



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

ANALYSIS OF OPERATIONAL DATA AND DETECTION OF ANOMALIES DURING THE SUPERCOMPUTER JOB EXECUTION

ANALÝZA PROVOZNÍCH DATA A DETEKCE ANOMÁLIÍ PŘI BĚHU VÝPOČETNÍCH ÚLOH NA SUPERPOČÍTAČI

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. PETR STEHLÍK

SUPERVISOR

VEDOUCÍ PRÁCE

doc. Ing. JIŘÍ JAROŠ, Ph.D.

BRNO 2018

Brno University of Technology - Faculty of Information Technology

Department of Computer Systems

Academic year 2017/2018

Master's Thesis Specification

For: **Stehlík Petr, Bc.**

Branch of study: Information Technology Security

Title: **Analysis of Operational Data and Detection of Anomalies during Supercomputer Job Execution**

Category: Data Mining

Instructions for project work:

1. Get acquainted with the architecture of current supercomputers and the software for job management and monitoring.
2. Study data mining techniques applicable on supercomputer operational data.
3. Identify critical metrics describing the actual state of the supercomputer and executed jobs. Design a graphical user interface.
4. Propose a soft computing approach to analyse the job operational data and detect anomalies during the job execution.
5. Implement the proposed solution.
6. Test the proposed solution on available supercomputers.
7. Evaluate the results and discuss benefits of the proposed system in real-world scenarios.

Basic references:

- According to the supervisor's instructions.

Requirements for the semestral defense:

1. Fulfil items 1 to 4.

Detailed formal specifications can be found at <http://www.fit.vutbr.cz/info/szz/>

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Jaroš Jiří, doc. Ing., Ph.D., DCSY FIT BUT**

Beginning of work: November 1, 2017

Date of delivery: May 23, 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
60200 Brno, Božetěchova 2



Lukáš Sekanina

Professor and Head of Department

Abstract

Using the full potential of an HPC system can be difficult when such systems reach the exascale size. This problem is increased by the lack of monitoring tools tailored specifically for users of these systems. This thesis discusses the analysis and visualization of operational data gathered by Examon framework of a high-performance computing system. By applying various data mining techniques on the data, deep knowledge of data can be acquired. To fully utilize the acquired knowledge a tool with a soft-computing approach called Examon Web was made. This tool is able to detect anomalies and unwanted behaviour of submitted jobs on a monitored HPC system and inform the users about such behaviour via a simple to use web-based interface. It also makes available the operational data of the system in a visual, easy to use, manner using different views on the available data. Examon Web is an extension layer above the Examon framework which provides various fine-grain operational data of an HPC system. The resulting soft-computing tool is capable of classifying a job with 84 % success rate and currently, no similar tools are being developed. The Examon Web is developed using Angular for front-end and Python, accompanied by various libraries, for the back-end with the usage of IoT technologies for live data retrieval.

Abstrakt

Tato práce se zabývá analýzou a vizualizací shromážděných provozních dat superpočítače. Použitím různých technik na dolování dat byly získány hluboké znalosti o provozních datech superpočítače monitorovaného systémem Examon. S pomocí těchto znalostí byl vytvořen nástroj se soft-computing přístupem nazvaný Examon Web. Ten je rozšiřující vrstvou systému Examon, která poskytuje různá detailní provozní data HPC systému. Examon Web je schopen rozpoznat anomálie a nežádoucí chování úloh spuštěných na monitorovaném HPC systému a informovat uživatele o tomto chování prostřednictvím webového rozhraní. Examon Web také zpřístupňuje provozní data systému vizuálním a snadno konzumovatelným způsobem, přičemž používá různé pohledy na dostupná data. Výsledný nástroj je schopen klasifikovat úlohu do dvou tříd s úspěšností 84 %. Examon Web byl vyvinut pomocí frameworku Angular pro front-end a Pythonu, doprovázeného různými knihovnami, pro back-end s využitím IoT technologií pro získávání aktuálních provozních dat superpočítače.

Keywords

big data, neural networks, deep learning, high performance computing, HPC, anomaly detection, web, GUI, back-propagation, decision trees, Angular, Python, Cassandra, KairosDB, MQTT, Internet of Things, IoT, WebSocket

Klíčová slova

big data, neurální sítě, hluboké sítě, superpočítač, HPC, detekce anomálií, web, GUI, back-propagation, rozhodovací stromy, Angular, Python, Cassandra, KairosDB, MQTT, Internet of Things, IoT, WebSocket

Reference

STEHLÍK, Petr. *Analysis of Operational Data and Detection of Anomalies During the Supercomputer Job Execution*. Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Jiří Jaroš, Ph.D.

Rozšířený abstrakt

V posledních několika letech jsme téměř na dosah superpočítače s výkonem přes 1 exaFLOP. To znamená, že superpočítače jsou stále větší a složitější, s čímž souvisí problém využití plného potenciálu takového systému. Tento problém se umocňuje díky nedostatku nástrojů pro monitorování, které jsou specificky přizpůsobeny uživatelům těchto systémů. Cílem této práce je vytvořit nástroj, nazvaný Examon Web, pro analýzu a vizualizaci shromážděných provozních dat superpočítače a provést nad těmito daty hloubkovou analýzu pomocí neurálních sítí. Ty určí, zda daná úloha běžela korektně, či vykazovala známky podezřelého a nežádoucího chování jako je nezarovnaný přístup do operační paměti nebo např. nízké využití alokovaných výpočetních zdrojů. O těchto zjištěných faktech je uživatel informován pomocí grafického uživatelského rozhraní. Examon Web je postavený na frameworku Examon, který sbírá a procesuje metrická data ze superpočítače a následně je ukládá do databáze určené pro velká data–KairosDB. Implementace této práce zahrnuje mnoho disciplín od návrhu a implementace GUI, přes rozsáhlou datovou analýzu, těžení dat a neurální sítě až po implementaci různých rozhraní na serverové straně. Examon Web je zaměřen zejména na uživatele superpočítače, ale může být také využíván systémovými administrátory. GUI je vytvořeno ve frameworku Angular společně s knihovnamí Dygraphs a Bootstrap. Uživatel díky tomu může jednoduše analyzovat časové řady různých metrik své úlohy spuštěné na superpočítači a stejně jako administrátor se může informovat o současném stavu celého superpočítače. Tento stav je zobrazen jako několik globálně agregovaných metrik v posledních 30 minutách nebo jako 3D model (či 2D model) celého superpočítače, který získává živá data ze samotných uzlů superpočítače pomocí protokolu MQTT. Pro kontinuální získávání dat bylo využito rozhraní WebSocket, ve kterém byl implementován vlastní mechanismus přihlašování a odhlašování konkrétních metrik zobrazovaných v modelu. Při analýze spuštěné úlohy má uživatel dostupné tři různé pohledy na danou úlohu. První pohled nabízí celkový přehled o úloze a informuje uživatele o využitých zdrojích, času běhu a celkovém vytížení části superpočítače, kterou úloha využila společně s informací z neurálních sítí o podezřelosti úlohy. Další dva pohledy zobrazují jednotlivé metriky z výkonnostního energetického hlediska. Pro naučení neurálních sítí bylo potřeba vytvořit novou datovou sadu tvořených daty ze superpočítače Galileo. Tato datová sada obsahuje přes 1100 úloh spuštěných a monitorovaných na tomto superpočítači z čehož 500 úloh bylo ručně anotováno a následně použito pro trénování neurálních sítí. Neurální sítě využívají model back-propagation, který je vhodný pro anotování časových sérií fixní délky. Celkem bylo vytvořeno 12 sítí pro metriky zahrnující vytížení procesoru, paměti a dalších částí a například také podíl celkového času procesoru v úsporném režimu C6. Tyto sítě jsou na sobě nezávislé a po experimentech jejich finální konfigurace 80-20-4-3-1 (80 vstupních až 1 výstupní neuron) podávaly nejlepší výsledky. Poslední síť (v konfiguraci 12-4-3-1) anotovala výsledky předchozích sítí, kde na vstup této sítě jsou přiloženy výsledky předešlých sítí. Celková úspěšnost celého systému klasifikace do 2 tříd je 84 %, což je na použitý model velmi dobrá úspěšnost. Výstupem této práce jsou dva produkty. Prvním je uživatelské rozhraní a jeho serverová část Examon Web, která jakožto rozšiřující vrstva systému Examon pomůže s popularizací a rozšířením daného systému mezi další uživatele či přímo další superpočítačová centra. Druhým výstupem je částečně anotovaná datová sada, která může pomoci dalším lidem v jejich výzkumu a je výsledkem spolupráce VUT, UNIBO a CINECA. Oba výstupy budou zveřejněny s otevřenými zdrojovými kódy. Examon Web byl prezentován na konferenci 1st Users' Conference v Ostravě pořádanou organizací IT4Innovations. Dalším rozšířením této práce může být kompletní anotace datové sady a také rozšíření Examon Web o rozhodovací stromy, které určí přesný důvod špatného chování dané úlohy.

Analysis of Operational Data and Detection of Anomalies During the Supercomputer Job Execution

Declaration

Hereby I declare that this master thesis was prepared as an original author's work under the supervision of Dr Jiří Jaroš. The supplementary information was provided by Dr Andrea Bartolini, et. al. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Petr Stehlík
May 21, 2018

Acknowledgements

I would like to thank my supervisor Dr Jiří Jaroš for leading this thesis and Dr Andrea Bartolini and his team (namely Dr Francesco Beneventi and Dr Andrea Borghesi) for advising this thesis. This work was done with the usage of facilities of CINECA located in Bologna, Italy and IT4Innovations in Ostrava, Czech Republic. The base of Examon Web was done during the PRACE Summer of HPC internship in CINECA.

