In: The 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2020), Vasteras, Sweden, March 2020.

M3AT: Monitoring Agents Assignment Model for Data-Intensive Applications

Vladislav Kashansky*†, Dragi Kimovski*, Radu Prodan*, Prateek Agrawal*‡, Fabrizio Marozzo§, Gabriel Iuhasz¶, Marek Justyna‖ and Javier Garcia-Blas**

*Institute of Information Technology, University of Klagenfurt, Austria
†School of Electrical Engineering and Computer Science, South Ural State University, Russia
‡ Lovely Professional University, India
§ DIMES Department, University of Calabria, Italy
¶ West University of Timisoara, Romania
‖ Poznan Supercomputing and Networking Center, Poland
** Department of Computer Science and Engineering, University Carlos III of Madrid, Spain

***Abstract*—Nowadays, massive amounts of data are acquired, transferred, and analyzed nearly in real-time by utilizing a large number of computing and storage elements interconnected through high-speed communication networks. However, one issue that still requires research effort is to enable efficient monitoring of applications and infrastructures of such complex systems. In this paper, we introduce an Integer Linear Programming (ILP) model called M3AT for optimized assignment of monitoring agents and aggregators on large-scale computing systems. We identified a set of requirements from three representative data-intensive applications and exploited them to define the model's input parameters. We evaluated the scalability of M3AT using the Constraint Integer Programing (SCIP) solver with default configuration based on synthetic data sets. Preliminary results show that the model provides optimal assignments for subsystems composed of up to 200 monitoring agents with complex I/O policies, while keeping the number of aggregators constant and demonstrates variable sensitivity with respect to the scale of monitoring data aggregators and limitation policies imposed.**

***Index Terms*—Monitoring systems, high performance computing, aggregation, systems control, data-intensive systems, generalized assignment problem, SCIP optimization suite.**

I. Introduction

In the last years, the ability to produce and gather data has exponentially increased. In the Internet of Things' era, huge amount of digital data is generated and collected from various sources, such as sensors, cameras, mobile devices, GPS devices, web applications and services [1], which must be acquired, pre-processed and analyzed in real-time through a large set of computing and storage nodes. Examples of systems are social media platforms that analyse concurrently various patterns and trends related to human behaviours, driver-less vehicles that receive information from a very large number of sensors and need to make immediate decisions, or medical imaging applications that continuously analyse different types of images to provide unique and complementary information to the medical professionals.

Novel architectures, programming models and systems paired with high performance computing (HPC) systems, such as many-core and multi-core computers, clouds, and multi-clusters, are commonly used by data analysts to tackle big data issues and get valuable information and knowledge in a reasonable time [2]. One of the main issues that still requires additional research efforts is efficient monitoring of data intensive applications and corresponding HPC infrastructures. The major problem for deploying large-scale monitoring platform is the measurement of performance metrics in the context of computing infrastructures composed of million of nodes with complex interconnects and multi-layered architectures. It stands to the reason that it is impractical and many times impossible to globally measure the performance metrics of large-scale applications, while preserving, for example, I/O limitation policies. Thus, it is critical to identify:

- The parameters to monitor and the granularity level (e.g dynamic tracing, profiling, per-node aggregated statistics, per cluster I/O heatmaps);
- The measurement interval and the communication patterns in relation to these intervals;
- The aggregation and pre-processing of performance metrics at a monitor granularity for further analysis.

This paper focuses on answering the last two questions by the introducing a mathematical model called M3AT for the optimized assignment of monitoring agents and aggregators, that reduces the transfer time of the raw monitoring data, while meeting strict I/O limitations. We identified a set of requirements from three representative data intensive applications of the ASPIDE project[1] and exploited them to define the input parameters of the model. The contributions of this paper are the following:

1) a background review of the monitoring practices for the converging HPC and Cloud computing systems;
2) a preliminary architecture of the large-scale monitoring system, proposed in ASPIDE project;
3) an ILP approach for assigning monitoring agents based on the Generalized Assignment Problem (GAP) formulation;

[1]https://www.aspide-project.eu/

4) a practical approach for selecting monitoring aggregation points using the SCIP optimization suite using several variable complexity data sets and precision.

The remainder of the paper is structured as follows. Section II discusses related work. Section III describes the use-case applications, and the requirements of the proposed model and the monitoring environment. Section VI-A describes the proposed M3AT assignment model. Section VII presents an experimental evaluation and Section VIII concludes the paper.

II. Background and Related work

This section reviews current approaches, tools and frameworks for monitoring of large-scale parallel and distributed systems for general purpose and data-intensive processing.

A. Data-intensive cloud application monitoring

A cross-layer monitoring scheme is crucial for big data platforms on the large-scale cloud systems due to the distribution of application components on multiple virtual machines across cloud layers. Currently, most data intensive applications run on Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) or Infrastructure-as a-Service (IaaS). The work in [3] analyses the problem for public clouds and proposes an architecture for a monitoring platform. In a IaaS deployment, one typically monitors system metrics like CPU usage, together with its state, memory, storage and network utilization. In contrast, PaaS and SaaS-level metrics focus on more abstract or complex metrics such as byte throughput, status of system services, uptime, and availability. A good example is a Hadoop YARN deployment that uses several monitoring metrics such as MapReduce processing time, job turnaround, or shuffle operation.

