Towards Seamless Integration of Data Analytics into Existing HPC Infrastructures

Dennis Hoppe, Michael Gienger,
Thomas Bönisch, Oleksandr Shcherbakov
High Performance Computing Center Stuttgart
Stuttgart, Germany
e-mail: {hoppe, gienger, boenisch,
shcherbakov}@hlrs.de

Diana Moise
Cray Inc.
Basel, Switzerland
e-mail: dmoise@cray.com

***Abstract*—Customers of the High Performance Computing Center (HLRS) tend to execute more complex and data-driven applications, often resulting in large amounts of data of up to 1 Petabyte. The majority of our customers, however, is currently lacking the ability and knowledge to process this amount of data in a timely manner in order to extract meaningful information. We have therefore established a new project in order to support our users with the task of knowledge discovery by means of data analytics. We put the high performance data analytics system, a Cray Urika-GX, into operation to cope with this challenge. In this paper, we give an overview about our project and discuss immanent challenges in bridging the gap between HPC and data analytics in a production environment. The paper concludes with a case study about analyzing log files of a Cray XC40 to detect variations in system performance. We were able to identify successfully so-called aggressor jobs, which reduce significantly the performance of other simultaneously running jobs.**

Keywords—High Performance Computing; Big Data; Data Analytics; Cray XC; Urika GX; Log File Analysis

I. INTRODUCTION

The High Performance Computing Center (HLRS) is a research and service institution affiliated to the Information Centre of the University of Stuttgart. It is one of the three national supercomputing centers in Germany and one of the three members of the Gauss Centre for Supercomputing [1]. HLRS has strong ties to the industry, where the majority of industrial users stem from the engineering domain. HLRS conducts regularly academic workshops and industrial trainings. Both are targeted towards end-users with limited knowledge about HPC as well as domain experts. Topics covered include parallel programming (MPI), performance optimization and debugging, and introductions to programming languages for scientific computing such as Fortran and C++. Although not representative, these topics reveal that the HPC domain is geared towards maximizing performance by leveraging very specialized applications, tools, as well as low-level programming languages and paradigms.

The first supercomputers were introduced in the early 1960s, and thus current technology and software stacks are mature. Current HPC systems such as the Cray XC40 [2] enable end-users to perform not merely more complex simulations, but rather execute considerably more simulations in the same period of time. For example, executing and analyzing a few crash simulations is still easy to manage. However, with today's computing power (e.g. the XC40 system at HLRS has a peak performance of 7.4 PFlops), automotive companies tend to execute several hundred crash simulations; up to 1 Petabyte of result data can thus be generated within a single day.

Since it no longer becomes feasible that data is processed and analyzed manually by domain experts, customers of HLRS are very interested in solutions to process automatically large amounts of data. With respect to crash simulations, analyzing result data of each simulation in near real-time can help to adjust input parameters for future simulations. Automotive companies could reduce the number of simulations from saying 1,000 to 5,000 while leaving out unwanted simulations. That way, the runtime of jobs could be reduced, and customers save money. Although a movement exists to combine HPC with Big Data Analytics, we believe that very few people are already there: Current HPC systems are not designed to meet the demands of Big Data Analytics [3,4].

We have established in 2016 a new project that deals with the challenge of combining HPC with data analytics for both academia and industry. Our customers should be enabled to perform their simulations leveraging the power of HPC, and then perform seamlessly data analytics on result data. In an ideal scenario, discovered knowledge is fed back into applications running on the HPC infrastructure to adapt future executions. Since the installed HPC system at HLRS does not fully satisfy requirements listed above, we have put a high-performance data-analytics system, a Cray Urika-GX [5], into operation. In the first project phase, both HLRS systems, the Cray XC40 and the Urika-GX, are operated independently. Our vision is to combine both in a seamless manner. However, this means that we have to tackle various challenges including security aspects, data transfer, and accounting, to name but a few.

The remainder of this paper is organized as follows. Section 2 overviews predominant architectures in HPC and data analytics. Section 3 then highlights some of the main differences between HPC and performing data analytics. Section 4 continues to present the motivation for HLRS to

Figure 1. Hazel Hen, a Cray XC40 system, at HLRS premises.

emphasize on Big Data. Section 5 then discusses challenges and opportunities that arise when operating the Urika-GX in combination with an existing HPC infrastructure. Section 6 presents a first case study, which was done in collaboration with Cray. Log files of the HPC infrastructure are analyzed in order to identify performance variations. This paper concludes with an outlook.