This work was supported by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center – LM2015070".

Contents

1	Introduction	4
2	Related Work	6
2.1	HPC System Monitoring	6
2.2	Job Monitoring	9
2.3	Job Classification	10
3	Theoretical Background	11
3.1	High Performance Computing	11
3.1.1	Top-down analysis	11
3.2	Job Scheduling	14
3.3	HPC System Monitoring	14
3.3.1	IPMI	14
3.3.2	Examon	15
3.4	Data Mining Techniques	17
3.5	Neural Networks	17
3.5.1	Decision Tree	18
3.5.2	Backpropagation Network	18
4	Design	21
4.1	Target Audience & Users	21
4.2	Examon Web	22
4.2.1	Front-end	22
4.2.2	Back-end	24
4.3	Job Anomaly Detection	25
4.3.1	Data	25
4.3.2	Data Acquisition	26
4.3.3	Data Labelling	27
4.3.4	Data Processing	27
4.3.5	Metric Networks	27
4.3.6	Job Network	28
5	Implementation	29
5.1	Graphical User Interface	30
5.1.1	Homepage	30
5.1.2	User Management	30
5.1.3	Job Module	31
5.1.4	System Module	34

5.1.5	Cluster Module	34
5.1.6	Front-end Adaptations	36
5.2	Back-end	36
5.2.1	REST API	37
5.2.2	MQTT and WebSocket Communication	40
5.3	Dataset Creation	43
5.4	Job Classification	45
5.5	Job Anomaly Classification	46
5.6	Summary	47
6	Conclusions	49
6.1	Current Deployment	49
6.2	Contributions & Impact	50
6.3	Further Work	50
	Bibliography	51
A	Appendices	55
A.1	PBS Pro Hooks Lifecycle	55
A.2	Examon Web Job Info Wireframe	56
A.3	Examon Web Cluster Overview Wireframe	57
A.4	Example Jobs Spotted During Labelling	58
A.5	Networks' Error Rates	62
A.6	Poster for IT4Innovations' 1st User Conference	63
A.7	Contents of the Attached Media	64

Acronyms

μ Ops micro-operations. 11–13

BMC baseboard management controller. 14, 15

CPU Central Processing Unit. 4, 6, 10–12, 14, 25, 32, 34, 36, 38, 43, 47, 49

CQL Cassandra Query Language. 24

GPU Graphics Processing Unit. 4, 16, 32

GUI Graphical User Interface. 5, 27, 30

HPC High Performance Computing. 4, 6, 8–11, 13–15, 21, 25, 26, 49, 50

I2C Inter-Integrated Circuit. 16

IoT Internet of Things. 6

IPMI Intelligent Platform Management Interface. 14–16

IPS instructions per second. 25

MIC Many Integrated Core processor architecture. 16

MQTT Message Queue Telemetry Transport. 2, 6, 15, 16, 22–25, 36, 37, 39–43, 49

PMBus Power Management Bus. 16

PMU Performance Monitoring Unit. 12, 13, 15, 16

REST REpresentational State Transfer. 22–24, 26, 30, 36–39, 47, 49

SPA single-page application. 22

TLB translation lookaside buffer. 13

WMA weighted moving average. 25

Chapter 1

Introduction

In recent years we are getting closer and closer towards exascale computing. With this goal in sight, the supercomputing systems are getting bigger, more powerful and more complex. These facts make it difficult to fully utilize the whole potential of the computing resources in the most efficient manner.

Future **HPC** systems will feature thousands of nodes each fitted with tens or even hundreds of **CPU** cores, large memory and a wide range of accelerators or **GPU**s. These systems must be connected via complex inter-node communication networks designed in intricate schemes such as N-dimensional torus or hypercubes [41]. From this point of view, it is getting harder to operate **HPC** systems at their peak performance. Another side of this problem is power consumption and energy efficiency. Hardware vendors are producing extremely efficient chips and other accompanying components but this is a solution only for a fraction of the problem. The quest towards the exascale supercomputer requires precise control of the energy and power consumed by the nodes and their components but also to precisely control the environment, mainly the cooling infrastructure, in which the system is operated.

One of the first things which can help with these problems is to give the users and system administrators of such facilities a definite place where they can easily analyze the utilization, energy consumption, performance and status of their executed tasks or the whole system in easy-to-consume visual manner.

In order to obtain and gather operational data of a supercomputer system Examon framework [4] is used. The Examon framework serves as the base for the tool developed and evaluated in this thesis called Examon Web. Examon is a fine grain monitoring framework which collects and handles a wide set of sensors and performance counters of the cluster computing resources, job scheduling data and infrastructure metrics all sampled at a fine granularity. With Examon Web, users and system administrators will be able to analyze operational data of **HPC** systems and jobs running on them using and combining data from the Examon framework. Examon Web will also provide detection of anomalies and poor performance during job execution using neural networks which can classify whether a job ran well or if it is in some way suspicious of unwanted behaviours such as poor performance or execution failure.

The system administrators are provided with many tools from hardware and software vendors of their systems which provide detailed information about the system and its parts but, usually, all this information is difficult to process in real-time by a single person.

Every user of a supercomputer needs to know whether their submitted job finished successfully and performed well. So far this tedious task is usually performed manually

using only the output of their program and over-simplified metrics such as job runtime and total utilized resources.

Both of these problems can be solved by Examon Web. For system administrators, it will provide a simple interface to oversee the status of their cluster in a single view. Users will have the option to monitor their jobs in real-time while Examon Web will provide useful information about the job and its status with the ability to inform users about possible problems with the job.

The goal of this thesis is to create a tool which will analyze operational data of a supercomputer and based on the gathered data detect anomalies during job execution. This information will be presented to a user via a **GUI** tailored for this tool.

The structure of this thesis is as follows: in Chapter 2 Examon Web is compared and aligned with currently available tools, the theoretical background needed for this thesis is presented together with the description of Examon framework in Chapter 3. In the following Chapter 4 the proposed design of the whole Examon Web is presented together with target audience and users. Afterwards, in Chapter 5 the implementation Examon Web and its underlying systems (namely neural networks) are laid out and in the last Chapter 6 the achieved results, summary and further work plans are discussed.

Chapter 2

Related Work

Nowadays there are several tools and libraries for collecting data from nodes and their components. CPUs embed performance counters accessible via software libraries in operating system structures and dedicated instructions inside them [2].

Using the read values a set of architectural, physical and performance quantities can be measured and evaluated. The top-down analysis introduced by Yasin, et. al. [57] uses specific counters to understand applications' bottlenecks using top-down analysis approach starting from a narrow set of metrics. It is possible, by looking at the microoperations flow inside CPU's pipeline, to detect and identify where the executed program was bound.

The related work is separated into three sections each presenting a specific field discussed and used in this thesis. Section 2.1 is about how an HPC system can be monitored and the state of the art tools are mentioned including data visualization tools as well. In Section 2.2 the related work in job monitoring is discussed and in the last Section 2.3 current state of classification of jobs run on HPC systems is shown.

2.1 HPC System Monitoring

There are several approaches to gather performance and energy metrics from HPC systems on a user-level basis. Usually, these approaches require user's intervention and scripting such as PAPI [12], Intel vTune [45], Linux perf tools [32] or Intel Performance Counter Monitor [11] and introduce interference on the program's execution. Moreover, these approaches cannot be used in a reasonable way for continuous monitoring deployed system-wide.

Beneventi, et al. [4] present the Examon (exascale monitoring) framework to overcome this limitation by wrapping the above-mentioned profiling libraries in a modular and extendable framework for accessing the performance metrics with a regular sampling. The collected data are then propagated to a scalable data handling back-end based on Internet of Things (IoT) and big data technologies.

Stefanov, et al. [49] introduced a monitoring framework called DiMMon as well. The framework is designed as a distributed modular system with heterogeneous agents which can measure and process different metrics or direct data flow of measured data.

If we compare Examon and DiMMon, Examon is built on top of the Internet of Things technologies (i.e. MQTT [28]), whereas DiMMon is a true agent-based system with custom message passing using UDP packets and custom message protocol in between its agents inside the system. This means the Examon's architecture is more homogenous and easier to maintain than DiMMon's as seen in Figure 2.1. Also from the overhead standpoint,