Optimizing data-intensive applications requires a comprehensive overview of historical metrics, as parameter tuning is difficult in the case of missing ones. For example, it is difficult or impossible to tune an application that runs on a big data platform by having access to system metrics only. In case of job and reducer task scheduling [4], monitoring data is crucial too. The authors in [5] described a dynamic architecture for monitoring large-scale distributed systems that uses the MonaLISA service to gather information for automatically improving task scheduling. Another work in [6] presented a distributed resource monitoring architecture to identify the optimal set of machines to execute an application, and an implementation of prediction methods to evaluate the overall performance of a node. The authors in [7] proposed a method for co-scheduling CPU and memory-intensive applications onto the same node by using monitoring information to improve the overall throughput and the energy efficiency. Lastly, [8] describes a minimalist monitoring approach.

There are several monitoring solutions currently in use or in development. For centralized monitoring, all resource states and metrics are sent to a centralized monitoring server that continuously pulls them from each monitored component. This allows a more controlled management and data access, while potentially sacrificing availability and elasticity by introducing a single point of failure, and eliminating the possibility of horizontal scaling. Decentralized architectures address most problems related to scaling and elasticity of centralized architectures with no single point of failure. The central authority is neutralized in structured peer-to-peer (P2P) systems, thus eradicating the central point of failure. Unstructured P2P network overlays are distributed in nature where the search directory is not centralized, while in hybrid P2P systems super-peers serve as localized search hubs for small network portions [9].

B. Monitoring data-intensive HPC applications

Contemporary HPC systems offer extreme levels of theoretically estimated peak hardware performance[2]. However, utilizing these systems in an efficient manner remains a major challenge [10] in practice. Scientific software developers and mathematicians usually have to invest huge efforts in finding suitable mathematical models for their problem and efficient implementations tuned for the target computing architectures. Moreover, it is not enough to write correct and efficient code for the target platform in a static manner, but is also important to provide fine-grained monitoring and adapt the application, while satisfying dynamic constraints. In a real computing environment, much of the complexity of the application I/O interfaces is hidden behind advanced abstraction layers of programming frameworks and data management systems, whose performance behavior is hard to predict, especially in a shared data-intensive environment.

Over the last decades, alongside with the development of new heterogeneous software-defined computing systems, a large number of model-specific performance measurement and analysis tools[3] emerged [11], [12]. These tools comprise low-level instrumentation, performance measurement and analysis based on hardware counters, time measurements, call graphs, operating systems tracing and sampling tools, and programming model-specific performance tools covering, for instance, multi-threaded, OpenMP, and MPI programs.

Finally, collecting detailed information to find performance bottlenecks introduces I/O contention and collection overheads [11]. Especially the transition [13] to pre-exascale systems with a high degree of parallelism, in conjunction with the "sharp" measure frames, leads to a significant escalation of this problem. One tool for scalable HPC I/O characterization is Darshan [14], designed to accurately capture important properties such as I/O access patterns with minimum overhead.

The approach employed by the ASPIDE project is the Bull Smart IO Instrumentation tool [15], composed of two parts:

a) I/O metrics gathering chain, consisting of:

- an `iolib` library intercepting I/O-related calls to the `glibc` library and a job agent running on compute nodes;
- a set of gateway services, receiving and aggregating I/O metrics from the compute nodes and storing them in a distributed MongoDB database. A gateway can serve up to 300 compute nodes;

[2]TOP500. https://www.top500.org/list/2018/11/

[3]VI-HPS. https://www.vi-hps.org/tools/tools.html

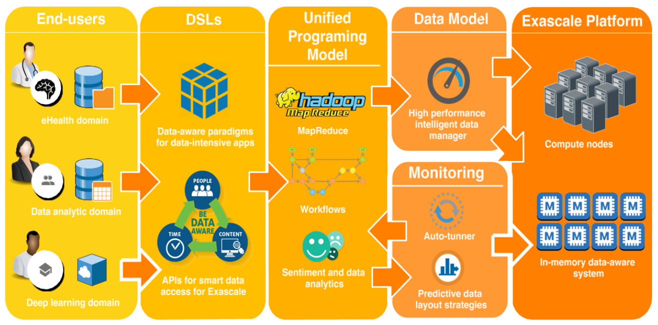

Figure 1: ASPIDE architecture design.

- a master service, in charge of storing job description data such as job identifier and start time.

b) Graphical User Interface: and API, composed of a Web server and an application accessing the MongoDB database and creating comprehensive views. The I/O instrumentation does not require application modification and relies on intercepting I/O function calls at the `glibc` dynamic library level through a `LD_PRELOAD` mechanism.

III. ASPIDE Project

This section covers the requirements for the monitoring model and environment from the project's perspective.

