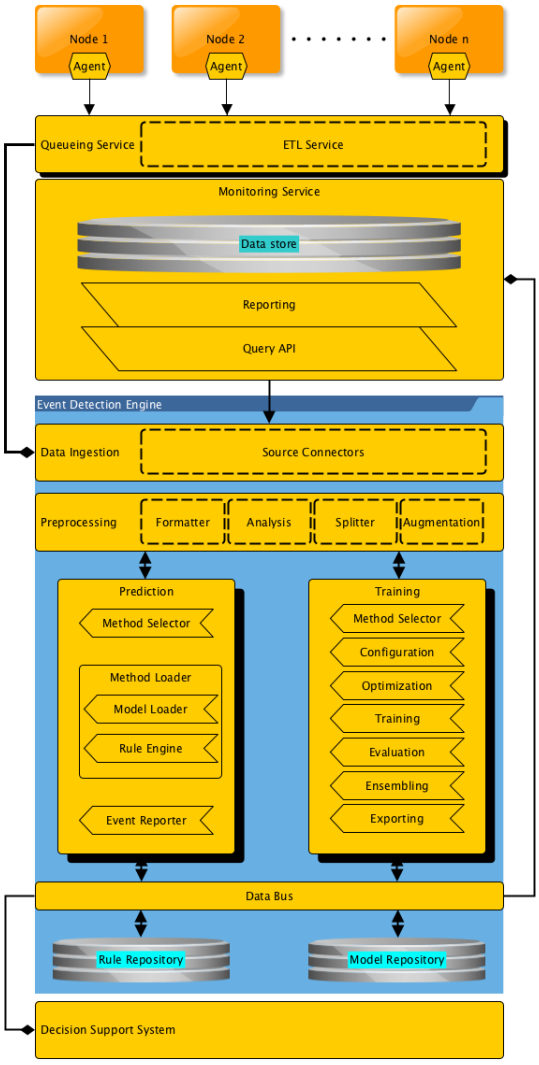

Figure 1: Event Engine Architecture

[1] https://www.hipeac.net/

IV.1 Monitoring

The monitoring infrastructure necessary for monitoring an exascale system is well beyond the scope of this article. However, we do make certain assumptions about such a system that are, with a high degree of certainty not only valid but necessary.

Systems as vast as exascale systems will require a monitoring system which is capable of handling metrics from thousands possible even millions of nodes. We do not make any assumption on the overall architecture of the monitoring architecture, it can be centralized (more likely to be hierarchic) or even completely distributed in nature. Most large scale monitoring solutions have a queuing service or buffer that intercepts the messages from the monitoring agents. This ensures that no new monitoring data is lost while the monitoring service tries to save the previous instances.

This queuing service usually has another functionality, namely it also is in charge of some degree of formatting of the incoming data; extract, transform and load (ETL). This is not always the case, in some situation the monitoring agents themselves are responsible for transforming the incoming monitoring data so that it can be loaded directly into the monitoring service. Even if this is the case queuing service is still crucial for ensuring no data loss.

The monitoring service itself handles storage and serving of monitored data. Data stores range from relational databases, NoSQL databases (i.e. MongoDB, Cassandra, CouchDB) or even time-series databases (i.e. InfluxDB) or search engines (i.e. Elasticsearch). All of these solutions have both their pros and cons. The choice of using one versus the others is highly dependant on the type of monitoring architecture and use case. Regardless of the technology stack chosen, a monitoring solution has to have two functionalities. The first one is an API which allows for external tools to access the data being stored and also to add or modify existing data. Second, a component which allows for end users to view the current status of the system and to report any other information related to what the monitored exascale systems current status is.

IV.2 Events and Anomalies

In the following section we will use the term events and anomalies seemingly interchangeably. However, we should note that the methods used for detecting anomalies are applicable in the case of events. The main difference lies in the fact that anomalies pose an additional level of complexity by their spars nature, some anomalies might have an occurrence rate well under 0.01%.

Event and anomaly detection can be split up into several categories based on the methods and the characteristics of the available data. The most simple form of anomalies are point anomalies which can be characterized by only one metric (feature). These types of anomalies are fairly easy to detect by applying simple rules (i.e. CPU is above 70%). Other types of anomalies are more complex but ultimately yield a much deeper understanding about the inner workings of a monitored exascale system or application [8]. These types of anomalies are fairly common in complex systems.