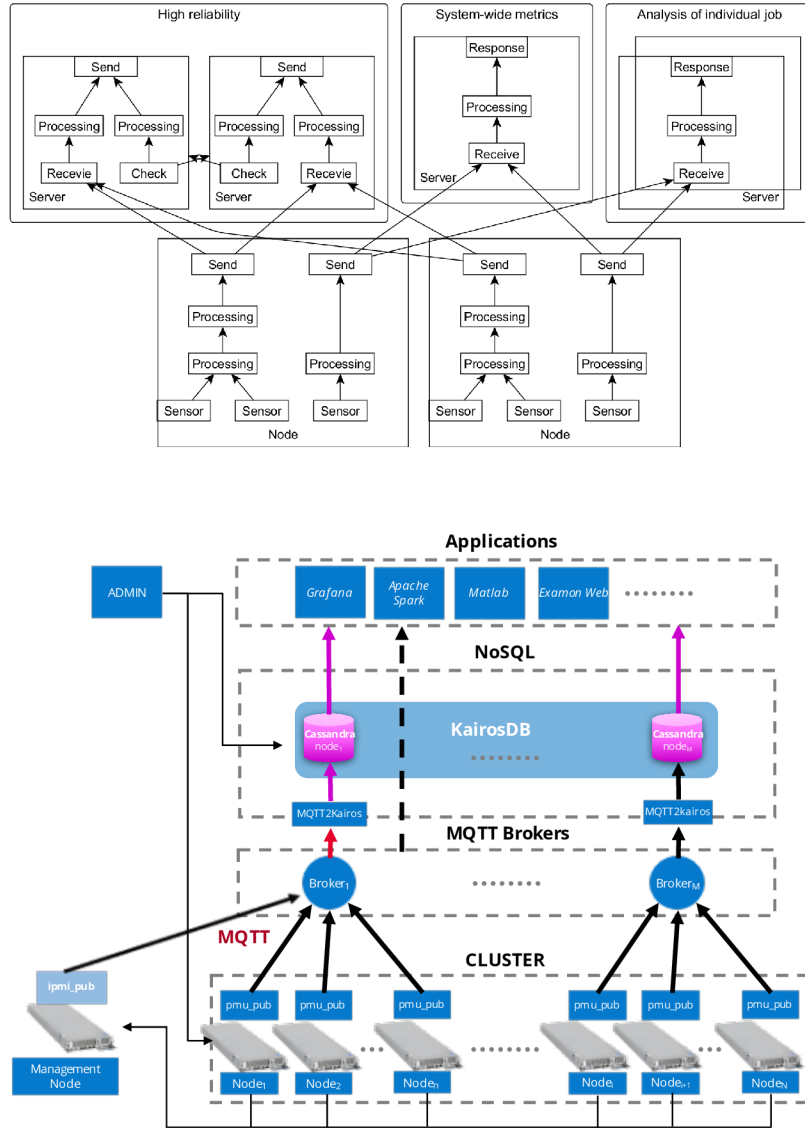


Figure 2.1: A side-by-side comparison of DiMMon (upper) and Examon (lower) architectures. The main difference between these tools is the approach to distribution of tasks. DiMMon takes advantage of heterogeneous agents which together complete the needed task such as data collection or data-processing. On the other hand, Examon deploys homogeneous agents across the whole cluster which send the collected data to one central point (big-data database) where data are processed as needed by users. Figures taken from [49] and [4] respectively.

Examon is more effective using below 1 % of system resources compared to DiMMon’s 3 % utilization.

Beneventi, et. al also discuss storage and other processing in their paper which is, in these days, a crucial part of analytics. Stefanov, et. al had another paper [54] published on utilizing collected data to analyze and classify job behaviour.

System-wide monitoring systems are available and widely used in today’s HPC facilities not just as a research work used only on a couple of facilities but also as established projects used across the world’s HPC facilities. One of best-known tools is Ganglia [29].

Ganglia is a scalable distributed monitoring system for HPC systems. Ganglia agents retrieve data from several sensors and store it on a per-metric basis using RRDtool [38]. Its architecture is based on a hierarchical design which relies on multicast-based protocol in listen/announce manner. It uses a tree of point-to-point connections amongst cluster nodes in order to collect the nodes’ states. Its main drawback is the lack of a simple way to combine information from different metrics and a not so easy to use web interface.

Having collected different kinds of data, the next logical step is to present and visualize it to the users in a meaningful and comprehensible way. A survey of surveys was conducted of the state-of-the-art data analysis and visualization tools [30] from which we can infer the best-known tools for visualization.

For performance visualization tools in large-scale data centres, we refer to [23] which discusses the state of art performance visualizations and [13] which discusses large-scale trace data analysis and visualization techniques. Taking into account the work of Beneventi, et al. on Examon, they use Grafana [37], an open source and flexible framework for time-series data visualization on the web. Even though Grafana is one of the best tools to visualize this kind of data, it lacks two major features in case of performance analysis: 1) no intuitive way to combine multiple data sources together (job and sensor data) to create per-job views; 2) complex initial setup for views able to handle tens or hundreds of sensor data streams needed for example to observe trends on the whole machine, therefore rendering Grafana useless for system overview use-cases.

Splunk [6] and Elasticsearch [19] are tools for visualization and analysis of various machine data. These tools are general in their nature and require difficult setup and initial time investment in order to fully utilize all their functions. With this in mind, Examon Web will provide an easy-to-use and comprehend interface while requiring minimal setup.

Monitoring of certain metrics that are also handled by Examon can be done using tools such as Nagios [1] or Zabbix [39]. These tools provide automatic alerting based on thresholds and monitoring of various metrics ranging from hardware to software services. At first, the goal of Examon Web can be seen similar to these tools but Examon Web will aim for different usages. Nagios or Zabbix can alert system administrators about a potential problem but Examon Web will aim to help with identifying the problems and why they happened and the ideal scenario will be to use one of these tools and Examon Web together.

Showerman [48] proposes a set of visualization approaches applied to data collected on an HPC system. Data collected from several sensors with a sampling time of one minute is stored in a database, to be later analysed and visualized. This work is orthogonal to a part of this thesis since it proves the benefit of observing data coming from a wide range of sources. On the other hand, the work of Showerman lacks a real-time monitoring capability. Moreover, the visualization approaches require a direct interaction with the database and are therefore much less accessible than Examon Web.

Gimenez, et al. [17] present a tool for memory performance visualization and analysis. The main goal is to help users to optimize their application since memory usage is often the major bottleneck for HPC applications. Their work is exclusively focused on memory (and related measurements) while Examon Web will be able to handle a wide range of different metrics other than memory.

2.2 Job Monitoring

When it comes to monitoring and mainly managing jobs in HPC facilities, a couple state-of-the-art tools are usually taken into account. Slurm [25], PBSPro [55] and TORQUE-based [10] systems such as MOAB HPC Suite [9]. The MOAB is specifically targeted for enterprise usage and therefore is not considered in this thesis as it is not suitable for educational and scientific purposes.

Slurm and PBSPro are very much alike with the difference Slurm is rather plugin-based with over 100 plugins available while PBSPro is shipped with many features. Historically PBSPro (or its predecessors) were widely used but in recent years many HPC facilities are turning over to Slurm [8].

Using job managers to monitor jobs on user level can be done in various ways. The most direct approach in many cases is manual monitoring of jobs using commands available through a local job manager. On the other hand, the easiest and most user-friendly way is to automatically monitor jobs, collect data about them and present collected data to users.

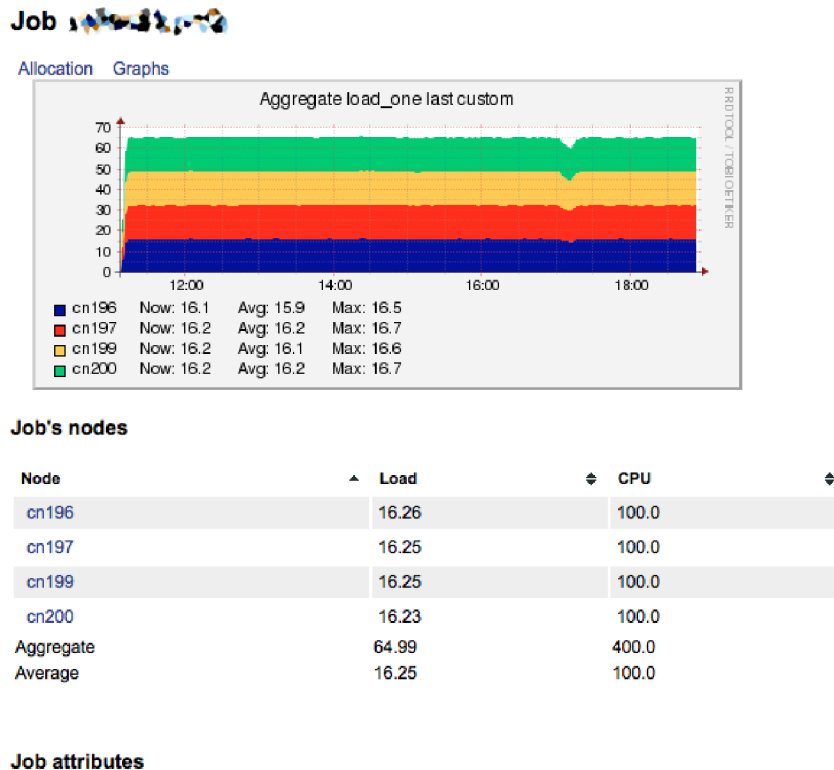


Figure 2.2: Anonymized job overview extracted from IT4I Extranet interface [24].

One can find tailored tools for job monitoring in **HPC** facilities around the world such as Extranet back-office interface at IT4Innovations in Ostrava, Czech Republic [24] which can be seen in Figure 2.2. No specific research was found on this topic and therefore we can conclude that work done in this thesis regarding job monitoring can be seen as innovative. One indirect way of monitoring jobs can be done using Ganglia tool and custom-built plugins but no specific example was found and this approach is only theoretical.

2.3 Job Classification

Only minor amount of work has been done to classify supercomputer jobs based on their behaviour. Voevodin et. al. [54] used random trees and statistical info about their jobs—median and oscillation rate. Metrics contained L1 and L3 cache misses per second, system and **CPU** load and other. Using this limited set of features they were successful in job classification into 3 groups: normal, abnormal and suspicious.

This thesis focuses on a similar goal with the extension of providing insights on why a suspicious job was labelled suspicious. The same target is discussed in further work but no other work was found on this topic.

