Project no. 801091

ASPIDE

Research & Innovation Action (RIA)
EXASCALE PROGRAMING MODELS FOR EXTREME DATA PROCESSING

Low overhead software instrumentation and monitoring
D3.1

Due date of deliverable: 14th March 2019

Start date of project: June 15^{th}, 2018

Type: Deliverable
WP number: WP3

Responsible institution: Klagenfurt University
Editor and editor's address: Dragi Kimovski and Vladislav Kashansky, Klagenfurt University

Version 2.2

Project co-funded by the European Commission within the Horizon 2020 Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision History

Revision	Date	Author(s)	Description
1.0	29/10/18	Dragi Kimovski (UNI-KLU), Vladislav Kashansky (UNI-KLU)	Initial version.
1.1	29/10/18	Dragi Kimovski (UNI-KLU)	Initial version.
1.2	22/01/19	Vladislav Kashansky (UNI-KLU)	More detailed description of the model. Restructuring the model and model's constraints.
1.3	13/02/19	Dragi Kimovski (UNI-KLU)	Background and requirements analysis sections added.
1.4	14/02/19	Gabriel Iuhasz (IEAT), Ioan Dragan (IEAT)	Addded initial SOTA for Big Data Monitoring.
1.5	15/02/19	Vladislav Kashansky (UNI-KLU)	Restructuring the model's constraints.
1.6	19/02/19	Gabriel Iuhasz (IEAT)	Comprehensive restructuring of SOTA for Big Data Monitoring section.
1.6.1	20/02/19	Vladislav Kashansky (UNI-KLU)	Small fixes in the constraints.
1.7	21/02/19	Gael Goret (BULL), Lionel Vincent (BULL)	Add a part related to the monitoring and profiling IOI tool.
1.8	25/02/19	Gabriel Iuhasz (IEAT)	Added aditional BD related paragraphs to Chapter 5.
1.8.1	04/03/19	Gabriel Iuhasz (IEAT)	Added DICE Monitoring architecture figure.
1.9	06/03/19	Vladislav Kashansky (UNI-KLU)	Major model revision, update of the bibliography and related work sections.
2.0	06/03/19	Dragi Kimovski (UNI-KLU)	Final revision and integration.
2.1	08/03/19	Javier Garcia-Blas (UC3M)	Adding management figure.

2.2	11/03/19	David E. Singh (UC3M)	Adding some references of monitoring-scheduling co-ordination.

Main Contributors

David E. Singh (UC3M)
Dragi Kimovski (UNI-KLU)
Gabriel Iuhasz (IEAT)
Gael Goret (BULL)
Ioan Dragan (IEAT)
Javier Garcia-Blas (UC3M)
Lionel Vincent (BULL)
Vladislav Kashansky (UNI-KLU)

Executive Summary

Deliverable 3.1 is intended to serve as an scientific summary of the research activities conducted within the first half of Task 3.1. More concretely, the deliverable describes multiple concepts pertaining to the development of a general monitoring model capable of supporting exascale architectures. Therefore, the initial activities conducted within Task 3.1 included requirements analysis for the monitoring system in relation to the use-cases. Consequently, the results from the requirements analysis were used to define the monitoring approach, which has been purposely tuned for data-intensive applications, thus offering fine-granular performance information with low-overhead at the exascale level. The approach defined during the modeling stage, will later be developed as a monitoring tool and will be integrated within the exascale monitoring system as defined in Task 3.2.

The placement of the deliverable is outlined in Figure 1.

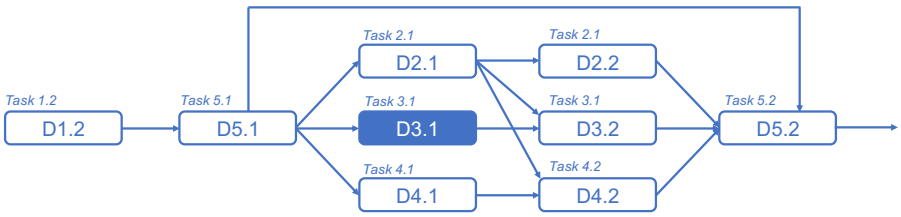

Figure 1: Deliverable dependencies of D3.1.

Contents

Chapter 1

Introduction

Deliverable 3.1 provides detailed information on the researched instrumentation, monitoring and measurement techniques related to Task 3.1. During the first half of the task duration the main goal is to define a monitoring model purposely tuned for data-intensive applications able to offer fine-granular performance information with low-overhead at the Exascale level. Later, within the second period of the task, it is intended for the developed model to be developed as a tool and integrated within the exascale monitoring architecture. The model was developed through a carefully requirements analysis of the use-case and with tight collaboration with WP2 and WP4. The instrumentation monitoring model enables trade-offs between measurement perturbation, measurement data accuracy, and monitoring response time. Furthermore, the model explored within this task, clusters the exascale system into multiple areas, and within each area finds multiple monitoring aggregation and analyis points, thus guaranteeing scalability and low-response time of the whole exascale system.

The monitoring model and the software that later will be developed will allow resource and application profiling. The monitoring data will be utilised to build resource and service-level profiles. The solution will be designed to handle large numbers of numeric time-series data, like dozens of performance metrics from thousands of servers. To collect data independently of the underlying software, the types of sensors and the implementation details, an additional abstraction layer will be created between them and a specific distributed database.

Chapter 2

Background

This section provides brief introduction on the algorithms, concepts and technologies utilized for the implementation of the monitoring aggregation tool for exascale computing systems.

2.1 Optimization

In general, optimization is a process of identifying one or multiple solutions, which correspond to the extreme values of two or more objective functions within given constraints set. In the cases in which the optimization task utilizes only a single objective function it results in a single optimal solution. Moreover, the optimization can also consider multiple conflicting objectives simultaneously. In those circumstances, the process will usually result in a set of optimal trade-off solutions, so-called Pareto solutions. The task of finding the optimal set of Pareto solutions is known in the literature as a multi-objective optimization.

The multi-objective optimization problem usually involves two or more objective functions which have to be either minimized or maximized. The problem of optimization can be formulated as: $min/max(f_1(Y), f_2(Y), \ldots, f_n(Y))$, where $n \geq 2$ is the number of objectives functions f that we want to minimize or maximize, while $Y = (y_1, y_2, \ldots, y_k)$ is a region enclosing the set of feasible decision vectors, which contain the so-called decision variables.

Even though the above formulation of the multi-objective optimization is without any constraints, this is hardly the case when real-life optimization problems are being considered. The real-life problems are typically constrained by some bounds, which divide the search space into two regions, namely feasible and infeasible region.