Contextual anomalies are extremely interesting in the case of complex systems. These types of anomalies happen when a certain constellation of feature values is encountered. In isolation these values are not anomalous but when viewed in context they represent an anomaly. These type of anomalies represent application bottlenecks, imminent hardware failure or software miss-configuration. The last major type of anomaly which are relevant are temporal or sometimes sequential anomalies where a certain event takes place out of order or at the incorrect time. These types of anomalies are very important in systems which have a strong spatio-temporal relationship between features, which is very much the case for exascale metrics.

IV.3 Event detection engine

The Event detection engine (EDE), as seen in Figure 1, has several sub-components which are based on lambda type architecture where we have a speed, batch and serving layer. Because of the heterogeneous nature of exascale systems and the substantial variety of solutions which could constitute a monitoring services the *data ingestion* component has to be able to contend with fetching data from a plethora of systems. Connectors will have to be implemented that serve as adapters for each solution. Furthermore, this component should also be able to load data directly from a file (HDF5, CSV or even raw format). This will aid in fine tuning of event and anomaly detection methods. We can also see that *data ingestion* can be done directly via query from the monitoring solution or streamed directly from the *queuing service* (after ETL if necessary). This ensures that we have the best chance of reducing the time between the event or anomaly happening and it being detected.

The *pre-processing* component is in charge of taking the raw data from the *data ingestion* component and apply several transformations. It handles data formatting (i.e. one-hot encoding), analysis (i.e. statistical information), splitter (i.e. splitting the data into training and validation sets) and finally augmentation (i.e. oversampling and undersampling). As an example the analysis and splitter are responsible for creating stratified shuffle split for K-fold cross validation for training while the augmentation step might involve under or oversampling techniques such as ADASYN or SMOTE[2]. This component is also responsible for any feature engineering of the incoming monitoring data.

The *training* component (batch layer) is used to instantiate and train methods that can be used for event and anomaly detection. The end user is able to *configure* the hyper-parameters of the selected models as well as run automatic optimization on these (i.e. Random Search, Bayesian search etc.). *Evaluation* of the created predictive model on a holdout set is also handled in this component. Current research and rankings of machine learning competitions show that creating an ensemble of different methods may yield statistically better results than single model predictions. Because of this ensembling capabilities have to be included. Finally, the trained and validated models have to be saved in such a way that enables them to be easily instantiated and used in a production environment. Several predictive model formats have to be supported, such as; PMML[2], ONNX[3], HDF5, JSON.

It is important to note at this time that the task of event and anomaly detection can be broadly split into two main types of machine learning tasks; classification and clustering. Classification methods such as Random Forest, Gradient Boosting, Decision Trees, Naive Bayes, Neural networks, Deep Neural Networks are widely use in the field of anomaly and event detection. While in the case of clustering we have methods such as IsolationForest, DBSCAN and Spectral Clustering.

[2] http://dmg.org/pmml/v4-1/GeneralStructure.html
[3] onnx.ai

Once a predictive model is trained and validated it is saved inside a *model repository*. Each saved model has to have metadata attached to it denoting its performance on the holdout set as well as other relevant information such as size, throughput etc. The *prediction* component (speed layer) is in charge of retrieving the predictive model form the *model repository* and feed metrics from the monitored exascale system. If and when an event or anomaly is detected EDE is responsible with signaling this to both the *Monitoring service* reporting component and to other tools such as the Resource manager and/or scheduler any decision support system. Figure 1 also shows the fact that the *prediction* component gets it's data from both the *monitoring service* via direct query or directly from the *queuing service* via the *data ingestion* component. For some situations a rule based approach is better suited. For these circumstances the *prediction* component has to include a rule based engine and a rule repository.

Naturally, detection of anomalies or any other events is of little practical significance if there is no way of handling them. There needs to be a component which once the event has been identified tries to resolve the underlying issues. Let us suppose that a particular job shows anomalous behaviour, now the system must decide is the job compromised beyond recovery? Is its internal state consistent can it be restarted from a checkpoint? If not, can it be rescheduled on a different node? Does the new node have the data required, if not what is the most effective way of coupling the data with the job to be executed? To solve these questions is by no means an easy task. One of the ways in which this could be solved is to use a Belief Desire Intention cognitive model where EDE represents the current beliefs of the exascale system state while desires and future intentions are formulated based on these.

V. Proposed Technology stack for EDE

V.1 Machine learning and data processing libraries

The choice of what underlying software framework to use for all of these components is a complicated one. For most machine learning tasks Python has become for many the language of choice. We have frameworks such as Scikit-learn, Keras, XGBoost, LightGBM, CatBoost, Pandas etc. which are widely used for these types of tasks provide Python bindings. Moreover, all other frameworks respect naming and API conventions from scikit-learn, making the processing pipeline needed for *training* and *prediction* much easier. Furthermore, all components listed for EDE are loosely coupled and can have a high cardinality making them an excellent choice for a distributed deployment. Moreover, in some situations it is advisable to have more than one component instance for each major functionality. These major functionalities are: data collection and formatting, training and validation.