The paper's target group are system administrators only which are sent a daily report of suspicious jobs while this thesis will focus on users rather than system administrators in order to help users create more effective programs.

Chapter 3

Theoretical Background

The base of theoretical background needed to design and implement the Examon Web is presented in this chapter. First, a basic description of an **HPC** system is shown together with the top-down performance analysis and how an **HPC** system can be monitored. Section 3.2 describes how jobs submitted to a supercomputer are managed and how the jobs can be monitored. In the Section 3.4 techniques required to mine useful knowledge of the data gathered using Examon framework are presented, and in the last Section 3.5, back-propagation neural network and decision trees are described as these two neural network models are used in Examon Web.

3.1 High Performance Computing

Supercomputers differ in many ways compared to general-purpose computers but also share a lot in common. Here, we will focus on the differences. A supercomputer is a massive computer with a high level of computing performance designed to undertake on massively parallel tasks. Supercomputers are built with up to tens of millions of **CPU** cores with enough operational memory for each core. The basic unit of a supercomputer is a node. Each node can contain different hardware and nodes that are similar and near themselves create partitions. All nodes are interconnected in a specific manner. For interconnection special network architectures were created such as InfiniBand [43] or Intel Omni-Path [5].

Only Intel **CPUs** are taken into account in this thesis since the majority of current supercomputers uses Intel-based **CPUs**. The top-down analysis description and figures are sourced from [22, 57].

3.1.1 Top-down analysis

To make applications take advantage of **CPU** microarchitectures, we need to know how the application is utilizing available hardware resources. Modern **CPUs** use pipelining, hardware threading, out-of-order execution, instruction-level parallelism or speculative branching to fully utilize available resources. Even with these features, we can find constructs such as linked data structures with indirect addressing, that result in inefficiencies. This behaviour commonly causes many idle instructions in the **CPU** pipeline while waiting for data to be retrieved and no other instructions available to execute in the meantime.

In order to fully understand the hardware pipeline of a modern **CPU**, it is better to divide the pipeline into two parts, front-end and back-end. Front-end fetches instructions of a program and decodes it into low-level hardware operations called **micro-operations**

(μ Ops). The μ Ops are then sent to the back-end’s allocation unit and then executed once an execution unit is available. The moment when a μ Op finishes is called retirement and during the retirement results of the μ Op are committed to CPU registers or written back to memory.

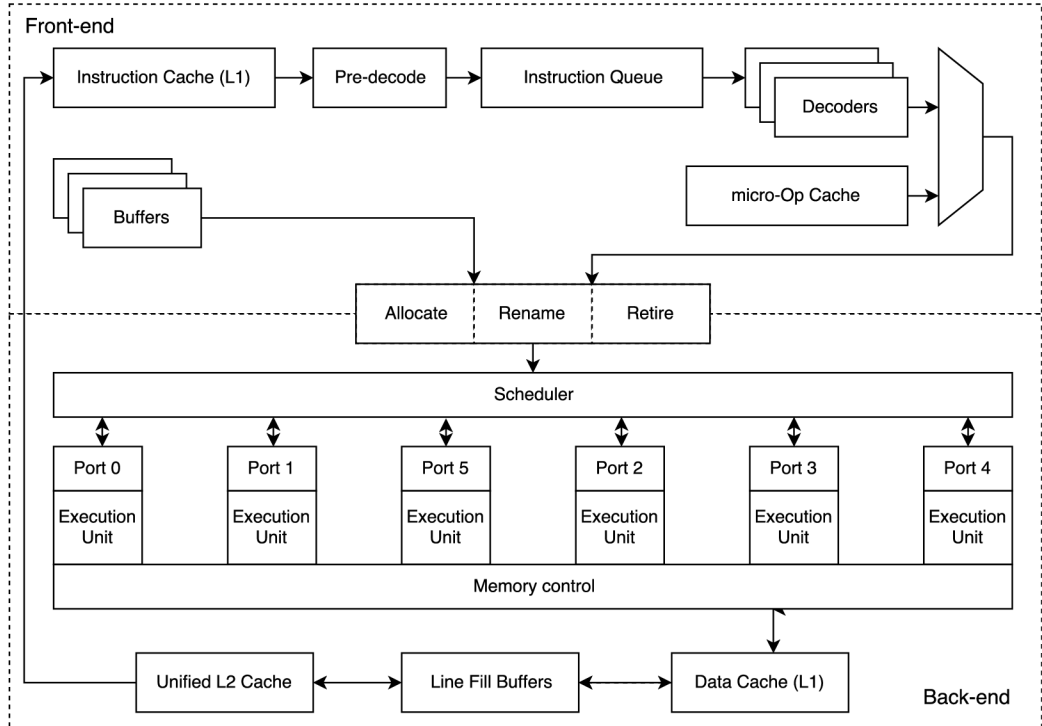


Figure 3.1: Schematic of a CPU microarchitecture divided into front-end and back-end parts.

First, we need to define an abstract concept that represents the hardware resources needed to process one μ Op–pipeline slot. The top-down analysis assumes there are several pipeline slots available for each CPU core, at each clock cycle. We can use specially designed on-chip Performance Monitoring Unit (PMU) to analyze how well the pipeline slots are utilized.

PMUs are specific pieces of logic specially dedicated to performance monitoring. The monitoring is done by counting specific hardware events happening on the system. Exemplary events can be cache misses, branch mispredictions or μ Ops retirement. When specific events are combined we can calculate high-level metrics such as cycles per instruction (CPI). Each microarchitecture makes available slightly different PMUs but overall the number of PMUs is in hundreds. There also exist predefined events and metrics useful for top-down analysis in order to turn this information into useful knowledge about the program’s performance issues.

The status of pipeline slots is sampled at the allocation point right in between the front-end and back-end borders. An allocation point is a place where μ Ops leave front-end and enter the back-end pipeline.

We can derive four possible categories of an empty pipeline slot based on the simplified pipeline as seen in Figure 3.1 causing a stall in the CPU pipeline. Each item in the list is

annotated with an expected range of hotspots in a well-tuned **HPC** application. The figures are taken from [22].

- retiring (30–70 %): μOp successfully retires
- back-end bound (20–40 %): front-end has a μOp ready but can't deliver it because the back-end isn't ready to handle it
- front-end bound (5–10 %): front-end's inability to fill the slot with a μOp
- bad speculation (1–5 %): μOp doesn't retire because of incorrect branch prediction or due to a clearing event

These categories (as seen in Figure 3.2) cover the top level of top-down analysis and are the most crucial in determining the bottlenecks in programs. Each of them is calculated using a specific set of **PMUs**. The Examon framework takes care of this and can be abstracted in this text.

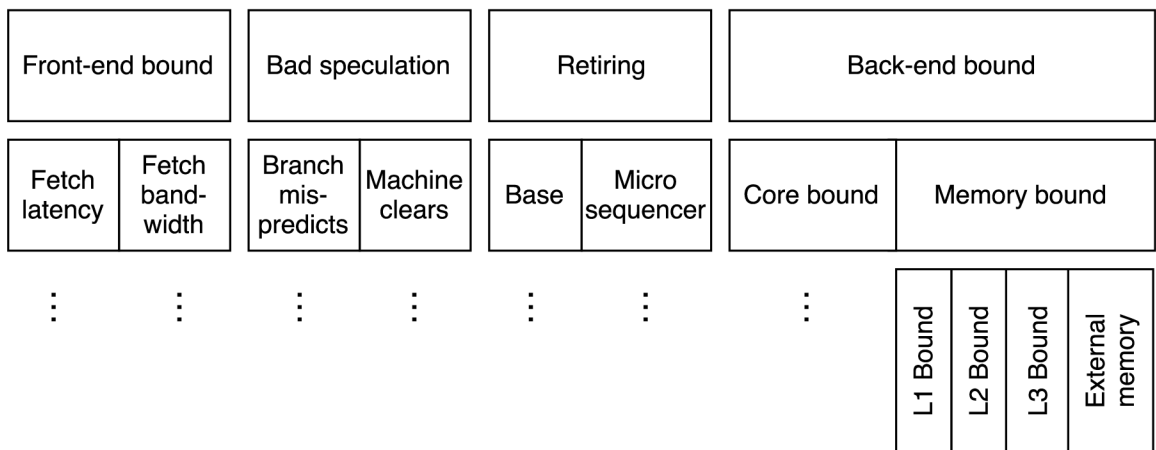


Figure 3.2: Top-down analysis hierarchy with only major categories and levels shown. Adapted from [57].

Front-end bound pipeline slots can be separated into two categories, fetch latency and fetch bandwidth. The latter is linked with cache and **TLB** misses. The former is caused when there is high instructions per second (IPC) count and therefore can dominate the performance.

Bad Speculation includes pipeline slots wasted because of incorrect speculations. We can distinguish between slots that don't retire and slots in which the pipeline was blocked due to recovery from bad speculations.

Retiring slots are again split into two categories. The successfully retired μOps are denoted as a base and the general goal is to achieve having all slots in this category. Microsequencer category labels μOps such as floating point assists that lower performance but still, these μOps retire successfully.

Back-end bound category consists of memory and core bounds. Memory bound stalls are caused by the memory subsystem meaning mainly cache misses at different levels. Core bound stalls mean short starvation periods or uneven execution ports utilization.