2.1.1 Linear integer programming programming

An integer programming problem is a mathematical approach for optimization in which part or all of the decision variables are restricted to be integers. In general,

for the purposes of industry and scientific research, the term refers to integer linear programming (ILP), in which the objective function and the constraints are linear. It is important to be noted that these constrains are applied in line with the integer values limitations.

In the mathematical theory it has been proved that Integer programming is a NP-complete. These complexity applies in particular for the special case of 0-1 integer linear programming, in which unknowns are binary.

Chapter 3

Related work

3.1 General-purpose Monitoring Technologies

Nagios[1] 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 the monitoring data. However, it is possible to create a hierarchy of Nagios servers that mitigate the disadvantages of a centralized server.

OpenNebula[2] can be viewed as a solution for monitoring and management of data-centers. The tool is designed to work over SSH in order to connect to all of the monitored machines. As advertised it was mainly designed to monitor physical infrastructures and also private cloud deployments.

There are also monitoring and management solutions customized for various instances. For example *OpsCenter*[3] provides a dedicated tool that can be used to monitor and administrate Cassandra. Another relevant tool that can be used in monitoring both Cassandra and do administration work on the nodes in the cluster is *Applications Manager*[4]. Application Manager is not bound to only monitoring Cassandra and provides ways of monitoring also NoSQL DB systems as MongoDB and others. When speaking strictly about database deployment we have a tool developed inside MongoDB *MMS*[5] that runs as a service and handles the inner monitoring of the application.

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

[1] https://www.nagios.org/
[2] http://opennebula.org/
[3] http://www.datastax.com/what-we-offer/products-services/datastax-opscenter
[4] https://www.manageengine.com/
[5] https://mms.mongodb.com/
[6] http://ganglia.info

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 globe.

3.2 Monitoring data intensive applications in Big Data Frameworks

Big Data tools and deployments are used in various use-cases starting from scientific workloads to analysis of customer data by corporations and SMEs. This uptake of these type of technologies have resulted in the creation of several tools and services which are geared towards not only setting up of these platforms but also of optimizing and tuning a particular deployment or data intensive application. One common thread in all solutions is that they require a comprehensive and global overview of the current state of both the underlying framework and application. In short monitoring is a key aspect in the age of big data.

In general when monitoring big data platforms on cloud based deployments a cross layer monitoring scheme is employed. This is due to the fact that components of individual applications can be distributed on various cloud layers as well as on multiple Virtual Machines (VMs). For this purpose the monitored parameters should be transmitted across all the cloud layers used by the applications.

By doing this we enable the creation of a complete overview of the running data intensive application at any given moment during execution. Currently most data inteinsive applications are running on SaaS (Software as a Service), Pass (Platform as a Service) and/ro IaaS (Infrastructure as a Service). In [15] the problems of monitoring public clouds are identified and an architecture for a monitoring platform is proposed.

In the case of a IaaS deployment typically we would like to monitor system metrics which include but are not limited to; CPU usage together with its states, memory usage and state, storage utilization, finally we have network related metrics.

PaaS and SaaS level metrics are focused on more abstract or complex metrics such as, byte throughput metrics, status of system services, uptime, availability etc. A good example of this is a Hadoop/YARN deployment where a we have metrics such as MapReduce processing time, Jon Turnaround, Suffle operation etc.

When trying to optimize the performance of data intensive applications a complete overview of not only current bu historical metrics is key. The act of parameter tuning is made more difficult in the case of missing metrics. For example it is very difficult or even impossible to tune an application that runs on a big data platform by only having access to system metrics. In the case of job and reducer task scheduling [4] monitoring data is crucial. A dynamic meta-scheduling architecture model for large scale distributed systems based on monitoring has been described in [23]. In this work, MonALISA service is used in combination with ApMON to collect customized information to provide automated decisions for improving the task scheduling. A distributed resource monitoring and prediction architecture

was presented in [25] allowing to find the best set of machines to run an application based on the collected information and the result of a prediction algorithm, which evaluate the potential performance of a node. Co-scheduling of CPU and memory intensive applications in the same node using monitoring information has been proposed in [11] to improve energy efficiency and overall throughput of a supercomputer. Another approach can be seen in [18] where minimalist monitoring is used. A solution for monitoring wireless devices is proposed WiGriMMA and presented in [10].

The importance of which monitoring metric is required for a comprehensive performance analysis is also tightly linked with what type of workload a data intensive application has to execute. In the case of video streaming application data transfer rate and quality are crucial while in the case of batch processing workloads basic process metrics and task execution times.

Monitoring of applications is not a requirement unique for data intensive applications. Several monitoring solutions have been developed for both HPC and Big Data platforms. All solutions can be split into several types based on the underlying architecture.

Most Big Data platforms have a built in metrics subsystem which can give great insight into the current state and performance of the system. In practical terms we can consider that all major Big Data platforms are instrumented. It is important to note that these subsystems are geared towards reporting not for storing and analysis of metrics. A dedicated datastore and analytically framework is still needed, in short a monitoring solution.

There are several types of monitoring solutions currently in use or in development. For centralized monitoring, all resource states and metrics are sent to a centralized monitoring server. These metrics are continuously pulled from each monitored component. This approach allows for a more controlled management and data access while potentially sacrificing availability, scalability and elasticity. It introduces a single point of failure while at the same time eliminating the possibility of horizontal scaling. Vertical scaling is only possible until a certain point, it is relatively easy for high network traffic volume to create a bottleneck which in turn can lead to packet drops and incomplete or corrupted monitoring data.

Decentralized architectures are designed to address most of the problems related to scaling and elasticity of centralized architectures. In these types of architectures no one component should be a single point of failure. All components should be loosely coupled which will help both with scaling and faliure recovery. In structured Peer-to-Peer systems the central authority is defused thus eliminating the central point of failure. Unstructured Peer-to-Peer network overlay is meant to be distributed, however, the search directory is not centralized. Besides the aforementioned we also have the hybrid Peer-to-Peer systems in which *super peers* can serve as localized search hubs for small network portions [3].

In [16] some of the most used monitoring tools and platforms for cloud computing and big data are presented. Part of these platforms have been adopted from High-performance computing (HPC) scenarios while others have been designed

specifically for this task.

3.2.1 Monitoring tools and platforms

Application Performance tools

NewRelic[7] provides a solution for monitoring both the infrastructure in a traditional, hybrid or cloud setup as well as the applications running on them. It is a serverless solution that is designed to handle a large variety of tasks based on code instrumentation. NewRelic defines code as both the application which is running as well as the VM on which it is running. At first this approach might seam strange however, by doing this it enables the easy identification of observable effects of the deployment of new features on the existing infrastructure.