II. BACKGROUND

This section briefly introduces the predominant architectures available for HPC and data analytics based on four layers: hardware, file system, resource managers, and programming model. We kindly refer readers with background knowledge in both domains to Section 3, which highlights main requirements of data analytics that are currently hardly satisfied by HPC architectures.

A. HPC Architectures

HPC is a well-established domain, in which everything from hardware to software is optimized for performance. It starts with the hardware, which is based on server-based components such as the Intel Xeon processor family, and interconnects that provide high-throughput and low-latency. A well-known interconnect is InfiniBand. It has a throughput of up to 10 times higher than a standard 10 Gigabit-Ethernet (10GigE) connection, and the latency if up to 10 times less than 10GigE. Figure 1 depicts Hazel Hen [6], HLRS' Cray XC40 system, which is composed of more than 7,700 compute nodes interconnected through Aries. Each diskless compute node has two sockets with an Intel Haswell processor, yielding nearly 200,000 cores. Furthermore, each compute node has 128 GB of random access memory (RAM) installed. Hazel Hen has a peak performance of 7.42 Petaflops, and is currently ranked 14 in the Top500 [7].

As already described, compute nodes are disk-less, and thus storage is available globally via a distributed parallel file system such as Lustre [8] or Network File System (NFS). The Lustre at HLRS provides about 10 PB of storage to its users. Since the majority of HPC applications are batch jobs, resource managers for job submission—such as SLURM [9] or TORQUE [10]—are optimized for large scale.

On the application layer, high performance programming languages, tools, and libraries are dominant: C/C++ and Fortran for the development of parallel applications using libraries such as MPI [11], OpenMP [12], and OpenCL [13]. MPI, for example, is a specification to allow for efficient communication in heterogeneous environments. It should be noted that developing for HPC requires a deep understanding and good knowledge about the underlying architectures, network topologies, programming environments and libraries. For instance, MPI-based applications require knowledge in communication concepts (point-to-point, one sided, collective, blocking vs non-blocking, …), declarative concepts (groups and topologies), and process management. In summary, it is not trivial to develop highly-efficient HPC applications from scratch. In addition, many HPC centers provide custom software, libraries and tools to manage jobs and storage. This often results in a vendor block, because customers cannot move seamlessly their jobs to other HPC centers. Thus, training courses and consultancy are often two activities offered by today's HPC centers to attract new customers with limited knowledge.

B. Big Data Architectures

Commodity hardware including x86 processors, Ethernet interconnects, and local disk-based storage solutions provide the basis for common data analytics clusters. In comparison to HPC, data analytics systems often have local storage to be used for the Hadoop Distributed File System (HDFS). HDFS is the dominant file system used in the domain of data analytics. The file system features data replication, fault tolerance, and streaming data access, to name but a few. HDFS is further optimized towards storing large file sizes of up to several terabytes, and thus it is the main building block of a data analytics system. On top of HDFS reside a vast amount of frameworks and tools including Apache Hadoop [14] and Spark [15]. Both frameworks allow for efficient batch processing of individual jobs. Whereas Hadoop is based strictly on the well-known MapReduce paradigm (intermediate data is always stored on disk), Spark allows for in-memory processing. As a consequence, Spark can process data sets that fit into memory significantly more efficient. The Apache project offers also various tools to work with data stored in HDFS on a high-level; tools include Hive and Pig, which allow for explorative data analysis. Finally, developers and data scientists can choose from a rich variety of applications and libraries. Software is developed mainly in Java, Python, or GNU R. Available frameworks such as Apache Mahout as well as Spark's MLlib, enable developers to easily perform both unsupervised and supervised machine learning tasks.

Table 1. Comparison between HPC and Big Data based on four layers.

Layer	***HPC***	***Big Data***
Programming Model	C/C++, Fortran MPI, OpenMP, OpenCL	Java Hadoop, Spark
Resource Manager	TORQUE, SLURM	YARN Mesos, Marathon
File System	Lustre, NFS	HDFS
Hardware	Server components (e.g. Intel Xeon, InfiniBand)	Commodity components (e.g. GigE)

Table 2. Technical details of the Urika-GX systems installed at HLRS.