The deployment can be based on Big Data technologies such as Spark in the case of *prediction* and *training* or even taking advantage of machine learning services such as ML Engine[4] by Google or Kubeflow[5]. The *data bus* has the role of passing messages and data between all of the EDE components. It can be implemented using technologies such as Apache Kafka or even a simpler solution based on RabbitMQ.

[4]https://cloud.google.com/ml-engine/

[5]https://github.com/kubeflow/kubeflow

V.2 Proof-of-concept prototype

We already have a close-related framework implemented for the H2020 DICE project [1]. The DICE anomaly detection tool [9] is also based on a lambda type architecture, having both bath and speed layers encapsulating training and prediction very similar to the envisioned architecture for EDE. However, DICE framework is of a much more limited scope, it's components being tightly coupled making a distributed deployment difficult, and also, wrapping a much more limited machine learning and pre-processing method selection. The main issue is the difficult scaling of the DICE tool, as data ingestion and pre-processing cannot be scaled separately. EDE solves this by relying on decoupling of ingestion and pre-processing components allowing selective scaling of these.

The next steps will be in a comprehensive overview of the type of metrics inherent in all exascale systems which can be used to create a coherent and global overview of not only of the system itself but on the jobs that run on them. Selection of machine learning methods to be used for event detection is another important step. Including the checking of different evaluation metrics in conjunction with state of the art machine learning methods.

VI. Conclusions

In this paper we have discussed the challenges related to exascale systems in particular related to event/anomaly detection. We show how most issues in these kinds of systems can be linked back to the need for automatic handling of certain events and functionalities which in turn requires a deep insight into the current state of not only the exascale system but the jobs running on it.

Our main contribution is the identification of challenges for exascale systems and the proposing of our event detection engine. We detailed not only the overall architecture of the system based on the challenges identified in this paper, but also provided a comprehensive list of technologies that can be used for a prototype implementation. A complete prototype implementation of the proposed EDE framework presented in this paper is expected to be available in the next two years.

VI.1 Future Work

Interoperability is a key aspect in the success of any new platform and this is no different in the case of EDE. One of our first steps will be to ensure that the data integration component can interface with as many monitoring platforms and underlying storage solutions as possible (i.e. Elasticsearc, Influxdb, MongoDB etc.). Also linked with interoperability is the necessity to provide a common format for model export (i.e. Omnx. PMML etc.). Because of the distributed nature of exascale systems EDE will have to support a distributed deployment scheme which enables the scheduling of concurrent tasks for both training and prediction. For distributed training we have several viable underlying platforms o which we can base EDE implementation, such as using Apache Spark, Apache AirFlow or even a custom solution based on asynchronous task queues such as Celery for Python. This will potentially aid in the hyperparameter optimization of detection algorithms by enabling concurrent deployment of candidate parameter configurations.

Furthermore, the selection of machine learning algorithms coupled with the evaluation functions used for training is a key issue

which we will have to tackle in order to ensure good predictive performance. A comprehensive study of state of the art algorithms and evaluation functions is needed. These will also include special attention to security and privacy related events and anomalies, a topic which is not sufficiently explored in current exascale systems.

VII. Acknowledge

This work has received funding from the EC-funded H2020 ASPIDE project (Agreement 801091). This work was supported with hardware resources by the Romanian grant BID (PN-III-P1-PFE-28).

References

[1] G. Casale, D. Ardagna, M. Artac, F. Barbier, E. D. Nitto, A. Henry, G. Iuhasz, C. Joubert, J. Merseguer, V. I. Munteanu, J. F. Pérez, D. Petcu, M. Rossi, C. Sheridan, I. Spais, and D. Vladušič. Dice: Quality-driven development of data-intensive cloud applications. In *Proceedings of the Seventh International Workshop on Modeling in Software Engineering*, MiSE '15, pages 78–83, Piscataway, NJ, USA, 2015. IEEE Press.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[3] G. Costa, T. Fahringer, J.-A. Rico-Gallego, I. Grasso, A. Hristov, H. Karatza, A. Lastovetsky, F. Marozzo, D. Petcu, G. Stavrinides, D. Talia, P. Trunfio, and H. Astsatryan. Exascale machines require new programming paradigms and runtimes. *Supercomputing Frontiers and Innovations*, 2(2):6–27, June 2015.