3.2 Job Scheduling

Scheduling a job on an **HPC** system is a crucial step in a job execution. If a job requires a larger portion of the **HPC** system it might even take days to allocate that many computational resources. The job is submitted to a queue with a calculated priority. The calculation of priority includes many parameters which are set up by system administrator depending on their preferences but usually the larger the job, the lower the priority. Basic features of current state-of-the-art schedulers include:

- define workflows or dependencies via interfaces
- automatic executions
- monitor the job's execution via API
- different queues and priorities to control execution order

In this thesis, monitoring a job's execution is extensively used. Other features are rather user-related and heavily used by everyday **HPC** users. Monitoring a job is usually done in a rude and simple way (i.e. only checking exit status and total running time).

The schedulers that are widely used across system include PBSPro [55] and Slurm [25]. In the former, to access monitoring information specially designed plugins called *hooks* can be used. The latter provides an interface to create plugins which can obtain such information.

An example of a job's lifetime events can be seen in the Appendix A.1 from which various data can be obtained such as allocated resources, queueing time, user info or job parameters.

3.3 HPC System Monitoring

In this section, we describe how an **HPC** system can be monitored and what tools can be used. Examon framework is presented in great detail with emphasis on data storage and accessibility. Only technologies designed for system monitoring are presented here but Examon also utilizes non-standard ways of monitoring using performance counters located on **CPUs** which have been shown in Section 3.1. This section is mainly focused on the Examon framework which is heavily used throughout this thesis.

3.3.1 IPMI

IPMI (Intelligent Platform Management Interface) [34] is a set of interface specifications for system management and monitoring. In terms of **HPC IPMI** allows querying node level statistics such as power consumption, utilization and temperature at different locations on the motherboard. **IPMI** is completely independent of the host's **CPU**, firmware or OS and the communication is done using out-of-band (LAN) network. The most used scenario is to power on a node remotely using only **IPMI** without the need for direct access to other node's hardware.

IPMI's architecture consists of several modules of which the **BMC** (baseboard management controller) is the centrepiece of all. It provides the intelligence to the whole **IPMI** architecture and is a specialized microcontroller embedded on the motherboard. Various sensors such as temperatures, cooling fan speeds, power or OS status report to **BMC** via

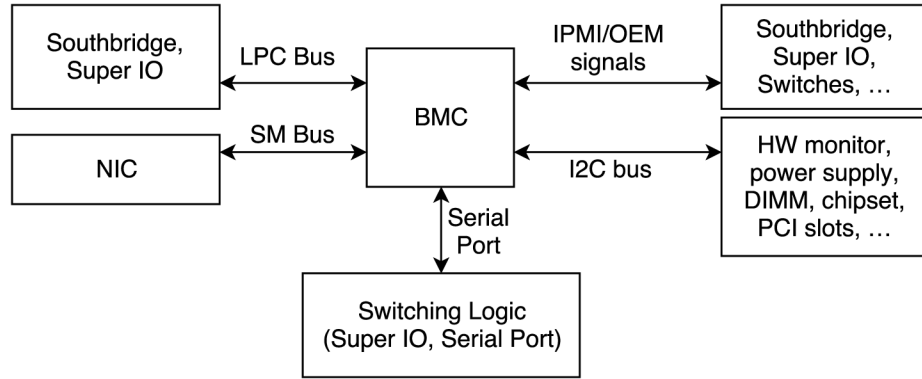


Figure 3.3: Basic IPMI architecture showing BMC's interfaces for communication with vendor's hardware.

different bus interfaces as seen in Figure 3.3. Using this data, we can determine the state of the monitored node, control it, do basic diagnostic tasks and much more.

3.3.2 Examon

Examon is a highly scalable framework for performance and energy monitoring of HPC systems developed at UNIBO [51]. It collects and processes various monitoring data from several sources. Its architecture is separated into several layers and described in this section.

In Figure 3.4, we can see the following layers (from bottom): 1) at cluster level, we have data collection agents (PMU, IPMI and job scheduler), 2) data transport layer realized via MQTT [27], 3) database layer which consists of KairosDB and Cassandra and 4) the application layer which connects to the database layer or transport layer (MQTT) and Examon Web will reside in this layer.

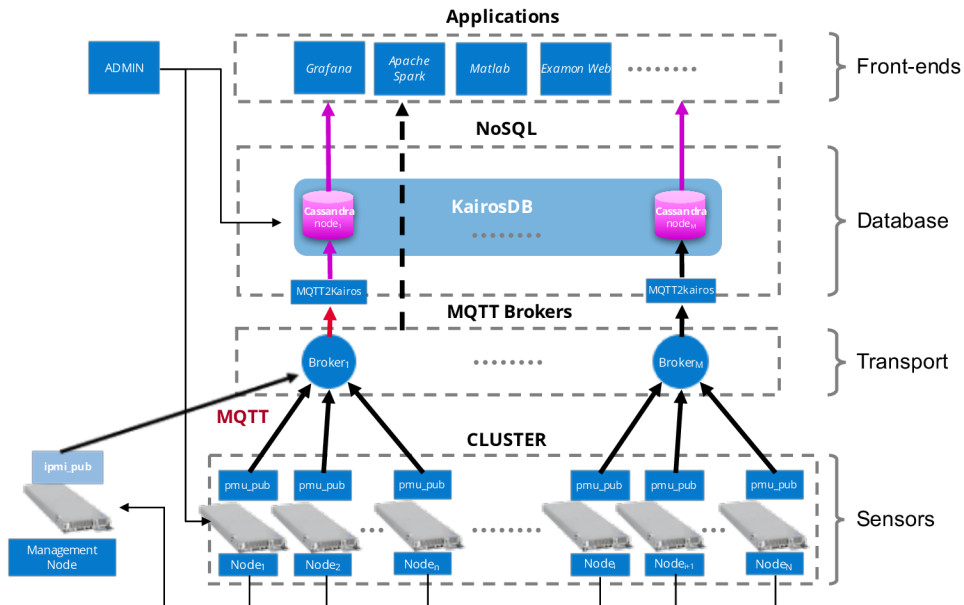


Figure 3.4: High-level architecture of Examon framework split into layers.

Data Collection

Examon collects two types of data: 1) physical sensor measurements and 2) workload data obtained from the job manager. The agents running on nodes collect various sensor data scattered across the system and publish it in a uniform format `<unix timestamp>;<value>` via **MQTT** to the upper layer of the stack. They are composed of two APIs, **MQTT** and Sensor API. The former implements the **MQTT** protocol functions and it is the same among all the collectors while the latter implements custom sensor functions related to the data sampling and is unique for each kind of collector. Considering the specific sensor API object, we can distinguish collectors that have direct access to hardware resources like **PMU**, **IPMI**, **GPU**, **MIC**, **I2C** and **PMBus** and collectors that sample data from other applications as batch schedulers (PBS and Slurm) and tools such as perf, PAPI, and PCM.

The second type of data regards the jobs running in the system and its workload. To collect such data, the job scheduler is extended by a plugin that collects this data and publishes them via **MQTT** in JSON format.

Communication Layer (**MQTT**)

MQTT (Message Queuing Telemetry Transport) protocol implements the “publish-subscribe” messaging pattern and requires three different agents as seen in Figure 3.5.

The *publisher* has the role of sending messages on a set topic to a predefined broker. The *subscriber* subscribes to certain topics at a broker and waits for incoming messages. The *broker* has the functions of receiving data from publishers, making topics available and delivering data to subscribers. Basic **MQTT** communication mechanism is as follows. When a publisher agent sends some data having a certain topic, the topic is created and managed at the broker and any subscriber to that topic will receive the associated data as soon as available to the broker. In terms of Examon, collector agents have the role of “publishers”.

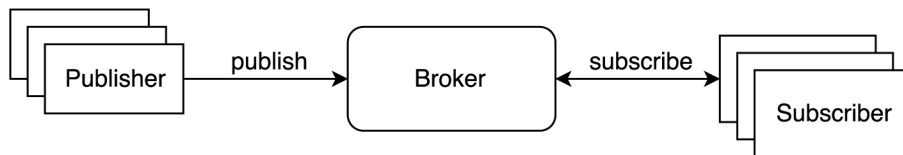


Figure 3.5: Publisher-subscriber **MQTT** model.

Storage Layer

Examon stores collected data mainly for visualization and analytical purposes. It uses a distributed and scalable time series database (KairosDB) that is built on top of a NoSQL database (Apache Cassandra [26]) as back-end. An **MQTT** subscriber (MQTT2Kairos) was implemented to provide a bridge between the **MQTT** datastream and the KairosDB data insertion mechanism. MQTT2Kairos takes advantage of predefined **MQTT** topics structure to automatically form the KairosDB insert statements and eventually queries as well.

Applications Layer

The data gathered by Examon can serve multiple purposes, as mentioned in the application layer. For example, machine learning techniques can be applied to extract predictive models or devise online fault detection mechanisms as discussed in this thesis. Another usage is real-time visualization on the web as extensively described in Chapter 4.

3.4 Data Mining Techniques

The basic techniques required to obtain meaningful data for further analysis are described in this section. Data mining [21] is a crucial step in developing a working classification model and understandable visualizations. The process of data mining can be seen as a pipeline of several individual steps shown in Figure 3.6.