A. Project architecture and goals

The project aims to provide programming models to assist developers in building data-intensive applications for large-scale systems, while ensuring compliance with requested data management and performance. More concretely, it targets support for large-scale data processing through HPC, starting with the vision of changing the current programming paradigms in a data-driven style, improving the data access mechanisms, and developing novel methods for heterogeneous data analytics.

The project vision shown in Figure 1 establishes three research directions. First, it focuses on the design of fundamental models and paradigms for large-scale parallel programming in heterogeneous systems with runtime support for data-driven malleable computation. Second, it emphasizes the programming of extreme data analytics applications and the development of data-centric tools by exploiting sate-of-the-art monitoring, debugging and data analysis techniques. Third, it relies on big data applications to identify barriers in the management of existing HPC systems, and removes them with automated data-layout strategies and cross-layer data management at both client and server sides.

IV. Monitoring system requirements

We analyzed the project use-case applications to steer the monitoring model development and identified a set of critical requirements, summarized in Table I. Furthermore, we also introduced a monitoring data representation format compliant with the M3AT monitoring model. All applications require detailed hardware and software monitoring parameters to be correctly and efficiently executed, grouped into three categories:

- *Application-related parameters* and metadata contain information on the application execution, such as start and end time, dynamic traces, memory utilization, eventual migrations, and intercepted exceptions;
- *I/O-related parameters* contain information on the I/O operation during application execution, including read/write bytes, read/write operations and I/O exceptions;
- *Node and operating system-related parameters* contain aggregated information on physical and virtual execution nodes, such as resource utilization and node identification.

Table I shows that system metrics such as CPU, RAM and I/O with per-node granularity are important for the both HPC and big data systems. At the same time, there are several underlying layers [16] of metrics to consider for application execution. Combining them with power measurement data and environmental parameters such as temperature introduces additional complexity for system control software. Thus, it is important to provide an approach that allows aggregation and pre-processing of coarse-grained data subject to the required application scale. We therefore propose the concept of software-defined monitoring agents and aggregators that form an intermediate pre-processing system and introduce a chain of new research and engineering questions in order to make it feasible in real world. In this context, one of the most important questions is assigning these agents while keeping the imposed limitations and providing required performance.

V. ASPIDE monitoring system

The monitoring and analytics system enables a distributed data-centric monitoring and analysis, exposing and associating the collected metrics with the potential application bottlenecks. The key insight behind such an approach is that the source of a bottleneck in a data-intensive applications is often takes place not where it is detected (i.e. where the data is processed with a high communication or thrashing overhead). The conceptual monitoring system depicted on Figure 2 consists of four components:

1) *M3AT* for monitoring agents and aggregators assignment;
2) *Aggregation and event detection component (AEDC)* provides monitoring data aggregation from the agents and detects possible events. This component is decentralized and runs an instance on every aggregation node;
3) *Main analysis component (AC)* is centralized and provides a set of analytic tools, including smart monitoring of application performance and bottlenecks detection;
4) *Application Programming Interface (API)*, which enables access to the services provided by the monitoring platform.

VI. M3AT Architecture Design

M3AT, depicted on Figure 3, performs time-optimal assignment of monitoring agents and aggregation points. It targets efficient low latency monitoring to assist the main analysis component (AC) in timely identification of the inter-relationships between monitored applications, crucial for the

Table I: Software and monitoring requirements in the ASPIDE project.

Use case	Software	Monitoring
Urban computing	DMCF, Hadoop Spark	CPU cores, RAM and disk, Apache Ambari [1 min], I/O latency heatmap
Magnetic resonance image processing	FreeSurfer and ANTS. FSL, MRtrix3, ANTS, SMT, Bash/Nipype pipelines	CPU/RAM use, Nipype [1 s], I/O latency heatmaps
Object detection and computational fluid dynamics	SLURM, Lustre, Python, Scikit learn, TensorFlow with cuDNN, Horovod	CPU, GPU, memory, power, temperatures; BEMOS, Nagios, dynamic profiling [10 s]

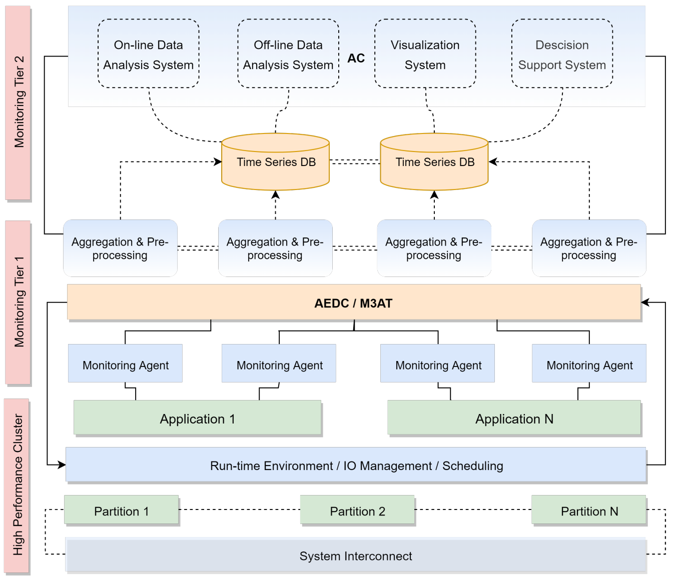

Figure 2: Conceptual monitoring system architecture.