Honeycomb[8] is an event driven monitoring solution intended for debugging of systems, databases and applications. Aggregation is done on the fly during data read which in theory enables fast analytics over big datasets without worrying about the underlying schemas and indexes. It also features integration with messaging systems such as slack which helps with collaboration. Also, every member of a development team is able to explore the system without superuser rights.

DataDog[9] is a full stack monitoring solution. It can be used to handle IaaS, PaaS as well as application monitoring. Furthermore, it also provides log management capabilities. It is designed as a turn-key solution for aggregation of metrics and events.

Hadoop Performance Monitoring UI [28] is designed as a built-in solution for finding bottlenecks in Hadoop set-ups as well as providing visual representation of the available tunable parameters for better performance of Hadoop. In essence, one can view the tool as being a lightweight monitoring UI for Hadoop servers. The fact that it is built-in the Hadoop ecosystem and easy to use, one can consider this solution for minor system tuning. Although built-in, it lacks performance, a good example where it laggs can be considered the time spent in garbage collection by each of the tasks.

SequenceIQ[10] is yet another solution for monitoring Hadoop clusters. The architecture proposed by SequenceIQ[11] and used in order to do monitoring is based on the ELK stack, that is, Elasticsearch[12], Logstash[13] and Kibana[14].

SequenceIQ uses an architecture based on Docker containers in order to obtain a clear separation between the Hadoop deployment and the monitoring tools. In a nutshell the monitoring solution consist of client and server containers. The server

[7] https://newrelic.com/
[8] https://www.honeycomb.io/
[9] https://www.datadoghq.com/
[10] http://sequenceiq.com/
[11] http://blog.sequenceiq.com/blog/2014/10/07/hadoop-monitoring/
[12] https://www.elastic.co
[13] http://logstash.net
[14] https://www.elastic.com/producs/kibana

container takes care of actual monitoring tools. In this particular deployment Kibana is used for visualization and Elasticsearch for consolidation of the monitoring metrics. Through the capabilities of Elasticsearch one can horizontally scale and cluster multiple monitoring components. The client container contains the actual deployment of the tools that have to be monitored. This instance contains Logstash, Hadoop and the collectd modules. Logstash connects to the Elasticsearch cluster as a client and stores the processed and transformed metrics data there. Collectd [1] is a daemon which collects system (and application) performance metrics and provides mechanisms to store the values in a different ways.

Due to its design the proposed solution is created by using multiple tools that are used in order to monitor metrics from different layers [17]. Because of containerization one can easily add or remove components from the system without affecting the overall monitoring system. Also, interrogating the system for various informations becomes an easy task to perform.

Hadoop Vaidya[15] (Vaidya in Sanskrit language means "one who knows", or "a physician") is a rule based performance diagnostic tool for MapReduce jobs. The mechanism behind Vaidya performs post analysis steps for map-reduce jobs. In order to achieve this goal it parses various execution statistics, or configuration files, from job history and stores them for later interrogation and usage.

Finding the performance problems that arise in the execution steps is done by application of individual rules implemented by Vaidya on the stored job execution statistics. Due to its design, rules are applied one by one and only known problems might be detected. By application of the rules, this system is able to provide users with advice for future executions such that some of the already occurring problems to be avoided. The output produces is an XML report based on the evaluation of individual test rules.

Apache Chukwa[16] is an open source data collection system for monitoring large distributed systems. At its core, Chukwa uses HDFS system and the Map/Reduce framework. By doing so, it provides mechanisms for easily scale. Although it is designed to collect the monitored data, it provides the users with a toolkit able to better understand the collected data. That is, it is capable of analyzing and display the results of different runs of the monitored software. Unlike other solutions, it is released under Apache license.

D-Mon, see Figure 3.1, is a monitoring tool developed during the H2020 DICE research projects. It was developed for the monitoring of data intensive applications during the development phase. 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

[15] http://hadoop.apache.org/docs/r1.2.1/vaidya.html
[16] http://chukwa.apache.org

also supports some data pre-processing steps during interrogation. Furthermore, it also supports all major big data technologies; HDFS, YARN. Spark, HBase, Storm, Flink, MongoDB, Cassandra, Solr, Elasticsearch.

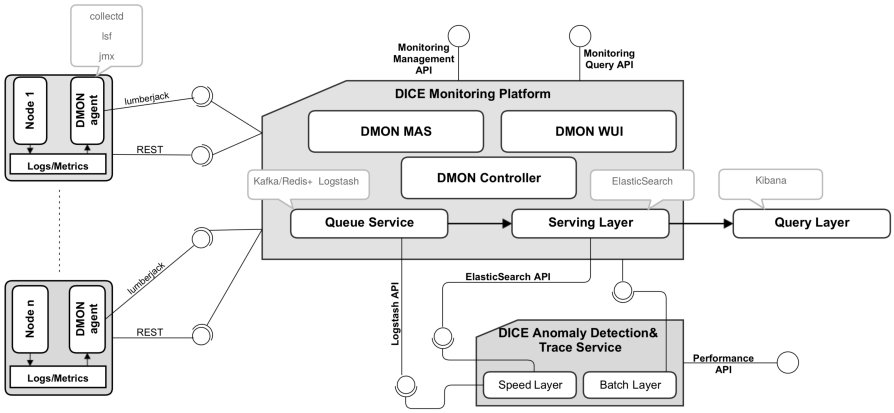

Figure 3.1: D-Mon Architecture overview.

Most monitoring solutions presented up until this point can be split up into two different categories. The first category are solutions geared towards monitoring of distributed applications and/or platforms such as web applications. You can use these types of monitoring tools for Big Data framework based applications it will require substantial configuration or modifications. Even more problematic is the fact that most tools don't support the full software stack of the data intensive application. The second category only offer post mortem analysis of monitoring data, giving no insight into runtime status of the application.

There are many commonalities between Big Data and future exascale systems. We can extrapolate from this that the monitoring system and requirements should also be similar. In the following paragraphs 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 directly. 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. Datastores 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.