	System 1 (Gilgamesch)	*System 2 (Enkidu)*
Nodes	48	16
Compute nodes	41	9
CPU	2x Intel BDW 18-Core, 2.1 GHz	
RAM	512 GB	
Local Storage	2 TB HDD; Intel DC P3608 SSD (1.6 TB)	
File System	Sonexion 900 with 240 TB 4.0 GB/s throughput	
Software	• YARN, Mesos, Marathon • Hadoop, Spark, Cray Graph Engine, GNU R • Apache Kafka, Cassandra • Apache Hive • …	

C. Hybrid Architectures

The trend goes towards merging high performance technologies such as InfiniBand with Hadoop clusters. This transition has already taken place with the Cray Urika-GX platform. The platform incorporates Intel Xeon processors as well as Cray's Aries interconnect, both hardware that is predominant in the HPC domain. HLRS have installed two systems with a total of 64 compute nodes; each equipped with 36-cores of Intel's current Xeon processor generation. Nodes are staffed with 512 GB RAM, as well as 1.6 terabytes of local SSD memory. Thus, the system is optimally designed to provide solutions for current and future challenges of data analytics. The platform has a state-of-the-art data analytics software stack installed. Moreover, the special-purpose Cray Graph Engine enhances the analysis of semantic data, which is commonly present in biology and chemistry [16]. Here, Cray offers an alternative to Spark's GraphX library.

III. When HPC Meets Big Data

HPC is best for demanding simulations that benefit from embarrassingly parallel computations; simulations typically generate and analyze data in-situ. As a consequence, the costly task of moving data around is well avoided. Big Data, however, introduces several new requirements [17–20].

A. Data Colocation

Data-intensive applications require to pre-process and read-in large amounts of data. Data movement itself is costly on current HPC architectures due to the immanent separation between computation and storage. For example, HLRS's Cray XC40 cluster has no local storage, and thus data is stored globally in the Lustre file system; moving data between nodes thus significantly depends on bandwidth and latency. Predominant architectures in data analytics, however, push the co-location of computation and data to be on the same node. Hence, data-intensive applications can benefit from data-locality; data movement in the cluster is minimized through optimal scheduling strategies that take locality into account: Required data is ideally already stored on the node, on which a job will be executed. Frameworks such as Apache Hadoop deal with data-locality through the prevailing distributed file system HDFS.

In order to incorporate data analytics with HPC, existing shared file systems such as Lustre need to support HDFS. Benefits of direct HDFS support are twofold. First, HPC centers can avoid to have separate storages for HPC (Lustre) and data analytics (HDFS). Second, cost-intensive data movements can be avoided. Recent research proposes solutions to provide Hadoop extensions for Lustre [21].

Furthermore, a performance gain can be achieved by integrating MPI with Hadoop MapReduce [17,22,23]. In [17], the author proposes a framework for Hadoop with OpenMPI. The optimized MapReduce implementation leverages MPI-IO for parallel data reads, and both map and reduce functions are implements as MPI processes. Finally, output data is also written out via MPI-IO.

B. Recurrent Analysis

A typical workflow (e.g., explorative analysis) or algorithm (e.g., clustering) in data analytics requires to process the same dataset several times, and thus will benefit significantly from data locality, too. In this context, Apache Spark can maintain data in-memory to allow for efficient parallel processing of distributed data with iterative algorithms.

Existing HPC systems such as HLRS' Cray XC40 have 128 GB RAM per node, and thus are limited compared to specialized data analytics hardware including the installed Cray Urika-GX at HLRS, which has 512 GB. Moreover, Spark allows to dynamically extend in-memory data to local storage such as SSDs, when RAM is not sufficient to hold the total amount of data. This mechanism, for example, is not support on current HPC architectures due to the lack of local storage. On the other hand, the Urika-GX system can handle this task with ease. The system also comes with a pre-installed and regularly updated industry-based software stack.

C. Everchanging Zoo of Software

The software stack in HPC is in general limited, but very optimized to perform embarrassingly-parallel tasks. The operating environment is often based on customized Linux derivatives (e.g., CentOS), and for development, languages such as FORTRAN, C, and C++ are supported. In order to allow for parallel programming, modules such as MPI and SHMEM are available. Accessing the system is done solely via the command line through remote access.

In comparison, the software stack in data analytics is manifold. The spectrum ranges from data ingestion (e.g., Apache Kafka, Apache Flume), storage (e.g., HDFS, HBase and Cassandra), processing (e.g., Apache Hadoop, Apache Spark), and querying (e.g., Apache Hive). All these individual tools are interconnected to form custom solutions. As a consequence, developers need to have the freedom to choose the best tool for their current problem. The

underlying infrastructure should therefore support the concepts of sandboxing and containerization.

D. Scheduling

Scheduling for HPC is optimized for large-scale batch jobs. Predominant resource managers are SLURM and TORQUE. In general, a standard FIFO scheduling strategy is used in combination with multiple job queues. Thus, a job is scheduled as soon as required compute resources are available. Ideally, one would like have at least three different queues: interactive, batch jobs, heterogeneous jobs. The difference between the last two queue types is that the latter requires a heterogeneous set of compute nodes (e.g., additional graphic nodes). Moreover, HPC centers can add another queue to prioritize specific jobs.