online scheduling and auto-tuning. M3AT revolves around an optimization and assignment module, which applies a discrete optimization formalism (described in Section VI-A) to optimally assign monitoring agents and data aggregation points. Through an external API, M3AT receives from the runtime and I/O management systems information about the application spawn map and monitored metrics, including data volume, bandwidth requirements, and system topology. Thereafter, the ILP module searches for an assignment of monitoring agents and aggregators that meets the strict I/O requirements. Lastly, the monitoring system receives the assignment solution for further configuration of the agents.

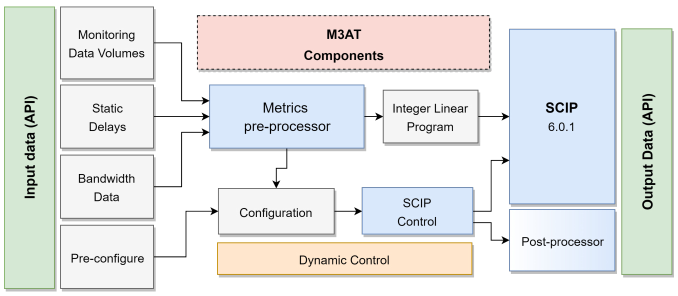

Figure 3: M3AT architecture design.

A. Monitoring agent assignment

The M3AT model aims to identify an optimal assignment [17] of monitoring agents (see Figure 4) and aggregation points, where the monitoring data needs to be pre-processed for further analysis. Initially, the monitoring agents are selected from the partitions, subject to a given application. Thereafter, we assign to the monitoring agents a subset of the required aggregators guaranteeing a low response time and a fixed amount of monitoring traffic within the given upper limits.

The assignment algorithms can vary in complexity, but the problem is generally proven to be NP-complete [18], [19], especially if we directly consider traffic limitations in the model and non-symmetrical matching on the bipartite graph. Thus, the problem needs to be decomposed on the architectural level in independent hierarchical pieces in a divide-and-conquer fashion. The reason for this architectural pattern is the nature of large-scale optimization problems, known to be very resource-demanding in both cases of pure or mixed ILP [20].

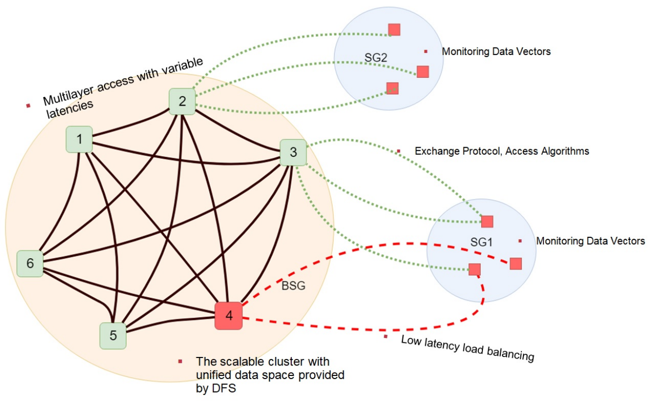
Multilayer access with variable latencies
SG2
Monitoring Data Vectors
Exchange Protocol, Access Algorithms
SG1
Monitoring Data Vectors
BSG
Low latency load balancing
The scalable cluster with unified data space provided by DFS
1
2
3
4
5
6

Figure 4: Graph model of data flows in the monitoring system.

Currently, the M3AT model covers an optimal *one-stage assignment approach* guaranteed to yield nearly-optimal solutions to the static model under a set of limiting assumptions. The algorithm captures communication costs of the agents as the edge weights in an assignment graph. A weighted sum of minimum response times in this graph then yields the optimal assignment. More precisely, we defined an objective function to minimize the time to perform the I/O operations and find an efficient monitoring agents and aggregators assignment for the monitoring system. The applicability of our model is limited by the following assumptions:

Table II: Mathematical model parameters and their description.

Symbol	Description
AG	Set of monitoring data aggregators
MA	Set of monitoring agents
$\widetilde{D}$	Delay matrix for each channel (i, j)
DL_{ij}	Static delay over the channel (i, j)
a	Number of monitoring agents
m	Number of aggregators
nh_{ij}	Number of Hops over the channel (i, j)
VD_j	Data volume to transmit for the given MA_j
SR_j	Sampling rate of MA_j
T_j	Measurement interval of MA_j
w_{ij}	Bandwidth of the channel (i, j)
x_{ij}	Binary variable, indicates that MA_j is assigned to AG_i
C_i	Bandwidth usage limiting factor for the given MA_i

- The a-priori information about the running application is already present, and delivered by the runtime system;
- The number and location of monitoring and aggregating agents is provided by runtime and data management systems;
- The relevant application performance metrics have already been selected and considered as the data volume, accumulated within the given push interval;
- The optimal control criteria (objective function) with the set of constraints is not changing during the solving procedure of the optimization problem.