3.3 Monitoring of Parallel Applications running on HPC Systems

Contemporary petascale supercomputing systems offer unprecedented levels of the theoretically estimated peak hardware performance [27], but in reality utilizing cluster's regions in efficient manner remains a major challenge [2]. Scientific developers usually have to put huge effort in their specific field of science to find most suitable mathematical models of the problem they want to solve and computer science to understand and exploit the target computing architecture. Moreover usually it's not enough to write correct and efficient code for the target platform in a static manner, but it is also important to provide a fine-grained monitoring of the application to adapt it while satisfying dynamic constraints. Also in the real computing environment much of the complexity of application I/O interfaces is hidden behind advanced abstraction layers of programming frameworks, whose performance behavior is hard to detect and predict, especially in a shared multi-user large-scale environment.

Over the last decades, in parallel with the development of new heterogeneous many-core parallel computing systems, a large number of performance measurement and analysis tools have been created for these architectures and systems [21, 22]. These tools comprise low level instrumentation, performance measurement and analysis based on hardware counters, time measurements, call graphs, operating systems tracing and sampling tools, programming model specific performance tools covering, for instance, multi-threaded, OpenMP, and MPI programs.

3.3.1 Instrumentation, Measurement, and Analysis Technologies

HPCToolkit[17] is a framework for program performance analysis running on the wide variety of systems (e.g. multi-core workstations, supercomputers). It is primarily designed for analysis of pure or threaded MPI applications, but it can be as well used for sequential or threaded applications. For performance measurement, HPCToolkit uses statistical sampling of hardware performance counters.

Paradyn[18] is a performance measurement tool for parallel and distributed applications based on a dynamic performance instrumentation and measurement. Paradyn uses several novel approaches so that it can be used for probing long running programs on the large-scale systems. It can provide precise performance data down to the procedure and statement level. Paradyn controls its overhead by monitoring the cost of its data collection, limiting its instrumentation perturbation to a desired degree.

Dimemas[19] is a performance analysis tool for message-passing programs. It allows the user to tune parallel applications on a workstation, while generating an accurate prediction of their performance on the target parallel machine. The Dimemas simulator reconstructs the time behavior of a parallel application on a machine modeled by a set of performance metrics. Dimemas generates trace files that are suitable for Paraver enabling the user to scope any performance problems indicated by a simulator run. The analysis module performs critical path analysis reporting the total CPU usage of different code areas, as well as their importance for the program execution time.

Extra-P[20] is a performance-modeling framework that that is specifically tailored to the identification parts of the program whose scaling behavior is subject to some bottlenecks, that are much worse than expected. Extra-P uses data from the various of performance metrics at different processor configurations as input and generates performance map the of code regions (including their calling context) as a function of the number of processes. Extra-P generates not only a list of potential scalability bugs but also human-readable models for all performance metrics available such as floating-point operations or bytes sent by MPI calls that can be further analyzed and compared to identify the root causes of scalability issues.

mpiP[21] It is a lightweight profiling library for MPI applications. Since it collects only statistical information about MPI functions it brings considerably less overhead and much less data than tracing tools. All the information captured by mpiP is task-local. Typically It generates report at the end of the experiment by merging results from all of the tasks into single file.

Open|SpeedShop (O|SS)[22] is a comprehensive and extensible performance anal-

[17] http://hpctoolkit.org/
[18] http://www.paradyn.org/
[19] https://tools.bsc.es/dimemas
[20] http://www.scalasca.org/software/extra-p
[21] http://mpip.sourceforge.net/
[22] https://openspeedshop.org/

ysis tool set, that allow the user to collect a variety of different performance statistics about an application. This includes Program Counter (PC) sampling, Call Stack sampling, Hardware Performance Counters inspection, MPI Profiling and Tracing, I/O Profiling and Tracing to study an application's I/O patterns; and Floating Point Exception (FPE) analysis to detect floating point exceptions that can slow down applications. This too

Scalasca[23] is an open-source toolset that can be used to analyze the performance behavior of parallel applications and optimization. It has been specifically designed for use on large-scale systems including IBM Blue Gene and Cray XT, but is also well-suited for small- and medium-scale HPC platforms. Scalasca integrates runtime summaries with in-depth studies of concurrent behavior via event tracing.

TAU Performance System[24] is a well-known profiling and tracing toolkit for performance analysis of parallel programs. It is capable of gathering performance information through instrumentation of functions, methods, basic blocks, and statements as well as event-based sampling. It has a long term development history and provides an extremely rich functionality [26]. The instrumentation can be inserted in the source code automatically based on the Program Database Toolkit (PDT), dynamically using DyninstAPI [12], or manually. TAU's profile visualization tool, paraprof, provides graphical displays of all the performance analysis results, in aggregate and single node/context/thread granularity. TAU's event traces can also be displayed with the Vampir or Paraver trace visualization tools.

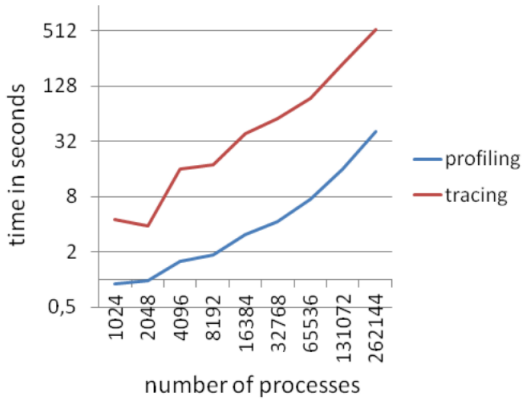

Figure 3.2: The finalization response-time of the Score-P measurement system depending on the number of processes. The PRIMA Project [22].

The Score-P[25] is a highly scalable measurement toolkit for profiling, event tracing, and online analysis of HPC applications[26]. The Score-P component consists

[23] http://scalasca.org/about/about.html
[24] http://tau.uoregon.edu
[25] http://score-p.org
[26] See paragraph *Motivation for a Joint Measurement Infrastructure* [19]

of an instrumentation framework, several run-time libraries, and some helper tools. The instrumentation allows users to insert measurement probes into C/C++ and Fortran codes that collect performance-related data when triggered during measurement runs. The collected performance data can be stored in the OTF2, CUBE4, or TAU formats.

3.4 I/O Instrumentation and Management Techniques

Performance monitoring of large scale parallel application reveals the next important problem. As it was previously explored in the literature (see Fig.3.2), when the number of processes in the parallel systems increases, the response time grows dramatically. The problem is that we have to collect a detailed enough information to find performance bottlenecks, yet collecting all this data can introduce serious collection bottlenecks. Especially, transition to the exascale systems [5, 6, 14, 24] with high degree of parallelism in conjunction with the "sharp" measure frames will lead to the significant escalation of aforementioned problem. Thus fine-grained control of the I/O ecosystem [21, p. 101] is required.