Well-established resource managers for Big Data are YARN and Mesos. YARN is specifically installed to manage Hadoop jobs, whereas Mesos allocates resources for all other jobs (e.g., Spark). In this respect, Mesos can be seen as a global resource manager for the whole infrastructure. As with SLURM, YARN is implemented for long-running batch jobs, and does not support interactive jobs. These jobs can be handled by Mesos, though. On a data analytics platform such as the Urika-GX, multiple resource managers are required: Hadoop jobs are served via YARN, Spark jobs are managed through Mesos, and the Cray Graph Engine uses internally SLURM. It should be mentioned, that a unified interface exists. Scheduling policies in YARN and Mesos are manifold. For example, the fair sharing policy is often used per default, meaning that all jobs are assigned an equal share of the available resources.

E. Services

Whereas on HPC infrastructures at most some monitoring tools collect information on job execution, performance and power consumption, a big variety of services and daemons exist while running data analytics tasks. It starts with resource managers such as YARN or Mesos, which require to have different services running on both managing and compute nodes. Each compute node, for example, has both a YARN node manager running as well as Mesos slave clients. Furthermore, each compute node provides access to a Web interfaces for retrieving status information. On the managing nodes, additional Web interfaces are installed to allow users to retrieve job-related information. In addition, the login nodes have more than ten services running including user interfaces for Zookeeper, Mesos, Spark, HDFS, Marathon, Hadoop, Hive, to name but a few. The Urika-GX platform is also shipped with the Hadoop User Experience (HUE) service, which is a Web UI that aggregates all individual services. Additional Web server start when individual jobs are triggered, e.g., a Spark launches another job monitoring server.

F. Sandboxing

Data scientists usually select a subset of the above mentioned tools and frameworks in order to build a customized solution that best tackles a given problem. Infrastructures can support this flexibility in daily work by offering containerization support. Containerization is a virtualization technique to encapsulate software in containers, that are running in the kernel space of the operating system. A prominent example are Docker containers. They are simple to describe and set-up, and thus can be easily shared and re-used.

The additional requirements imposed by Big Data can simply be satisfies through a dedicated infrastructure such as the Cray Urika-GX. It is a more challenging task when an existing HPC infrastructure should be used for data analytics, though. In [24], the authors report on their experience with executing Hadoop and Spark jobs on the Cray XC series. Whereas good performance could be reported for Spark, where the jobs could benefit from 32 GB RAM per compute node[1], the performance of Hadoop jobs was, in comparison, rather poor, because of the frequent read/write requests to the connected Lustre. In order to support all kinds of jobs with best performance, HLRS installed dedicated hardware.

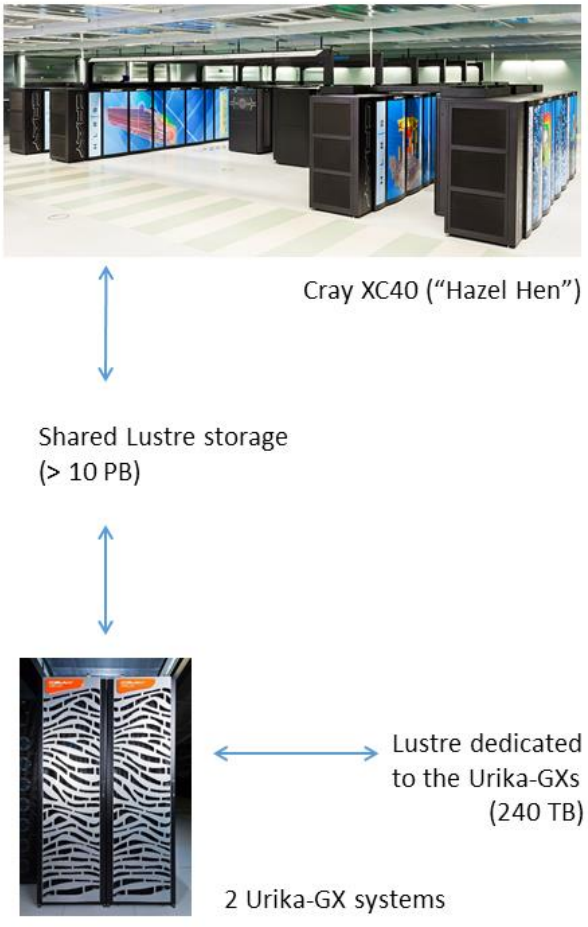

Figure 1. Part of the infrastructure at HLRS.