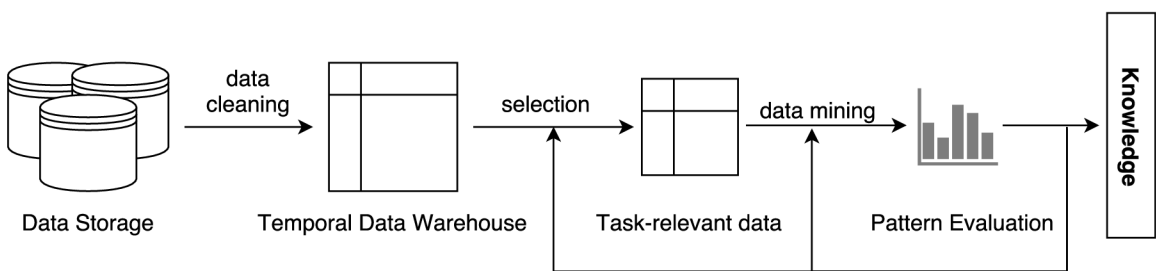


Figure 3.6: Data mining pipeline showing several stages of data during the data mining process.

We begin at the data storage where all collected raw data reside. With the help of various querying mechanisms, the raw data is cleaned of incomplete, corrupt and irrelevant records. This process is generally known as *data cleaning*. The cleaned data is then stored in a temporal data warehouse such as in-memory storage structures (i.e. lists or dictionaries) but can also be stored permanently for further analysis.

Next is *data selection* where only relevant data is selected for next steps. This step can be done multiple times in order to precisely select only extremely relevant data in case of large datasets.

During data cleaning and selection we can do further data processing such as noise reduction, attribute construction (derive new attributes from existing data), aggregation, generalization and normalization in order to provide better data for the next step.

What follows is the process of *data mining* itself where we extract insightful knowledge from pattern evaluation. In the context of this thesis when we talk about data mining we do data classification. Other types of data mining are data characterization, clustering or evolution analysis.

3.5 Neural Networks

Section 3.4 showed data mining techniques which often require the help of neural networks. In this section, we present best-known and widely used neural networks for classification (decision tree) and anomaly detection (backpropagation network). The sources used in this section for equations and descriptions cover [31, 18].

3.5.1 Decision Tree

A decision tree is a decision support tool that uses the properties and values of features that are classified. A decision tree resembles a tree-like structure as shown in Figure 3.7 where nodes in the tree can be thought of as units asking a yes/no question and the subtree is selected based on the answer. The decision tree has a definitive number of layers where the last one is the final segmentation layer of the classified feature.

The questions can also be conditions (e.g., comparators or thresholds) which can be modified during a training session of such tree. To modify the weights, one can use the backpropagation algorithm described below in Section 3.5.2.

Their main advantage is they are simple to understand and generally easy to interpret and visualize. Moreover, the supplied data require very little preparation including but not limited to normalization or dummy variables. On the other hand, decision trees can be easily over-designed in too complex structures and such trees won't generalize well, often overfitted on given dataset.

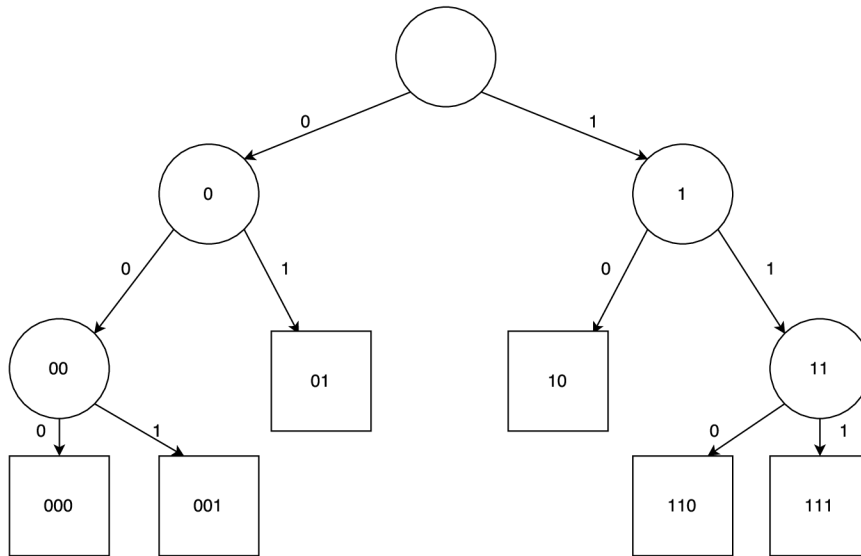


Figure 3.7: Example of a simple decision tree classifying sampled binary numbers. Each node sends the input to a corresponding child node on the left (0) or on the right (1). The leaf nodes are drawn as squares whereas the internal nodes as circles.

3.5.2 Backpropagation Network

Backpropagation networks are multi-layer feed-forward networks with supervised learning. There is no interconnection between neurons in the same layer but layers are fully connected to the next neighbouring layer in order to be able to do forward and backward propagation of values.

Forward-propagation

Each neuron in all layers but the input one disposes of a weight for each input initialized to a random value in the range $< 0, 1 >$, linear base function and sigmoidal activation function.

When forward propagating an input vector, the vector is laid out onto the input neurons. The vector is recalculated using following formulas for the next layer until the input vector is propagated to the output layer which outputs the response of the whole network itself.

The base function is shown in equation 3.1:

$$f(\vec{x}) = \sum_{i=0}^n w_i x_i \quad (3.1)$$

where \vec{x} is the input vector, w are weights for each input of a given neuron and n is the length of the neuron input vector and bias (term used as in [18, 31]) value resulting in $n = |x| + 1$.

The sigmoidal activation function is presented in equation 3.2 where λ is a constant set to $\lambda = 1$ and in further equations left out because of this fact.

$$g(u) = \frac{1}{1 + e^{-\lambda u}} \quad (3.2)$$

The output of a neuron is then given by using the equations 3.1 and 3.2:

$$y = g(f(\vec{x})) \quad (3.3)$$

Back-propagation

Back-propagation is used only when the network is trained and is one of the base methods for training feed-forward networks. The method is based on adjusting weights depending on the error calculated by equation 3.4 for an output neuron p using produced output (o) and desired output (d) values.

$$E_p = \frac{1}{2} \sum_{j=1}^m (d_{pj} - o_{pj})^2 \quad (3.4)$$

The change of weights is calculated using equation 3.5 where ∇E_p is the derived error gradient and μ the learning rate.

$$\Delta \vec{w}_p = -\mu \nabla E_p \quad (3.5)$$

To calculate one particular change of weight we use formula 3.6 where l is the given layer of the network, j is the j -th neuron in layer L and i is the i -th input of the neuron j .

$$\Delta^l w_{ji} = \mu^l \delta_j^l x_i \quad (3.6)$$

For the output layer l of the network we use formula 3.7.

$$\delta_j^L = (d_j - y_j) y_j (1 - y_j) \quad (3.7)$$

For hidden layers of the network, we use the same principle propagating the error backwards from the output layer as shown in equation 3.8 where $\lambda = 1$ and therefore left out.

$$\Delta^l w_{ji} = \mu^l \delta_j^l x_i = \mu \sum_{k=1}^{n_{l+1}} (\delta_k^{l+1} w_{kj})^l y_j (1 - y_j)^l x_i \quad (3.8)$$

Each repetition of forward and backward propagation of all inputs is called an epoch. Finally, we can choose when to update the weights, in batches after all input vectors were processed or using the stochastic method where weights are updated after processing each input vector.

Chapter 4

Design

The design of Examon Web is centred around the needs of **HPC** users and missing functionality of currently available tools such as Intel vTune. First, the target audience and users are defined in section 4.1. Next in Section 4.2, the interface and backend of Examon Web are laid out with basic wireframes provided in the Appendices. In the final Section 4.3, the job anomaly detection tool design is described.

4.1 Target Audience & Users

There are two main groups in the target audience for Examon Web. **HPC** system administrators and active users. Both groups differ in the level of details and domains of the available data. Where system administrators require mainly a global overview of the whole system with the availability to perform drill-down analysis down to the node-level info, users of **HPC** systems are oriented by the job-relative domain in order to precisely identify the resources used by their jobs.

System administrators are mainly seeking to overview the status of their monitored facilities. As stated in Chapter 2, system administrators can use other state-of-the-art tools such as Ganglia to monitor their system but these tools usually don't provide an easy to consume global system overview and therefore most of the facilities develop their own tools for this task. Examon Web can fill in this gap in the current tools while not trying to substitute such tools in order to encourage system administrators to use, e.g. Ganglia, and Examon Web together.

Using Examon Web as a basic monitoring tool can bring indirect benefits such as seeing the hotspots in the architecture of their mainframe thanks to a cluster visualization. Using the same view administrators can also see malfunctioning or powered-off nodes which then can be easily physically located in the cluster room.

HPC system users are the major target audience for Examon Web. By the combination of sensor and job data, users will be able to deeply analyze their jobs. They will be able to check the resource utilization in fine detail. Part of this analysis will be automated by the job anomaly detection tool which will provide quick insights on the performance of their jobs and the probable reason what caused the unwanted behaviour.

Using the same tool Examon Web can provide intelligent self-organizing dashboards. Such dashboards can organize the charts of sensor metrics in order of their importance in the displayed job. The order of metrics will be determined during implementation.

With the job data available, users will also be able to check the status of their jobs such as the queuing time, resource utilization or simply whether their jobs are finished yet. When it comes to rather large jobs that need a significant portion of a cluster, Examon Web can be used by users the same way the administrators do. To check the cluster status and the condition the cluster is in.

4.2 Examon Web

In this section, we will describe the Examon Web for the aggregation and visualization of collected and live data. In Figure 4.1, a high-level schematic of the Examon Web and its connection to the monitoring framework are laid out. The front-end is a web application available to users via a regular web browser as a **single-page application (SPA)** [33] built using the Angular framework. It consists of several views targeted to different users, displaying time-series charts, a visualization of a cluster and single-number metrics.