3.4.1 Bull Smart IO Instrumentation

Because I/O is one of the least known part of job executions, a tool suite to investigate the run-time behavior of applications is developed by Bull's Data Management Group. Those tools aim to help understand and to solve the bottlenecks that can appear during the run of applications in complex workloads.

An intuitive job level I/O profiling tool, Bull IO Instrumentation is a scalable tool that helps the system administrators, developers and product support experts analyzing the IO activity generated by HPC jobs, by collecting, storing and displaying a set of IO related metrics, such as IO volumes, IO response times, etc.

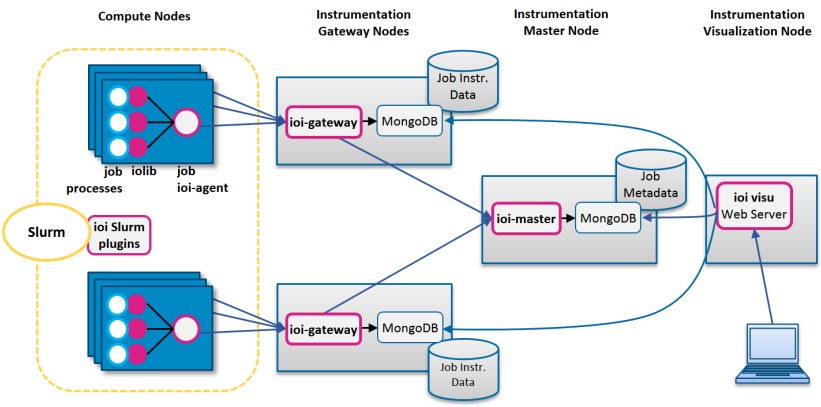

Figure 3.3: Bull IO Instrumentation Architecture.

Bull IO Instrumentation consists of two main parts:

- The IO metrics gathering chain, composed of:
  - an iolib library intercepting IO related calls to the glibc, and a job agent running on compute nodes.
  - a set of gateway services, receiving and aggregating IO metrics from the compute nodes, and storing them in a distributed MongoDB database. A gateway can serve up to 300 compute nodes and is typically instantiated on a service node.
  - a master service, in charge of storing job descriptions data such as job ID, start time, etc.
- A GUI (Graphical User Interface) and an API (Application Programming Interface), composed of a Web server and an application accessing the MongoDB database and creating comprehensive views.

IO Instrumentation does not require any application modification, as it relies on intercepting IO function calls at the glibc level through a LD_PRELOAD mechanism. Hence, it can instrument any application that is not statically linked with the glibc.

IO Instrumentation is integrated into SLURM through plugins that automate the instrumentation process. Once IO Instrumentation is installed and set up, any job submitted to SLURM using the command line option '–ioinstrumentation=yes' is automatically instrumented.

The job agent prepares a record for all the processes running on the node for each 5 seconds timeframe[27] the job is running, and sends these records to the gateway service it is attached to. Then, the gateway service performs additional processing on the node records received from all the compute nodes, and stores them in the MongoDB database. Additional information on jobs such as job name, environment, node list, start time, etc. is collected and is transferred to the master service, and stored in the MongoDB database as well.

The IO Instrumentation GUI is a Web application, designed as a database explorer. The main page displays all running and terminated jobs. Filters are available to select specific jobs. When a job is selected, an overview of the job IO behavior is displayed, and you can follow the evolution of the supported metrics over time.

Three user profiles can be defined:

- regular users, accessing only their own job records
- administrators, accessing all job records
- super-administrators, allowed to create and manage users

[27] The 5 seconds timeframe is not aligned with the job start time, but with the supercomputer system clock. So, the first and last time slice records of a job usually contain less than 5 seconds of job activity. This approach will allow, in a future IO Instrumentation release, to sum up time slice records from different jobs and to display a consolidated view of the overall supercomputer IO activity.

A REST API is provided to access raw database records.

Chapter 4

Requirement analysis

This section specifies the requirements for the **ASPIDE** monitoring model and consequently the monitoring environment. For that purpose, an analysis of the use cases information presented in Deliverable 5.1 has been conducted. From that analysis a list of monitoring requirements have been identified. Furthermore, a monitoring data representation format, which complies with the monitoring model, has been introduced.

Table 4.1: Software and Monitoring.

Use cases	Software	Monitoring
Urban computing	DMCF, Hadoop/Spark	CPU cores, RAM and disk, Apache Ambari, 1 min
Big Data application for mobile users	Hadoop (MapReduce, Spark, Yarn, HBase, ZooKeeper, Oozie)	Infrastructure modules status, Apache Ambari , 1s
Processing of magnetic resonance images	FreeSurfer and ANTS. FSL, MRtrix3, ANTS, SMT. Pipelined using Bash or Nipype	CPU and RAM usage, nipype, 1s
Deep learning	Python, scikit learn, TensorFlow with cuDNN, Horovod	CPU, GPU, memory, power, temperatures; BEMOS, nagios, 1s

We have conducted a detailed analysis of the use cases and utilized the information provided as appendix to Deliverable 5.1 to identify the monitoring requirements. The requirements of the use case applications in relation to the monitoring are defined in Table 4.1. It can be concluded that all use case applications require detailed monitoring information in order to function in a proper manner. Furthermore, we can not expect any future improvements in the execution time of such complex

applications in Exascale environment if we do not provide efficient infrastructure, hardware and software monitoring.

Therefore, based on the requirements described above, we have identified multiple important monitoring parameters and metadata. We have grouped these parameters into three distinctive categories:

- Application related monitoring parameters and metadata
- I/O related monitoring parameters
- Node related monitoring parameters

4.1 Application related monitoring parameters and metadata

The application related monitoring parameters contain detailed information on the execution of the application. More concretely, we envision the following parameters:

- "hostname" - contains information on which node the application is running
- "appName" - non-unique identification of the application with a human friendly naming
- "appID" - unique identification of the application
- "userName" - owner identification
- "startTime" - contains information when the application was started
- "endTime" - contains information when the application ended
- "migrationsCount" - contains information if the application was migrated and how many times
- "exceptionsI" - contains if errors were detected during application execution

In the case of Big Data platforms the above mentioned metrics can be easily found. For example in case of YARN and Spark we have metrics related to *hostname*, *appName* and *appID*. These metrics can be found in the form of *active_applications_across_yarn_pools*. Metrics related to *startTime* and *endTime* can be found in *JobHistory Server* for both Spark and YARN. Moreover, the metrics covered are more fine grained than mere job execution times, they include the breakdown into stages (i.e. map and reduce tasks) as well as a per *executor* or *NodeManager* view. Conversely, *migrationCount* can be also found in the history servers. In the case of *exceptionsI* we have several metrics which can fulfill this requirement. In case of YARN we have; *alerts_rate*, *events_critical_rate*, *jobs_failed_rate*, *jvm_blocked_rate* etc. For Spark we have metrics from the DAG Scheduler related to stages failed, message processing times and waiting stages.