[1] Please note that Spark performs computation in-memory.

IV. BIG DATA ANALYTICS AT HLRS

HLRS and Cray have launched an ambitious project to deal with the challenges of combining HPC with Big Data analytics. Hazel Hen, the current HPC flag-ship system of HLRS, is extended with specific data analytics hardware designed by Cray—a Urika-GX. In the first phase of the project, the Urika-GX system is operated in two individual configurations: a larger configuration (48 nodes) for production and a smaller configuration (16 nodes) for development and testing. Each configuration comes with an industry-based Big Data software stack including Apache Hadoop, Apache Spark, and the Cray Graph Engine as a means to perform data analytics. By the end of the project, it planned to merge both individual configurations.

In the project, we are investigating the hardware of the Urika-GX and its usefulness for our customers. Since the majority of today's data analytics algorithms are oriented towards text processing (e.g., news clustering) and graph analysis (e.g., social network studies), we are further in need to evaluate these algorithms with respect to their applicability to the engineering domain. Thus, we will examine future concepts for both hardware and software. We will therefore pursue multiple case studies from divergent domains throughout the next three years.

The project's vision is to incorporate the Urika-GX with the existing HPC system at HLRS. This ambitious goal requires to tackle various operative challenges including security aspects, fast and secure data transfer, and accounting, to name but a few. The project will (a) research new methodologies and develop frameworks to achieve a seamless integration of both systems; (b) further pursue multiple case studies in order to evaluate the applicability of current technologies to current challenges in both academia and industry; (c) investigate data analytics in industrial areas.

Our first case study was conducted in collaboration with Cray. In the past, we have randomly observed performance variations of our Cray XC40 system, which is composed of more than 7,000 compute nodes. System administrators tried to identify the root cause for these performance drops by manually studying log files of the entire compute cluster. However, they were unable to extract meaningful insights from these log files that track individual jobs, including information about when and where jobs were executed. Although system health management of IT infrastructures is a well-established methodology to identify errors in software or faulty hardware since decades, it is often infeasible for experts to identify any causes due to the vast amount of semi-structured data. Thus, monitoring today's IT infrastructures has actually become a big data challenge on its own. When classifying log data into the five V's of big data (volume, velocity, veracity, variety, and value), we argue that value is key with respect to this case study. In current IT infrastructures, monitoring is performed, large amounts of log files are stored for long term preservation, but system administrators are lacking the required time or expertise to analyze the data accordingly, and to derive from the results consequences towards future management and usage of resources. Due to the volume and velocity of the data, we utilized the Cray Urika-GX system with its pre-installed data analytics software stack to process and analyze log files. Before Section 6 reports on the methodology and first results obtained with respect to the log file analysis, Section 5 first discusses challenges that arise when integrating a new data analytics system into existing infrastructure.

V. INTEGRATION OF HPC WITH BIG DATA ANALYTICS

Making the decision to incorporate an additional system into existing infrastructure is always challenging. We therefore discuss next some key tasks that came up while adding the two new Urika-GX systems to our portfolio.

1) Usage Model

Data analytics platforms per se are targeted for single-user usage and for small teams that work closely together. Clusters are often operated in-house. In these cases, strict user and group permissions as well as access rights for jobs and data have not the highest priority. From an end user point of view, it is acceptable that other users from the same team can see others jobs and data. Users may have even root permissions to install additional software. However, coming from the HPC domain, security plays a significant role in order to maintain system health and data security. At HLRS, we aim to use the Urika-GX system in a multi-user scenario, and thus user and group permissions have to be aligned across the system including the file system. For example, users need to be prevented from having access to other user's data in HDFS.

2) Software

Although the Urika-GX is shipped with a fundamental set of pre-configured software for performing data analytics, future end users might request additional software to be installed on the system. Software components could, for example, include Google's TensorFlow or extra databases such as the well-established NoSQL database MongoDB. Critical here is that new software components are often required to be installed for distributed usage, and they also need to be incorporated into existing software like Spark. TensorFlow on the Urika-GX, for instance, is started through a specific Spark kernel which can be loaded interactively and used in Jupyter notebooks. These are just two examples for software that is requested by customers. In order to satisfy most end users and not limit one selves to a handful of software components, we need to implement a practicable solution for future software installations. Based on our experience with HPC, a future model could be composed of three layers: a) users are allowed to install components locally, b) Cray maintains a global software repository that includes supported software such as TensorFlow, and which is then accessible by all users, and c) HLRS will also maintain another repository of not supported software components, which is accessible by selected users.