B. Formal assignment model

Considering these requirements, we present a formal model and a set of essential definitions, summarized in Table II. The problem inspected in this model comprises a set of the monitoring agents MA and aggregators AG:

$$MA = \{MA_j \mid 1 \leq j \leq a\} \tag{1}$$
$$AG = \{AG_i \mid 1 \leq i \leq m\} \tag{2}$$

where a and m are the model limitations for the both classes of agents to be executed on the underlying partitions. The product of these two sets forms subset of communication channels used by the monitoring system to transfer the sampled data.

Every monitored application comprises a number of such channels that allow communication between each MA_j and AG_i with the delay properties, described by the matrix:

$$\widetilde{D} = \begin{array}{c c} & \begin{array}{ccc} MA_1 & \cdots & MA_j \end{array} \\ \begin{array}{c} AG_1 \\ \vdots \\ AG_i \end{array} & \begin{pmatrix} c_{1,1} & \cdots & c_{1,j} \\ \vdots & \ddots & \vdots \\ c_{i,1} & \cdots & c_{i,j} \end{pmatrix} \end{array} \tag{3}$$

$$c_{ij} = \overbrace{DL_{ij}}^{\text{static delay}} + \overbrace{\frac{VD_j(SR_j, T_j)}{w_{ij}}}^{\text{data-dependent delay}} \tag{4}$$

The term DL_{ij} is the static connection-estimation delay, modelled as a function of number of hops nh_{ij} for pure networking delays, or as a function-based on delay distribution of the given I/O management system for the complex multi-layered architectures. The optimal completion time of the monitoring data transfer can then be formulated as the GAP [17] by introducing per-aggregator capacities:

$$\sum_{j=1}^{a} w_{i,j} \cdot x_{ij} \leq C_i \qquad \forall i \in AG. \tag{5}$$

The objective is to minimize the sum of all possible completion times of data transfers for every MA_j to AG_i:

$$\begin{aligned} \underset{x_{ij}}{\text{minimize}} \quad & \sum_{i=1}^{m} \sum_{j=1}^{a} c_{ij} \cdot x_{ij}, \\ \text{subject to:} \quad & \sum_{j=1}^{a} w_{i,j} \cdot x_{ij} \leq C_i \qquad \forall i \in AG \\ & \sum_{i=1}^{m} x_{ij} = 1 \qquad \forall j \in MA \\ & x_{ij} \in \{0, 1\} \subset \mathbb{Z} \\ & w_{ij} \in (0, 1] \subset \mathbb{Q} \\ & c_{ij} \in \mathbb{Q}, \end{aligned} \tag{6}$$

where $x_{ij} \in 0, 1$ indicates the assignment of MA_j to AG_i. The first constraint is defined for every AG_i and represents a knapsack-type bandwidth limitation $w_{i,j}$ defined for each possible assignment x_{ij}. Practically speaking, it models the situation when the several monitoring agents are capable to transmit the data at the given rates, but the aggregator MA_j can only support the total inflow not exceeding the given policy-based limit C_i. The second constraint indicates that every monitoring agent MA_j must be assigned to any aggregation point and it impossible to have the unassigned MA_j within the given problem. The problem with this formulation is NP-hard [21], due to posing strict policy-based allocation limits.

C. Generalized Assignment Problem (GAP)

We observe that the problem can be reduced to the GAP, which has a huge importance in the operational research sciences. Practically, it has many real-life use-case scenarios [22] due to discrete nature of the decision in many controlled processes. Exact algorithms for GAP prove to be efficient only for problems [23] of a few hundred variables [21], [24]. The survey [25], for example contains the review of classic approaches to solve this problem, including method of Lagrangean relaxations. Most of the contemporary techniques follow the solution workflow diagram depicted on the Figure 5. Where, at first, the presolving stage occurs to reduce the possible search space, then Branch-and-Cut [26] procedure with several so-called primal heuristics are applied to move in the solution tree of the search space towards the optimum. For the large-scale optimization problems Danzig-Wolfe and Benders' [20], [26] decomposition methods with meta-heuristic strategies received a great degree of applicability [17], including simulated annealing and tabu search [23], variable depth search, ant colony and evolutionary algorithms.