4.2 I/O related monitoring parameters

The I/O monitoring parameters contain detailed information on the I/O operation performed during the execution of the application. More concretely, we envision the following parameters:

- "bytesRead" - contains information on the bytes read by the application
- "bytesWrite" - contains information on the bytes written by the application
- "operRead" - contains information on the read operations by the application
- "operWrite" - contains information on the write operations by the application
- "filesCreated" - contains information on the number of files created by the application
- "filesDeleted" - contains information on the number of files deleted by the application
- "exceptionsIO" - contains if errors were detected during I/O operations

As we have seen from the requirements at the start of chapter 5 in the case of Big Data frameworks the common denominator when it comes to storage and I/O is HDFS. This means that I/O related metrics detailed above can be matched to HDFS specific metrics. Read and writes to HDFS can be read at a DataNode level or globally. There are a wide range of read and write metrics detailing read/write rate times, data volume, replication count as well as JVM related metrics regarding garbage collection rates and times (i.e. *blocks_read_rate_across_datanodes*, *write_bytes_rate_across_datanodes* etc.). Metrics related to *filesCreated* and *filesDeleted* can be mapped to metrics such as *files_total*, *files_cached_across_hdfs_cache_directives*, *files_needed_across_hdfs_cache_directives*. In the case of *exceptionIO* in the case of HDFS we have metrics such as; *total_block_verification_failures_rate_across_datanodes*, *num_blocks_failed_to_cache_rate_across_datanodes* etc.

4.3 Node related monitoring parameters

The node related monitoring parameters contain detailed information on physical nodes where the applications are being executed. More concretely, we envision the following parameters:

- "nodeID" - contains information on the node identification
- "cpuUtil" - shows the current CPU utilization
- "ramAvail" - shows how much RAM memory is available
- "procNo" - shows the number of processes executed by the node

System metrics such as CPU, RAM and storage are important for any HPC and Big Data frameworks. In the case of Big Data frameworks we should keep in mind that there are several underlying layers of metrics which have to be taken into considerations. For example if we have a Spark application deployed on top of YARN, on any processing node we might find services related to these such as; NodeManager, DataNode, Spark Executor etc. Each of these will have their own CPU, RAM and Storage related metrics and will also influence the *procNo* parameter.

Chapter 5

Low overhead monitoring model

5.1 Performance Prediction and Assignment Problem - Informal Discussion

Our strategy aims at taking into account the network topology, amount of data exchanged between agents and current number of jobs, which perform intensive data exchange in the given scope (e.g. "C-area"), influenced by the probabilistic factors. Currently the model covers optimal *one-stage mapping algorithm* which is guaranteed to yield the best static solution under the set of limiting assumptions. The algorithm captures communication costs of the agents of the task as edge weights in an assignment graph.

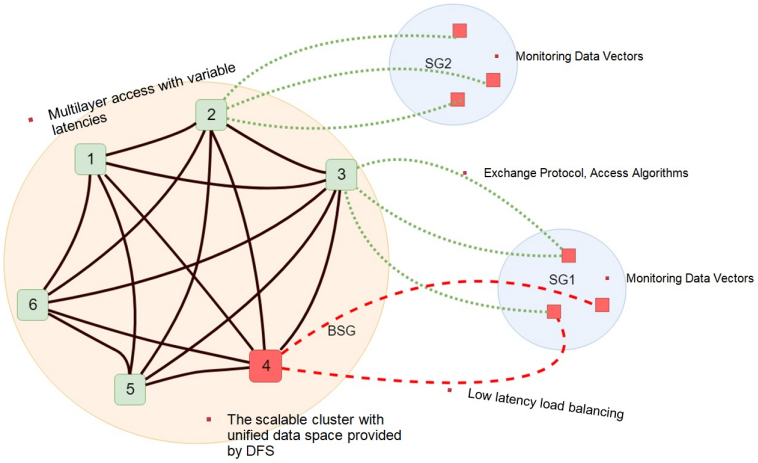

Figure 5.1: The graph model of the data flows in the monitoring system

A weighted sum of minimum response times in this graph then yields the optimal assignment. This algorithm has moderate complexity and is guaranteed to yield the optimal static assignment. More precisely, we defined an objective

function to minimize the time to perform the I/O and find an efficient monitoring agents and aggregators placement for the monitoring system. The general idea of our model lies in next basic steps to consider:

1. Obtain the information about currently running parallel application (as work flow) represented by the topology matrix.

2. Select the instrumented topology subspace, where you can neglect the time-dependent job distribution and architecture changes respect to the algorithm execution time.

3. Select the central node responsible for the subspace control.

4. Make the decision on the number and position of monitoring/aggregating agents in operation.

5. Select the relevant performance metrics and suggest the optimal control criteria subject to the set of constrains.

6. Estimate the performance of the obtained agents distribution.

7. Answer the question respect to the optimal criteria and number of constrains: Is the solution optimal?

8. If the optimality is not reached then re-distribute the number of agents and their positions while satisfying the set of given constrains.

5.2 Formal Model - The Problem of Response Time Optimization

This section presents a formal model and a set of essential definitions related to this deliverable. All notations related to the model are summarized in Table 5.1.

We define the HPC cluster as the set of $\mathbf{N} = \{n_i \mid 1 \leq i \leq MaxN\}$, homogeneous nodes interconnected by the corresponding network topology.

The network model covers:

1. Bitrate of each communication channel;

2. Fundamental and derivative delays;

3. Packet loss rate;

4. Density of the computational jobs per cluster[1]

[1]The influence of this factor will be explored during the second stage of T3.1 and reported in details in D3.2.

Table 5.1: Mathematical model parameters and their description.