3) Security

Following the multi-user model, security requirements on the Urika-GX systems need to satisfy the high security requirements that HLRS has already implemented on their HPC systems. This means that the Urika-GX is not directly connect to the World Wide Web, and that per default Web server have to be deactivated. As a consequence, some services mentioned in the previous sections are not available for end users. Furthermore, the Urika-GX systems have to undergo a security audit. When the audit is passed, the systems can be connected with the existing Lustre of HLRS' Hazel Hen to allow for a seamless data transfer between the HPC and Big Data infrastructures, as well as to HLRS' LDAP server to manage user accounts globally. When operating the Urika-GX independently, one can rely on the local LDAP server pre-installed on the Urika system. However, we would like to mount the shared Linux home for each user, and thus we will need access to the global LDAP.

4) Accounting

Performing proper user accounting on HPC is trivial: monitor per user the number of compute nodes used as well as the overall wall-clock time. Pricing is then done per node hour. While driving a multi-user model on the Urika-GX systems, accounting is not as straightforward. Users can allocate resources not just through a single resource manager, but rather via three different managers. Each resource manager such as Mesos has a different API and might not log the required information including cores allocated and HDFS/Lustre storage used. Mesos also maintains job information based on a global user, and thus a job is not directly associated with the user that started the job. Moreover, the same resources might be used simultaneously by different users, which subsequently, might even result in performance loss. An adequate solution might be to establish an independent resource monitoring tool that collects the required information as mentioned above. That way, the solution striven for is then independent of the resource managers installed on the system.

5) Data Ingestion and Storage

Since the Urika-GX systems are not directly connected to the Internet, the system cannot directly support applications or services that perform real-time analytics based on streaming data. Sources for streaming data could be social networks, for example. However, the focus of HLRS is on the engineering domain, where streaming data is not as prevalent as in other research domains. The lack of direct Internet access for data ingestion can be solved by an extra proxy node, on which specific services and ports are activated. More important for HLRS is the connection to the existing Lustre storage, so that data produced by applications on the HPC cluster can be transferred to the Urika-GX own Lustre storage or directly into HDFS.

VI. Case Study: Log File Analysis

Performance variability on HPC platforms is a critical issue with serious implications on the users: irregular runtimes prevent users from correctly assessing performance and from efficiently planning allocated machine time. The hundreds of applications, which are sharing thousands of resources concurrently, escalate the complexity of identifying the causes of runtime variations. On production systems, implementing trial-and-error approaches is practically impossible. On that account, making use of existing information represents a preferable path to solution. Cray systems collect large amounts of data related to user applications, data that can be highly valuable for understanding performance variability. Novel analytics tools enable the exploring of ways to use the data for identifying and understanding performance variability. In this context, we have developed a Spark-based tool for analyzing system logs with the goal of identifying applications that show high variability (*victims*) and applications potentially causing the variability (*aggressors*). Understanding the nature of both types of applications is crucial to developing a solution to these issues. This section goes through the different steps of the analysis (data aggregation and filtering, victim and aggressor detection, validation) and the configuration parameters that help refine the search space. The analysis was implemented using Spark's RDD and DataFrame API and was tested on data coming from production systems.

A. Dataset

In practice, detecting application interference is challenging due to the very large number of jobs running on the system at any given time. On a system like Hazelhen there are typically hundreds of jobs running at any time. Cray systems collect large amounts of data related to user applications, which can be valuable for understanding performance variability. This data is collected on the Cray System Management Workstation (SMW). A promising approach to identify and understand performance variability is to use analytics tools to explore this data. In this context, we have used Spark to develop tools for data analysis with the goal of identifying victim and aggressor applications.

On the HLRS system, we have worked with a dataset extracted from SMW log files. This dataset consists of system log files that contain relevant information regarding all the applications running on the machine.

B. Analysis

Spark-based analysis of SMW data consists of three main steps: data extraction and filtering, victim detection, aggressor detection and validation. The main goal of the Spark-based tools is to reduce the search space for aggressors. To this end, we have defined configuration parameters that help refine the results of the analysis. The main steps of the analysis are detailed in the following.