D. Discussion on the possible simplification

The most satisfying simplification of this problem is formulating it as a linear assignment problem [17]:

$$\begin{aligned}
\underset{x_{ij}}{\text{minimize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\cdot x_{ij}, \\
\text{subject to:} \quad & \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in AG \\
& \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \in MA \\
& x_{ij} \in \{0,1\} \subset \mathbb{Z} \\
& c_{ij} \in \mathbb{Q}.
\end{aligned} \tag{7}$$

This approach is known to have P-complexity [17] and can be solved by the Hunagrian algorithm [27], however it does not explicitly include traffic limitations expressed by Eq. 5. Moreover, it requires the number of monitoring agents a be equal to aggregation points m, which might not be feasible in practice, especially with a direct formulation of the problem. However, it is still applicable in opportunistic fashion by introducing, for example, monitoring agents clustering with subsequent cluster to aggregation point assignment. There are currently several approaches to solve this problem presented in [17], [27], [28]. For the current study, we have decided to proceed with formulation of the problem given by Eq. 6, by keeping the complexity, but compensate it by applying state-of-the-art techniques in the field of discrete optimization.

VII. Experimental evaluation

In this section, we evaluate the M3AT monitoring assignment model based on a synthetic benchmark data set.

A. Experimental setup

We implemented the model in C/C++ by using the SCIP Optimization Suite [29] with SCIP 6.0.2 with default[4] configuration and SoPlex 4.0.2 LP solver. We implemented the assignment model as a stand-alone service with a command-line interface that enables straightforward use of its functionalities (see Figure 3). We compiled the source files for the model and SCIP using `gcc` version 4.8.5 20150623 (Red Hat 4.8.5-36) and handled the experimental data using the BASH 4.2.46(2)-release and MariaDB 5.5.64. We performed the experiments on a dedicated dual-socket node Intel Xeon CPU E5620 at 2.4 GHz with 16 GB of memory, running CentOS 7 (3.10.0-957.27.2.el7.x86˙64).

B. Data Generation Policy

We tested the model on a set of assignments problems, sampled with a uniform distribution[5] using variable random seeds ranging form 100 to 150, generated by the MT19937 generator [30] of the GNU Scientific Library 2.0.1 [31], [32]. We identified the constants for the uniform distribution based on

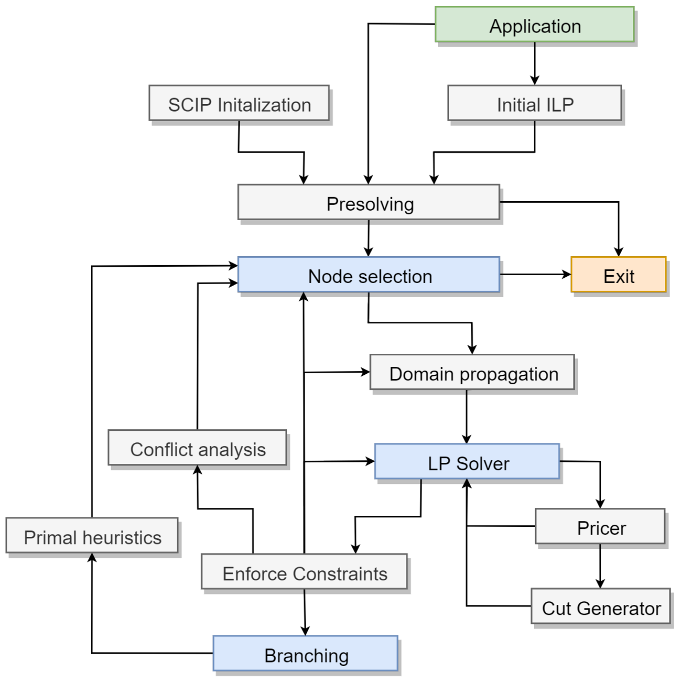

Figure 5: SCIP 6.0.2 internal diagram of solution-workflow process.

the use-case ecosystem requirements. In particular, Equations 8 and 9 model the situation with relatively low connection estimation latency (static delays) and high variation in the data-dependent transmission latency:

$$w_{ij} = \mathbb{U}(0.01, 1); \tag{8}$$

$$c_{ij} = \overbrace{\mathbb{U}(0.01, 5)}^{\text{static delay}} + \overbrace{\frac{\mathbb{U}(0.10, 1)}{w_{ij}}}^{\text{data-dependent delay}}. \tag{9}$$

We therefore created three complexity classes of the limitation policies $A^\dagger, B^\dagger, C^\dagger$ in relation to the works in [21], [33], [34]:

$$\text{Class } A^\dagger : C_i = \begin{cases} 0.6\cdot\frac{n}{m} + 0.4\cdot & \\ \cdot \max\limits_{\forall i \in AG} \sum_{j=1}^{a} w_{i,j}, & \mathbb{U}(0.00,1) \geq 0.5; \\ \text{Eq. 11}, & \text{otherwise}; \end{cases} \tag{10}$$

$$\text{Class } B^\dagger : C_i = m^{-1}\cdot\sum_{j=1}^{a} w_{ij}; \tag{11}$$

$$\text{Class } C^\dagger : C_i = 0.8\cdot m^{-1}\cdot\sum_{j=1}^{a} w_{ij}. \tag{12}$$

Each class provides possible I/O limitation scenarios imposed to the given aggregators set. The equations 10 – 12 model the real situations that occur in the practice. For example, complexity classes $B^\dagger$ and $C^\dagger$ model an environment with bandwidth saturation within a given HPC cluster partition by setting limitation policies inversely proportional to the current number of aggregators and possible amount of traffic

[4]https://scip.zib.de/doc-6.0.2/html/PARAMETERS.php

[5]Sample size of 50 for each complexity class $A^\dagger$, $B^\dagger$, $C^\dagger$, m and a

in circulation. The complexity class $A^\dagger$ is also derived from the application use-cases and allows probabilistic variability in the bandwidth saturation for a given aggregator.