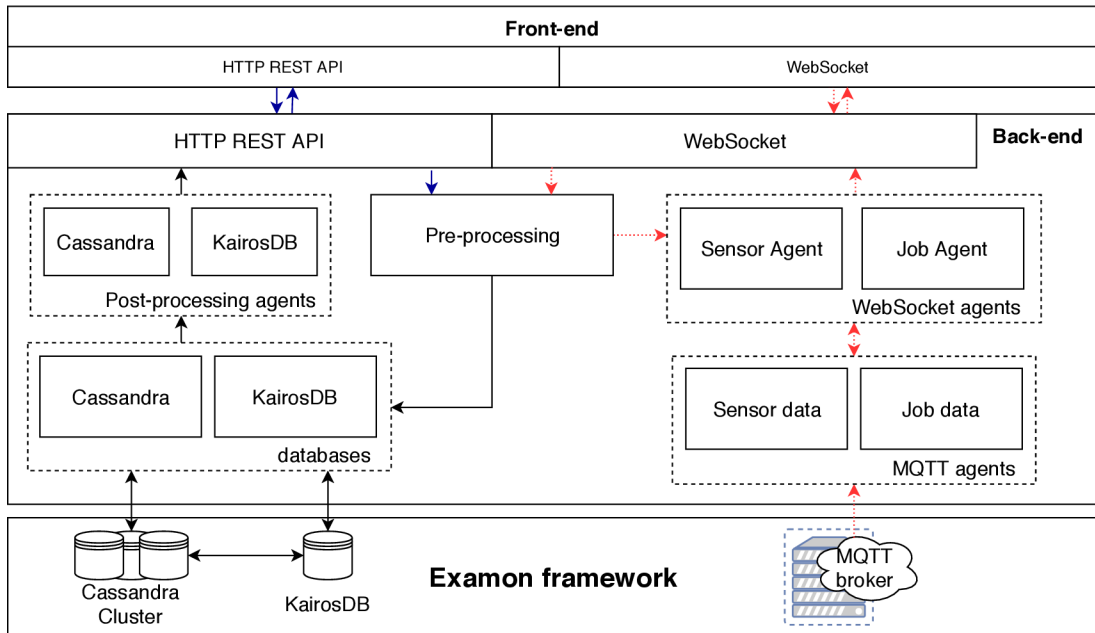


Figure 4.1: High-level schematics of Examon Web architecture.

The back-end of Examon Web resides on a server and serves as an interconnection between front-end and the data sources while ensuring a consistent data format that is fed to the front-end. The back-end exploits Examon framework, specifically by connecting to its data sources available via KairosDB, Cassandra cluster and **MQTT** broker.

4.2.1 Front-end

The front-end component is the client side of Examon Web which connects to the back-end, retrieves data and visualizes them to the user. It uses two different interfaces: the **HTTP REST API** [15] service and the **WebSocket** [14] interface.

The **REST** API implements a request-response communication mechanism and it is used to retrieve time-series-based data from the back-end. The goal is to give a concise

but informative insight on resource requested and used by each job (both running and completed ones).

The WebSocket interface implements a socket-like communication in order to feed live data straight to the application without polling (conversely to the **REST** API). The interface should be implemented with the help of Socket.IO library [44]. This enables the application to continuously fetch new data without the need of any request to the server.

The Examon Web front-end uses the Angular [53] framework as its base on top of which several other libraries are used. The crucial ones are Dygraphs [52] and Bootstrap [40]. The former produces powerful time-oriented charts utilizing the **canvas** HTML element. The latter is a CSS framework to generate uniform user interface across the whole application and various browser. The front-end consists of three major parts. The job information dashboard visualizing job-related data, cluster dashboard visualizing cluster-level aggregated data and the last part rendering a live visualization of a cluster utilizing the **MQTT** data stream.

Job Information Dashboard

The main task of the job information dashboard is to inform a user about their submitted jobs and their state. The job ID assigned by the job scheduler is used in the Examon Web to look up the desired job. The user interface offers to query by manual input or, for ease of access, a list of currently active jobs. The last successfully finished job is also displayed. An active job is a job that is currently in a queue, a finished job is a job removed from the queue and stored in the Cassandra cluster.

Once the job ID is submitted a request is sent to the back-end which will then assess whether the job is active, finished or non-existent and responds with appropriate data. In the case of a job being successfully found, the application performs additional queries to retrieve additional data to provide more details in form of time-series based charts.

The low-fidelity wireframe of job overview dashboard can be seen in Appendices **A.2**.

Cluster Dashboard

The cluster dashboard is very similar to the job dashboard in terms of used components and the form of data. The main difference between them is that the cluster dashboard is mainly designed for a panoramic view of the whole cluster.

The dashboard provides insights on the supercomputer in the form of time-series chart as seen in low-fidelity wireframe in Appendices **A.3**. Each time-series chart is accompanied by two single-number metrics providing the latest and average value of given metric.

Cluster Visualization

The cluster visualization dashboard can be split into two types. The first type displays the cluster via a 3D render model with which a user can interact. Users can go through the cluster and select any node to see node's details such as temperatures or loads. This type of visualization can be displayed if the facility can provide a 3D model of their supercomputer.

Otherwise, the second type of cluster visualization can be made using only generated 2D model displaying nodes as simple boxes with their name and current metric value.

In both cases, the nodes are colour-coded in terms of heatmaps to easily spot hotspots in the cluster. Also, both types utilize the **MQTT** data stream delivered via WebSocket interface to display desired metrics.

4.2.2 Back-end

The back-end resides at the server which serves requests from the Examon Web's front-end. It implements the same interfaces as front-end (HTTP **REST** API and the WebSocket) and on the other side it connects to various database sources and an **MQTT** broker.

REST API

REST API is used to retrieve the offline data stored in KairosDB and Cassandra cluster while maintaining consistent API with JSON-formatted responses. The JSON data format was chosen because it is native to the front-end and easy to convert in the back-end. After successfully receiving a request (from the front-end) the Flask [46] micro-framework executes a set of pre-processing routines to parse the request and to delegate the task to the correct internal function. These functions utilize Cassandra or KairosDB database driver in order to connect to the mentioned databases and retrieve data.

In the case of the Cassandra driver, a prepared **Cassandra Query Language (CQL)** query is used to fetch a specific set of data. This data is then post-processed in order to provide the front-end with consistent JSON-formatted data. In the case of KairosDB, a set of built-in functions enable to retrieve a subset of available data, which is aggregated by KairosDB itself. This subset contains only the selected metric in given time range with specified tags. These built-in functions (i.e. **sum** or **avg**) move post-processing phase, with respect to interaction with Cassandra, to the KairosDB itself.

WebSocket

The interface is used to retrieve live data from the **MQTT** stream. A subscription-based model was designed for immediate data retrieval. The model is applied to the two WebSocket agents (sensor and job agents). This interface allows handling the continuous real-time data stream generated by the cluster.

Once the front-end instance is connected to the WebSocket interface, back-end pre-processes the request and use one of the two available WebSocket agents. The two agents are intended for two different data sources coming from **MQTT** broker (job and sensor-related data) while having similar purposes. The need for two separate agents was the form of data received from the **MQTT** broker. Job-related data is received in JSON compared to sensor data which is received in `<timestamp>;<value>` format.

A front-end instance subscribes to a sensor metric or job ID. This creates a Socket.IO room inside the WebSocket agent with a given keyword (sensor metric or job ID) if such room does not exist, otherwise only increases the number of participants in the already present room. Front-end instance can subscribe to one keyword at a time for each WebSocket agent. Every time a value with a keyword, that is also an existing room, is received, the updated value is sent to all subscribed users in such room. This way the agents can manage which data should be sent via WebSocket to front-end instances or stored without any WebSocket interaction.

The **MQTT** agent for sensor data gathers data from **MQTT** topics used by sensors in the cluster and computes a weighted **weighted moving average (WMA)** of data using the equation 4.1:

$$v_{new} = v_{current} \times \alpha + v_{previous} \times (1 - \alpha) \quad (4.1)$$

where $v_{previous}$ value is set to the first available value during initialization and afterwards to the last v_{new} and $\alpha = 0.75$ as a default value was chosen based on a short-term evaluation. The agent exposes data using a Python dictionary where the key values are gathered metrics.

Job data agent subscribes to all **MQTT** topics related to job data published by PBS hooks. Each job must fulfil two conditions in order to be marked as a finished job. The first condition is the job must be finished before its timeout time (computed from the start timestamp and its required time). The second condition is a specific set of **MQTT** messages received in order: 1) **runjob** event, 2) **jobs_exec_start** event(s) and 3) **jobs_exec_end** event(s). In case of 2) and 3) the agent expects the same number of events as is the number of allocated cores. Once the front-end instance subscribes to a job, all messages with given job ID trigger a callback function which sends the updated record to all subscribed instances which then update their view.

4.3 Job Anomaly Detection

When a user runs a job on an **HPC** system, usually the only way to monitor the behaviour and state of their program is to manually inspect the output of it and eventually the execution time and exit code. With large and long jobs this way of monitoring is not a viable solution. Using neural networks it is possible to detect anomalies in programs' runtime observing side effects of such behaviour such as low or extremely high **IPS**, high cache misses or **CPU** power-saving states. The designed solution for job anomaly detection is presented in this section together with the data, its filtering and processing needed for anomaly detection.