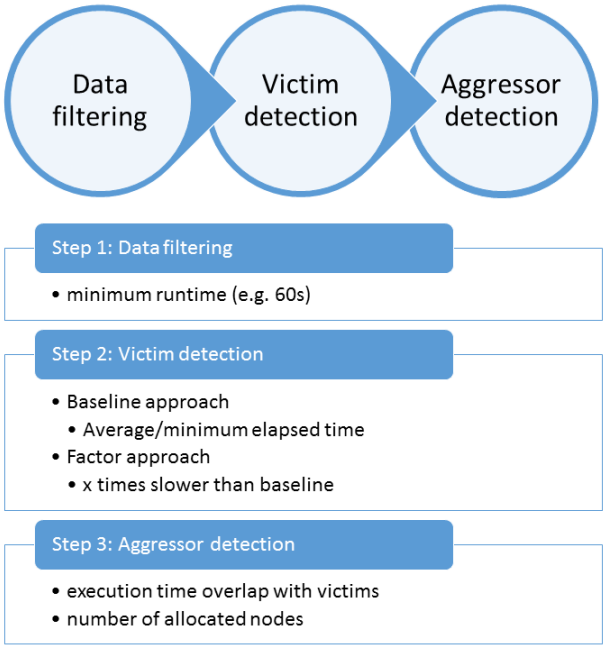

Figure 2. Three steps of the analysis workflow (data filtering, victim detection, and aggressor detection) including configuration parameters.

1) Data extraction and filtering

The SMW log files from the Cray XC40 were pre-processed at the customer site to make the details anonymous. This was done using python scripts that were developed by Cray. It would have been possible to do this step using Spark, if that had been available on site. Next, the files extracted from the above process were passed to Cray and loaded as an entire dataset onto a Urika-GX platform. The data contained information on each application running on the Cray XC40 system: job start time, end time, elapsed time, command that was used to run the application and the nodes used.

Finally, the dataset required filtering to keep only the useful, correct information (e.g.: remove bad lines, discard lines with incomplete commands, disregard application runs that are too short probably because of errors, etc.).

2) Victim detection

Applications that show high variability in runtime over several runs with identical setup are labeled as victims. First, we determine the baseline for each application as either the average or the minimum elapsed time over all the runs of each application. Second, we detect all the runs that were slower than the baseline by a certain (configurable) factor. The victim set contains these slow runs for all the applications.

3) Aggressor detection

Based on the victims set, we include in the potential aggressor set, all the applications running at the same time as the victims. Applications (sharing the same command) that are found more often are promoted to the top of the list. Further refining can be done based on the size of the aggressor (number of nodes it was using) and the time overlap of the aggressor with the victims.

Parameters introduced at each step of the analysis act as leverage mechanism for reducing the search space. Figure 2 summarizes the parameters available in the workflow.

4) Implementation and results

The three parts of the analysis were implemented as a single Spark application using the core RDD and DataFrame API. The initial dataset extracted from the SMW logs is processed by several Spark workers by loading the data into memory and performing the analysis described above. All the processing takes place in memory, only the final results are written to disk (Lustre).

While the filtering of the data and the victim detection steps do not require significant resources (CPU and memory), the last step of finding aggressors does need considerable computation power. For each victim, Spark starts a thread to search through the application set and get all the applications running at the same time as the victim. Considering the victim set can be quite large in number (in the order of thousands, reaching tens of thousands), we have used Spark's FutureRDD API to run these threads in parallel on the Urika-GX.

We have run the analysis on an initial dataset reflecting HLRS applications spanning two weeks. The factor used to classify victims was set at two times the average runtime and the factor used to classify aggressors was set to more than 1,000 allocated nodes. This analysis—with the factors set as above—identified 472 victims, and 2,892 potential aggressors. Seven of those potential aggressors were running on more than 1,000 nodes and three of them were found repeatedly.

On a larger dataset (SMW data over three months), the same analysis found 3,215 victims, 67,908 aggressors and 17 of them using more than 1,000 nodes. The analysis took 268 seconds on 300 cores on the Urika-GX.

VII. Conclusions and Outlook

Our vision is to integrate data analytics with HPC in order to support our customers in analyzing the increasing amount of results produced by data-intensive applications. We therefore put two Urika-GX platforms into operation to extend our current HPC clusters. In this paper, we reported on current challenges that arise due to different requirements in data analytics (e.g., data colocation, recurrent analysis, and scheduling) as well as due to hurdles to be tackled while incorporating a new system into existing infrastructure (e.g., usage model, security, data storage, and accounting). We also highlighted one of our first use cases about a log file analysis in order to detect jobs that cause performance loss on our HPC cluster. Here, we successfully demonstrated that dividing jobs into aggressors and victims can help to identify a subset of jobs that affect the runtime of simultaneous running jobs significantly.