C. Experimental results

We first evaluate the behaviour of M3AT for different $A^\dagger$, $B^\dagger$ and $C^\dagger$ data classes and present the results in Table III. We use the default SCIP configuration with 10% relative gap. For this scenario, we fix the number of aggregators to $m = 20$ based on the requirements identified in the use-cases ecosystems analysis. We gradually increase the number of agents in the range of $a \in [20, 200]$ by varying the asymmetry factor $\rho = \frac{a}{m}$, scaled to the order of 10. The reason for this experimental direction is that data aggregation systems typically demonstrate high asymmetry [35] in relation to the number of monitoring agents per aggregator, and is a matter of practical interest to reveal the behaviour of the model in this case.

For $A^\dagger$-class problems, SCIP demonstrates a very fast convergence rate, requiring $100\,\mathrm{ms}$ on average for several hundreds of LP iterations and a negligibly small number of cuts. The memory consumption is around $6\,\mathrm{MB}$ on average. At maximum it takes $224\,\mathrm{ms}$ for $m = 20$ and $a = 200$ with very close 0% exit relative GAP (RGAP), a solver-specific parameter that restricts the optimality degree of the obtained solutions[6]. The $B^\dagger$- and $C^\dagger$-class problems exhibit different behaviour, since they introduce constraint generation based fully on the number of aggregators. For example, it could take more than $100\,\mathrm{s}$ for $m = 20$ and $a = 180$ to reach the optimum with 0% RGAP, while requiring several thousand LP iterations and cuts. Generally within the 10% RGAP limit, the execution time of the solver does not require more than $10\,\mathrm{s}$ for $B^\dagger$- and $C^\dagger$-class problems of 200×20 monitoring agents and aggregators, by consuming $20{-}50\,\mathrm{MB}$ of the RAM.

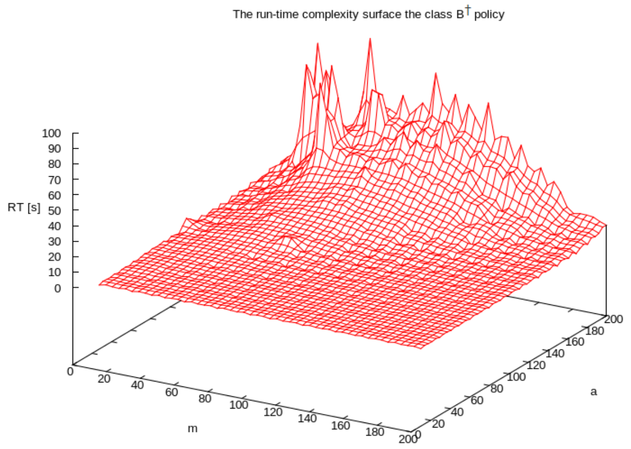

Figure 6: Execution time of SCIP solver with default configuration and 10% relative gap for different combinations of monitoring agents and aggregators.

[6]https://scip.zib.de/doc-6.0.2/html/PARAMETERS.php

Table III: Experimental results for $A^\dagger$, $B^\dagger$ and $C^\dagger$ classes for $m = 20$ aggregators with default SCIP configuration and 10% relative gap. Average SCIP runtime in seconds (ART), average SoPlex LP iterations (ALI) and average SCIP cuts (ACU) computed for a sample size of 50.

	Class $A^\dagger$			*Class $B^\dagger$*			*Class $C^\dagger$*		
a	*ART*	*ALI*	*ACU*	*ART*	*ALI*	*ACU*	*ART*	*ALI*	*ACU*
40	0.056	65	2	1.537	2,364	152	1.332	2,265	138
60	0.077	101	4	1.449	4,846	250	1.747	5,758	274
80	0.090	112	2	2.524	7,166	343	2.766	7,818	361
100	0.137	163	9	3.495	8,294	389	3.690	9,076	399
120	0.146	184	5	4.250	9,076	410	4.491	10,480	425
140	0.145	194	1	4.967	10,172	421	5.100	10,996	429
160	0.170	220	2	5.412	10,258	435	5.771	11,530	431
180	0.190	239	1	5.804	10,552	424	6.556	11,969	438
200	0.224	286	3	6.402	10,868	420	6.962	12,432	419

It is worth mentioning that the $C^\dagger$-class problems demonstrate large initial RGAP of approximately 365% and leap-convergence dynamics in the $2{-}3\,\mathrm{s}$ range to approximately 2%. The best strategy for handling these policies is relaxing the RGAP limit to $10 - 25\%$ to keep the computation within reasonable limits by sacrificing the optimality. Although this precision is not required by this problem, the primary objective was to provide a nearly optimal solutions, while satisfying the required time limits. The execution time includes all phases except LP preparation phase, as it does not dominate over the SCIP execution time on the smaller scales, staying around $15{-}20\,\mathrm{ms}$ for the entire data range.