$Symbol$	Description
N	Number of Computing Nodes
AG	Set of Aggregatros
MA	Set of Monitoring Agents
$DL_{m,a}$	Adjacency Matrix filled with delay values $DL_{i,j}$
m	Number of Monitoring Agents
a	Number of Aggregators
$nh_{i,j}$	Number of Hops
VD_i	Volume of Data to transmit over the channel (i, j)
SR_i	Sampling Rate of MA_i
T_i	Measurement Interval of MA_i
MO_i	Measurement Overhead of MA_i
$BW_{i,j}$	Bandwidth of the channel (i, j)
$PF_{i,j}$	Probabilistic Factor of the channel (i, j)
$X_{i,j}$	Binary variable which takes 0,1 values and Indicates that MA_i is assigned to the AG_j
Q_i	Resource usage limiting factor
$w_{i,j}$	Resource usage weight matrix
c_j	Connection(s) limiting factor
b_i	Connection(s) limiting factor
G	Bipartite Graph Generator that generates an optimal solution to the Linear Integer Programming Problem

The problem inspected in this model comprises a set of aggregators $\mathbf{AG} = \{AG_\gamma \mid 1 \leq \gamma \leq a\}$ and monitoring agents that perform statistical sampling $\mathbf{MA} = \{MA_\gamma \mid 1 \leq \gamma \leq m\}$, where a and m are the corresponding limitations required by the operator for the agents to be executed on the underlying HPC nodes. The product of those two sets forms subset of communication channels used by monitoring system to transfer the sampling data. Respect to the aggregators two cases are possible:

1. There is no single parallel data space on the cluster or we introduce our own application-specific data space (might be non-POSIX). So we can fully control the placement of the aggregating agents that export an access to the local file system;

2. We don't have an access to the control system (there is a single parallel data space managed by arbitrary DFS solution and nothing else is allowed) and can not manage the placement of the aggregators. Here we can affect the performance only by managing monitoring agents;

$$\mathbf{DL_{i,j}} = \begin{pmatrix} DL_{1,1} & DL_{1,2} & \cdots & DL_{1,m} \\ DL_{2,1} & DL_{2,2} & \cdots & DL_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ DL_{a,1} & DL_{a,2} & \cdots & DL_{a,m} \end{pmatrix} \tag{5.1}$$

Every job being monitored $Job_{mon} = (AG, MA, AG \times MA)$ comprises a number of such a channels that allow communication data between each other based on the next adjacency matrix 5.1. $DL_{i,j}$ is formed by the next equation:

$$DL_{i,j} = \left[DL_0 * nh(i,j) + \frac{VD_i(SR_i, T_i, MO_i)}{BW(i,j)} \right] (1 + PF(i,j)) \tag{5.2}$$

where $DL_{i,j}$ - communication delay, $nh(i,j)$ - number of hops to traverse, DL_0 - connection estimation delay, VD_i - volume of monitoring data to transmit per monitor, which depends on the selected SR - sampling rate, T - observation interval and MO - Meausrement Overhead, PF - Probabilistic Factor, which depends on the required measurement interval, Job density and MTBF-related factors.

We consider the *Response Time - RT* as the single objective which measures the efficiency of the of the monitoring agents and aggregators placement on the available nodes of an HPC cluster.

$$\mathbf{DL_{i,j}} = \begin{pmatrix} DL_{1,1} & DL_{1,2} & \cdots & DL_{1,m} \\ DL_{2,1} & DL_{2,2} & \cdots & DL_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ DL_{a,1} & DL_{a,2} & \cdots & DL_{a,m} \end{pmatrix} \tag{5.3}$$

RT is a very important parameter, which estimates the transition processes duration related to the data transfers from monitoring agents to the aggregators.

We define the *placement* of the monitoring application $Job_{mon} = (AG, MA, AG \times MA)$ on a subset of HPC nodes N as a function $G : (N \times N) \rightarrow (AG \times MA)$ that generates a bipartite graph $(AG \times MA)$ out of complete graph $(N \times N)$ subject to the set of constraints.

The optimal completion time problem of the monitoring operation is formulated in terms of the Linear Assignment Problem (LAP) [13]:

$$\min_{(G)} \sum_{i=1}^{m} \sum_{j=1}^{a} DL_{i,j} * X_{i,j} \tag{5.4}$$

Subject to the following constraints:

$$\begin{cases} \sum_{i=1}^{m} X_{i,j} = 1 \\ \sum_{j=1}^{a} X_{i,j} = 1 \\ 0 \leq SR_i \leq \frac{1}{\tau}, SR_i \in \mathbb{Q}, \forall i \in \{0...a\} \\ X_{i,j} \in \{0,1\} \subset \mathbb{Z} \end{cases} \tag{5.5}$$

Currently there several approaches to solve this problem presented in [9, 13, 20]. So far the problem can be extended to the Generalized Linear Integer Programming Problem, which can be express in the next form:

$$\min_{(G)} \sum_{i=1}^{m} \sum_{j=1}^{a} DL_{i,j} * X_{i,j} \tag{5.6}$$

Subject to the following constraints:

$$\begin{cases} \sum_{j=1}^{m} w_{i,j} * X_{i,j} \leq Q_i \\ \sum_{i=1}^{a} X_{i,j} \leq c_j \\ \sum_{j=1}^{m} X_{i,j} \leq b_i \\ 1 \leq a \leq n, 1 \leq m \leq n \\ 0 \leq SR_i \leq \frac{1}{\tau}, SR_i \in \mathbb{Q}, \forall i \in \{0...a\} \\ X_{i,j} \in \{0,1\} \subset \mathbb{Z} \\ Q_i, c_j, b_j, a, m \in \mathbb{Z} \end{cases} \tag{5.7}$$

where $X_{i,j}$ - binary variable which takes 0,1 values, indicating that MA_i assigned to the AG_j, $\sum_{j=1}^{m} w_{i,j} * X_{i,j}$ - limitation (knapsack type) for resource usage, c_j and b_i - connection limiting constants. The more detailed description and implementation will be reported in D3.2.

5.3 Further Work and Model Improvement

Real-world installation in general case represents by itself an open system and experiences high variety of perturbations which can seriously affect the performance. It is known, for example, that with increase of the jobs density per node, intensity of the data exchange (the way they interact) and duration of the calculations the probability of failure increases as well. Many researchers report that in the case of extreme-scale data-intensive systems the factor of probability will become much more dominant and it will be no longer possible to neglect it in the mathematical model for *accurate* performance predictions. This leads us to the problem of optimal dynamic discrete control [7, 8] with required degree of noise influence, where it

should be decided for each monitoring agent how many channels it should select and how frequently it should communicate with aggregators to perform it's job in shortest possible period.