Whereas, we have presented a so-called offline scenario for the log analysis, we aim in the future to implement an online scenario, where log data is streamed directly to the Urika-GX system to identify performance variations on the

HPC cluster in near real-time. Furthermore, future work will also focus on enhancing the analytics part to account for and identify faulty hardware in advance using anomaly detection. System administrators deal currently with a so-called post-mortem scenario, where the error or hardware fault has already happened. Since a system could then be non-functional for several hours, post-mortem scenarios should be avoided by offering a sophisticated means to predict hardware failures.

ACKNOWLEDGMENT

The project is funded by the State of Baden-Württemberg, Ministry of Science, Research and the Arts Baden-Württemberg. Cray Inc. is partner in the project.

REFERENCES

[1] Gauss Centre for Supercomputing e.V., "About GCS," http://www.gauss-centre.eu/gauss-centre/EN/AboutGCS/aboutGCS_node.html.

[2] Cray Inc., "Cray XC Series Brochure," http://www.cray.com/sites/default/files/Cray-XC-Series-Brochure.pdf.

[3] T. C. Chiang, "Can HPC and Big-Data Analytics co-exist?" 10/20/2016, http://comcen.nus.edu.sg/technus/hpc/can-hpc-big-data-analytics-co-exist/.

[4] A. Jackson, "Big Data: What's wrong with HPC?" 9/3/2015, https://www.epcc.ed.ac.uk/blog/2015/08/27/big-data-whats-wrong-hpc.

[5] Cray Inc., "Cray Urika-GX Product Brochure," http://www.cray.com/sites/default/files/Cray-Urika-GX-Product-Brochure.pdf.

[6] High Performance Computing Center Stuttgart, "Systems: Cray XC40 (Hazel Hen)," http://www.hlrs.de/systems/cray-xc40-hazel-hen/.

[7] TOP500, "Top500 List - November 2016," https://www.top500.org/list/2016/11/?page=1.

[8] Seagate Technology LLC, "Lustre Filesystem," http://lustre.org/.

[9] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*, G. Goos, J. Hartmanis, J. van Leeuwen et al., Eds., vol. 2862, pp. 44–60, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[10] G. Staples, "TORQUE---TORQUE resource manager," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing - SC '06*, B. Horner-Miller, Ed., p. 8, ACM Press, New York, New York, USA, 2006.

[11] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable parallel programming with the message-passing interface*, MIT Press, Cambridge, Mass., 1999.

[12] L. Dagum and R. Menon, "OpenMP: An industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[13] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–72, 2010.

[14] T. White, *Hadoop: The Definitive Guide / Tom White*, O'Reilly, Farnham, 2012.

[15] The Apache Software Foundation, "Apache Spark - Lightning-Fast Cluster Computing," http://spark.apache.org/.

[16] C. Joslyn, B. Adolf, S. al-Saffar et al., "High Performance Descriptive Semantic Analysis of Semantic Graph Databases," in *1st Workshop on High-Performance Computing for the Semantic Web at ESWC*.

[17] A. Cheptsov, "HPC in Big Data Age: An Evaluation Report for Java-Based Data-Intensive Applications Implemented with Hadoop and OpenMPI," pp. 175–180.

[18] N. Malitsky, "Bringing the HPC reconstruction algorithms to Big Data platforms," in *Bringing the HPC reconstruction algorithms to Big Data platforms*, pp. 1–8, IEEE, 2016.

[19] S. Singh, N. Narayan, and G. Raj, "Survey on Data Processing and Scheduling in Hadoop," *International Journal of Computer Applications*, vol. 119, no. 22, pp. 27–30, 2015.

[20] P. Xuan, J. Denton, P. K. Srimani et al., "Big data analytics on traditional HPC infrastructure using two-level storage," in *Proceedings of the 2015 International Workshop on Data-Intensive Scalable Computing Systems - DISCS '15*, P. C. Roth, Ed., pp. 1–8, ACM Press, New York, New York, USA, 2015.

[21] O. Kulkarni and D. Ferber, "Hadoop MapReduce over Lustre," http://cdn.opensfs.org/wp-content/uploads/2014/10/8-Hadoop_on_lustre-CLUG2014.pdf.

[22] X. Lu, F. Liang, B. Wang et al., "DataMPI: Extending MPI to Hadoop-Like Big Data Computing," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pp. 829–838, IEEE, 2014.

[23] S. J. Plimpton and K. D. Devine, "MapReduce in MPI for Large-scale graph algorithms," *Parallel Computing*, vol. 37, no. 9, pp. 610–632, 2011.

[24] R. Schmidtke, G. Laubender, and T. Steinke, "Big Data Analytics on Cray XC Series DataWarp using Hadoop, Spark and Flink," in *CUG Proceedings*, 2016.