During the first phase of the experiment, we identified specific solution dynamics for the complexity class $B^\dagger$, which can lead to scalability issues. We decided to explore the runtime complexity for searching the entire space of m and a for the class $B^\dagger$ to better explore the limited scalability of the solver. Figure 6 presents the initial results as the correlation between the runtime of the solver and the variation of the monitoring agents $a \in [20, 200]$ and the aggregators $m \in [20, 200]$, with a relative gap of 10%. In general, the model scales well with the execution time when the number of monitoring agents and aggregators are within the domain not containing high peaks. Practically, in case of I/O limitation policies similar to the class $B^\dagger$ and an asymmetry factor $\rho = \frac{a}{m}$ within the high peaks area (see Figure 6), one needs to relax either the limitation policies or the precision of the assignments.

VIII. Conclusion

In this paper, we described a preliminary monitoring architecture of the pilot ASPIDE project and a mathematical model called M3AT for the optimal assignment of monitoring agents and aggregators intended for large-scale computing systems. We identified set of requirements for the given model, including optimization problem and its solving approaches based on the several data-intensive applications, corresponding ecosystems, and current state-of-the-art discrete optimization techniques. We initially evaluated the scalability of our model based on varying-complexity data sets in relation to the specific SCIP precision configuration. The approach scales well when the number of agents varies within the corresponding boundaries, demonstrating high sensitivity to the problem scale, asymmetry factor and the imposed I/O limitation policy. To further improve the scalability and performance of the model, we will work on defining tightly-connected decomposition techniques, heuristics, architectural practices and fine-grained adaptive tuning strategies, suitable for large-scale optimization problems.

Acknowledgment

This work has been supported by the ASPIDE Project funded by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No 801091.

References

[1] Loris Belcastro, Fabrizio Marozzo, and Domenico Talia. Programming models and systems for big data analysis. *International Journal of Parallel, Emergent and Distributed Systems*, 34:632–652, 2019.

[2] Domenico Talia, Paolo Trunfio, and Fabrizio Marozzo. *Data Analysis in the Cloud*. Elsevier, October 2015. ISBN 978-0-12-802881-0.

[3] 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.

[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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Khalid Alhamazani and et al. An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art. *Computing*, 97(4):357–377, April 2015.

[10] Erika Abraham, Costas Bekas, and et al. Preparing hpc applications for exascale: Challenges and recommendations. In *Network-Based Information Systems (NBiS), 2015 18th International Conference on*, pages 401–406. IEEE, 2015.

[11] 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].

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

[13] 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.

[14] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)*, 7(3):8, 2011.

[15] Lionel Vincent, Mamady Nabe, and Gaël Goret. Self-optimization strategy for io accelerator parameterization. In *International Conference on High Performance Computing*, pages 157–170. Springer, 2018.

[16] Daniel A Reed and Jack Dongarra. Exascale computing and big data. *Communications of the ACM*, 58(7):56–68, 2015.

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

[18] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[19] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[20] Michel Minoux. *Mathematical programming: theory and algorithms*. John Wiley & Sons, 1986.

[21] Silvano Martello. Knapsack problems: algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimiza tion*, 1990.

[22] Temel Öncan. A survey of the generalized assignment problem and its applications. *INFOR: Information Systems and Operational Research*, 45(3):123–141, 2007.

[23] Ibrahim H Osman. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Operations-Research-Spektrum*, 17(4):211–225, 1995.

[24] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16(2):133–151, 2004.

[25] Dirk G Cattrysse and Luk N Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European journal of operational research*, 60(3):260–272, 1992.

[26] Ambros Gleixner et al. The SCIP Optimization Suite 6.0. ZIB-Report 18-26, Zuse Institute Berlin, July 2018.

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

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

[29] Ambros Gleixner et al. The scip optimization suite 6.0. Technical Report 18-26, ZIB, Takustr. 7, 14195 Berlin, 2018.

[30] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[31] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, Fabrice Rossi, and R Ulerich. *GNU scientific library*. Network Theory Limited, 2002.

[32] Brian Gough. *GNU scientific library reference manual*. Network Theory Ltd., 2009.

[33] Silvano Martello and Paolo Toth. A bound and bound algorithm for the zero-one multiple knapsack problem. *Discrete Applied Mathematics*, 3(4):275–288, 1981.

[34] Marshall L Fisher, Ramchandran Jaikumar, and Luk N Van Wassenhove. A multiplier adjustment method for the generalized assignment problem. *Management science*, 32(9):1095–1103, 1986.

[35] Jack Dongarra and Alexey L Lastovetsky. *High performance heterogeneous computing*, volume 78. John Wiley & Sons, 2009.