[4] S. Fiore, M. Bakhouya, and W. W. Smari. On the road to exascale: Advances in high performance computing and simulations—an overview and editorial. *Future Generation Computer Systems*, 82:450 – 458, 2018.

[5] K. Gai, K. R. Choo, M. Qiu, and L. Zhu. Privacy-preserving content-oriented wireless communication in internet-of-things. *IEEE Internet of Things Journal*, 5(4):3059–3067, Aug 2018.

[6] K. Gai and M. Qiu. Blend arithmetic operations on tensor-based fully homomorphic encryption over real numbers. *IEEE Transactions on Industrial Informatics*, 14(8):3590–3598, Aug 2018.

[7] K. Gai, M. Qiu, L. Tao, and Y. Zhu. Intrusion detection techniques for mobile cloud computing in heterogeneous 5g. *Sec. and Commun. Netw.*, 9(16):3049–3058, Nov. 2016.

[8] M. Gander, M. Felderer, B. Katt, A. Tolbaru, R. Breu, and A. Moschitti. Anomaly detection in the cloud: Detecting security incidents via machine learning. In A. Moschitti and B. Plank, editors, *Trustworthy Eternal Systems via Evolving Software, Data and Knowledge*, volume 379 of *Communications in Computer and Information Science*, pages 103–116. Springer Berlin Heidelberg, 2013.

[9] G. Iuhasz, I. Dragan, G. Casale, T. Ustinova, M. Bersani, and I. Torres. Quality anomaly detection and trace checking tools - final version. Technical report, H2020 DICE, available at http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/08/D4.4_Quality-anomaly-detection-and-trace-checking-tools-Final-version.pdf, 2017.

[10] T. Jakobs, J. Lang, G. Rünger, and P. Stöcker. Tuning linear algebra for energy efficiency on multicore machines by adapting the atlas library. *Future Generation Computer Systems*, 82:555 – 564, 2018.

[11] M. Katevenis, R. Ammendola, A. Biagioni, P. Cretaro, O. Frezza, F. L. Cicero, A. Lonardo, M. Martinelli, P. S. Paolucci, E. Pastorelli, F. Simula, P. Vicini, G. Taffoni, J. A. Pascual, J. Navaridas, M. Luján, J. Goodacre, B. Lietzow, A. Mouzakitis, N. Chrysos, M. Marazakis, P. Gorlani, S. Cozzini, G. P. Brandino, P. Koutsourakis, J. van Ruth, Y. Zhang, and M. Kersten. Next generation of exascale-class systems: Exanest project and the status of its interconnect and storage development. *Microprocessors and Microsystems*, 61:58 – 71, 2018.

[12] D. Kerbyson, A. Vishnu, K. Barker, and A. Hoisie. Codesign challenges for exascale systems: Performance, power, and reliability. *Computer*, 44(11):37–43, Nov. 2011.

[13] S. Matsuoka, H. Sato, O. Tatebe, M. Koibuchi, I. Fujiwara, S. Suzuki, M. Kakuta, T. Ishida, Y. Akiyama, T. Suzumura, K. Ueno, H. Kanezashi, and T. Miyoshi. Extreme big data ebd: Next generation big data infrastructure technologies towards yottabyte/year. *Supercomput. Front. Innov.: Int. J.*, 1(2):89–107, July 2014.

[14] S. Perarnau, J. A. Zounmevo, M. Dreher, B. C. V. Essen, R. Gioiosa, K. Iskra, M. B. Gokhale, K. Yoshii, and P. H. Beckman. Argo nodeos: Toward unified resource management for exascale. In *2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2017, Orlando, FL, USA, May 29 - June 2, 2017*, pages 153–162, 2017.

[15] D. Petcu, G. Iuhasz, D. Pop, D. Talia, J. Carretero, R. Prodan, T. Fahringer, I. Grasso, R. Doallo, M. Martin, B. B. Fraguela, R. Trobec, M. Depolli, F. Almeida-Rodriguez, F. Sande, G. Costa, J.-M. Pierson, S. Anastasiadis, A. Bartzokas, C. Lolis, P. Goncalves, F. Brito, and N. Brown. On processing extreme data. *Scalable Computing: Practice and Experience*, 16(4):467–489, Dec. 2015.

[16] X. Zhao and N. Jamali. Energy-aware resource allocation for multicores with per-core frequency scaling. *Journal of Internet Services and Applications*, 5(1):9, Sep 2014.