Chapter 6

Conclusion

In this deliverable the **ASPIDE** monitoring model for Exascale systems, which is an essential requirement for deploying scalable monitoring architectures, has been described. The research and development work conducted during the first 9 months of the project timeline has resulted in development of a optimization model that exploits multiple system wide parameters, such as communication performance and jobs population, in order to optimize the distribution and placement of the monitoring aggregators in relation to the monitoring agents. As the project work deals with the implementation of a combinatorial ILP optimisation problem, where the main incentive is to find the proper mapping of aggregators/monitoring agents across highly distributed Exascale systems, we present a mathematical model that demonstrate the ability of the proposed approach to provide satisfactory distribution of the monitoring data aggregators across Exascale systems. The mathematical model itself has been developed by considering the requirements of the **ASPIDE** use-case applications. As described in the project description by the end of month 18, the model will be implemented as a software tool, will consequently will be integrated in the extreme scale monitoring architecture.

In this deliverable the **ASPIDE** monitoring model for Exascale systems, which is an essential requirement for deploying scalable monitoring architectures, has been described. The research and development work conducted during the first 9 months of the project timeline has resulted in development of a optimization model that exploits multiple system wide parameters, such as communication performance and jobs population, in order to optimize the distribution and placement of the monitoring aggregators in relation to the monitoring agents. As the project work deals with the implementation of a combinatorial ILP optimisation problem, where the main incentive is to find the proper mapping of aggregators/monitoring agents across highly distributed Exascale systems, we present a mathematical model that demonstrate the ability of the proposed approach to provide satisfactory distribution of the monitoring data aggregators across Exascale systems. The mathematical model itself has been developed by considering the requirements of the **ASPIDE** use-case applications. As described in the project description by the end of month

18, the model will be implemented as a software tool, will consequently will be integrated in the extreme scale monitoring architecture.

Bibliography

[1] Collectd documentation, Jan. 2016.

[2] Erika Abraham, Costas Bekas, Ivona Brandic, Samir Genaim, Einar Broch Johnsen, Ivan Kondov, Sabri Pllana, and Achim Streit. Preparing hpc applications for exascale: Challenges and recommendations. In *Network-Based Information Systems (NBiS), 2015 18th International Conference on*, pages 401–406. IEEE, 2015.

[3] Khalid Alhamazani, Rajiv Ranjan, Karan Mitra, Fethi Rabhi, Prem Prakash Jayaraman, Samee Ullah Khan, Adnene Guabtni, and Vasudha Bhatnagar. An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art. *Computing*, 97(4):357–377, April 2015.

[4] Vaggelis Antypas, Nikos Zacheilas, and Vana Kalogeraki. Dynamic reduce task adjustment for hadoop workloads. In *Proceedings of the 19th Panhellenic Conference on Informatics*, PCI '15, pages 203–208, New York, NY, USA, 2015. ACM.

[5] M Asch, T Moore, R Badia, M Beck, P Beckman, T Bidot, F Bodin, F Cappello, A Choudhary, B de Supinski, et al. Big data and extreme-scale computing: Pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry. *The International Journal of High Performance Computing Applications*, 32(4):435–479, 2018.

[6] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, et al. Exascale computing study: Technology challenges in achieving exascale systems peter kogge, editor & study lead. 2008.

[7] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1.

[8] Dimitri P Bertsekas. Dynamic programming and stochastic control. *Mathematics in science and engineering*, 125:222–293, 1976.

[9] Dimitri P Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of operations research*, 14(1):105–123, 1988.

[10] Mahantesh N. Birje and Sunilkumar S. Manvi. Wigrimma: A wireless grid monitoring model using agents. *Journal of Grid Computing*, 9(4):549–572, Dec 2011.

[11] J. Breitbart, J. Weidendorfer, and C. Trinitis. Case study on co-scheduling for hpc applications. In *2015 44th ICPP Conference Workshops*, pages 277–285, Sep. 2015.

[12] Bryan Buck and Jeffrey K Hollingsworth. An api for runtime code patching. *The International Journal of High Performance Computing Applications*, 14(4):317–329, 2000.

[13] Rainer E Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment problems*. Springer, 2009.

[14] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60, 2011.

[15] Juan Gutierrez-Aguado, Jose M. Alcaraz Calero, and Wladimiro Diaz Villanueva. Iaasmon: Monitoring architecture for public cloud computing data centers. *Journal of Grid Computing*, 14(2):283–297, Jun 2016.

[16] G. Iuhasz and I. Dragan. An overview of monitoring tools for big data and cloud applications. In *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 363–366, Sept 2015.

[17] Matthew Jacobs. Challenges and lessons learned building monitoring anddiagnostics tools for hadoop. In *Proceedings of the 2012 Workshop on Management of Big Data Systems*, MBDS '12, pages 33–34, New York, NY, USA, 2012. ACM.

[18] A. Kertesz, G. Kecskemeti, M. Oriol, P. Kotcauer, S. Acs, M. Rodríguez, O. Mercè, A. Cs. Marosi, J. Marco, and X. Franch. Enhancing federated cloud management with an integrated service monitoring approach. *Journal of Grid Computing*, 11(4):699–720, Dec 2013.

[19] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, et al. Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. In *Tools for High Performance Computing 2011*, pages 79–91. Springer, 2012.

[20] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[21] Michael Lysaght, Bjorn Lindi, Vit Vondrak, John Donners, and Marc Tajchman. A report on the survey of hpc tools and techniques - d7.2.1. http://www.prace-ri.eu/IMG/pdf/d7.2.1.pdf, 2013. [Online; accessed 25-Feb-2019].

[22] Allen D. Malony and Felix G. Wolf. Performance refactoring of instrumentation, measurement, and analysis technologies for petascale computing. the prima project. 1 2014.

[23] Florin Pop, Ciprian Dobre, Corina Stratan, Alexandru Costan, and Valentin Cristea. Dynamic Meta-Scheduling Architecture Based on Monitoring in Distributed Systems. In *2009 International Conference on Complex, Intelligent and Software Intensive Systems*. IEEE, 2009.

[24] Ioan Raicu, Ian T Foster, and Pete Beckman. Making a case for distributed file systems at exascale. In *Proceedings of the third international workshop on Large-scale system and application performance*, pages 11–18. ACM, 2011.

[25] S. Rajkumar, N. Rajkumar, and V. G. Suresh. Hybrid approach for monitoring and scheduling the job in heterogeneous system. In *International Conference on Information Communication and Embedded Systems (ICICES2014)*. IEEE, 2014.

[26] Sameer S Shende and Allen D Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[27] TOP500. List - november 2018. https://www.top500.org/list/2018/11/, 2018. [Online; accessed 25-Feb-2019].

[28] J. Venner, S. Wadkar, and M. Siddalingaiah. *Pro Apache Hadoop*. Apress, 2014.