4.3.1 Data

The necessary data are gathered via Examon framework and stored in KairosDB database and the Cassandra cluster. We can split the data into two categories, job and metric data.

The job data come from the job scheduler which reports various info about the job, mainly the allocated nodes, cores and other computational resources. We use this data for determining the job's execution time, its resource allocations and location of the job in a cluster (node names and core numbers). The job data is sent via **MQTT** and stored directly in a Cassandra cluster.

Using the job data, metric data can be queried. It is measured and gathered independently of the job data and monitored on per-core, per-**CPU** or per-node basis categorized into metrics. Each measured value is then sent via **MQTT** to KairosDB and stored according to its cluster location and metric. There are over 30 metrics monitored including but not limited to core load, C6 and C3 **CPU** state shares, system, **CPU**, IO and memory utilization or various temperatures gathered from various places inside a node or **CPU**.

For detecting job anomalies, twelve major metrics were chosen for the best reflection of the job performance. A short summary of the chosen metrics is shown in table 4.1.

Table 4.1: Overview of measured metrics with crucial information about the resolution and their units.

Metric name	Metric tag	unit	sampling rate	base
core's load	load_core	%	2s	per-core
C6 states	C6res	%	2s	per-core
C3 states	C3res	%	2s	per-core
instructions per second	ips	IPS	2s	per-core
system utilization	Sys_Utilization	%	20s	per-node
CPU utilization	CPU_Utilization	%	20s	per-node
IO utilization	IO_Utilization	%	20s	per-node
memory utilization	Memory_Utilization	%	20s	per-node
L1 and L2 bounds	L1L2_Bound	%	2s	per-core
L3 bounds	L3_Bound	%	2s	per-core
front-end bounds	front_end_bound	%	2s	per-core
back-end bounds	back_end_bound	%	2s	per-core

KairosDB provides us with a **REST** API for querying metric data in various ways. The queries are formed using JSON objects and results are also returned as JSON objects. The KairosDB limits all stored data to 21-day window.

For complete data acquisition, we combined the job data and metric data together and queried only jobs which fit several conditions described in [4.3.2](#).

4.3.2 Data Acquisition

First, job data need to be queried and filtered according to several rules:

- job runtime must be between 10 and 60 minutes
- job must occupy the whole node (multiplies of 16 cores)
- job must be run within the 21-day period

All data points are aggregated by 30 seconds on cluster level using averaging aggregator available in KairosDB. Cluster level can be achieved by averaging every used core and node in the job to one time-series per metric.

The rule of minimum 10 minutes is because shorter jobs are usually a development, not the production version of a program and in current **HPC** facilities, such program is very cheap to run and therefore no deep performance analysis is needed. The same goes for jobs smaller than one node (16 cores in case of Galileo supercomputer).

Jobs longer than one hour are not suited for training the network because of too large input vectors and the loss of information in further data processing.

4.3.3 Data Labelling

In order to correctly label all chosen jobs and their metrics, a simple graphical user interface must be created. The GUI is based on Examon Web. Visualization is done in time series fashion using a charting library. This helps to better understand the in time correlations between all metrics combined.

4.3.4 Data Processing

After labelling the job, all metric data with labels are generated and can be worked on further. All metric vectors must be interpolated to a fixed set of values in order to be applied to the input neurons of the metric networks.

All values should also be normalized to values between $\langle 0.0, 1.0 \rangle$ for more precise training. If the values are kept as they were recorded, there is a high chance of incorrect labelling of such metric.

4.3.5 Metric Networks

To achieve best results/speed ratio a backpropagation neural network is created for each metric. This gives the total of twelve networks completely independent of each other meaning the training process can be fully parallelized.

The number of input neurons must be determined depending on the dataset and the results during the implementation phase. With small input vectors the networks might over-generalize and with too wide input layer the network will have too many connections which can result in poor performance and generally the networks can easily overfit.

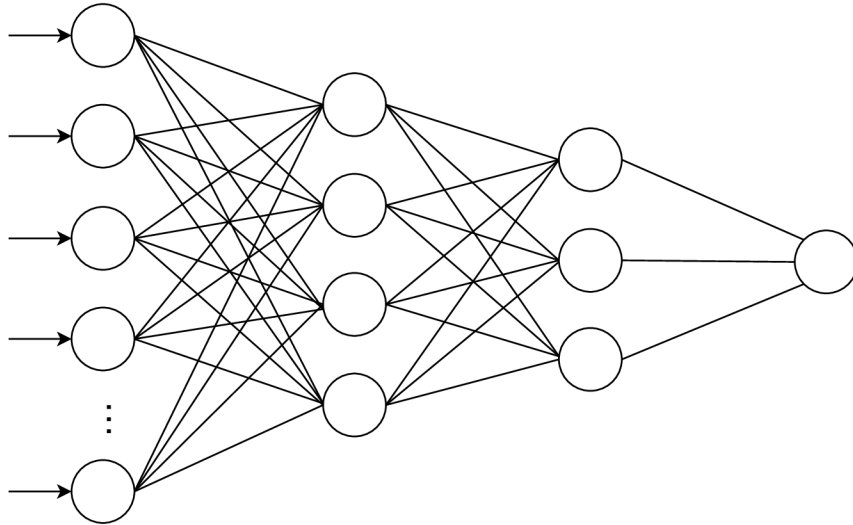


Figure 4.2: Visualization of metric's backpropagation network. On the left side there is a set of input neurons and on the right one output neuron.

All metric networks should be configured the same way to achieve uniform results and for easier detection of overfitting or over-generalization. The suggested configuration should be 60 - 180 input neurons, with 2 or 3 hidden layers depending on the input vector size. The output layer will be only one neuron since the problem distinguishes only two states

between which we can determine the level of certainty. In Figure 4.2 the proposed metric network is shown to better understand the architecture.

4.3.6 Job Network

Once all metric networks label their input data, the output will be the job network's input which results to a vector of 12 values each signifying a given metric.

The job classification network is created with 12 input neurons, one hidden layer of 4 neurons and one output neuron as this is an expected structure which should work well.

The job network should be trained on a labelled dataset as will be the metric networks but when evaluating the job classification network we can choose between the real outputs of the metric networks or the expected outputs from the classified dataset. Both evaluations should be very similar.

Chapter 5

Implementation

This chapter describes the implementation of the proposed Examon Web system with several extensions needed during the implementation. In the first section 5.1 the graphical user interface is presented with three versions of it each used for a different purpose. In the second section 5.2 the back-end of Examon Web is presented. Next section 5.3 describes the process and results of creating an annotated dataset suitable for several back-propagation neural networks which are presented in section 5.4. The job anomaly classification network utilizing decision trees is proposed in section 5.5 and all is summed up in the section 5.6.

The high-level schematic overview of implemented modules and other parts of Examon Web can be seen in Figure 5.1 to show how it is organized and what modules share common parts. Each displayed part is described in the following text.

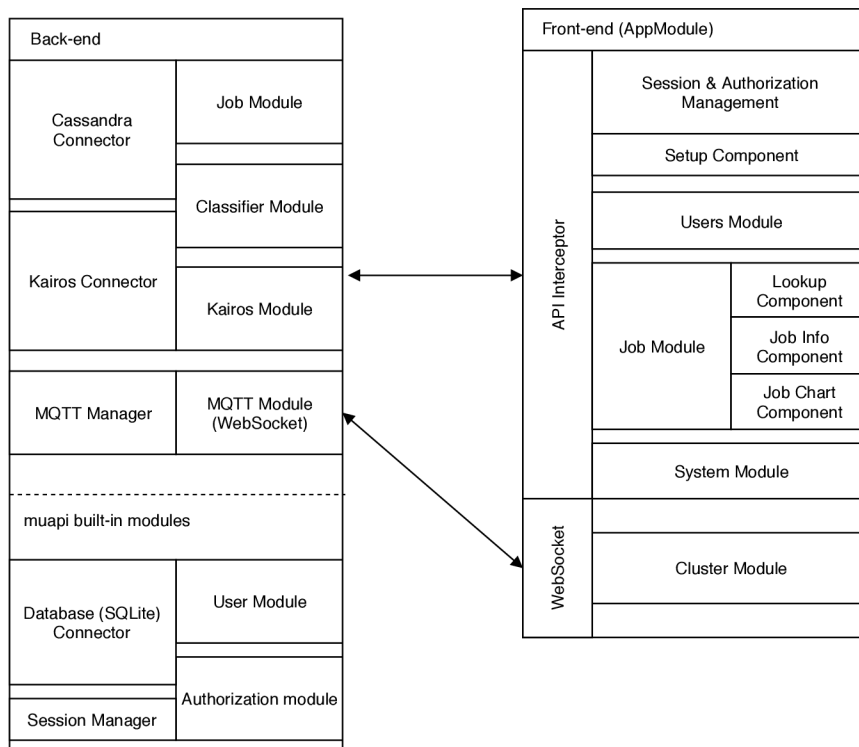


Figure 5.1: High-level schematic overview of Examon Web's modules and other major parts on both back-end and front-end sides.

5.1 Graphical User Interface

In this section, we present the implemented **GUI** architecture and major building blocks (modules in Angular terminology) used during the development. Afterwards, three different versions of Examon Web front-end are described as well their purpose.

The graphical user interface as designed in section 4.2 is built using TypeScript language and the Angular framework¹ and its scaffolding tool Angular CLI which provides a quick and easy way to bootstrap new components, services and modules.

Angular framework enforces a specific architecture of the application. The main building blocks are modules, components, services, templates and directives which are briefly explained.

Modules declare a compilation context of a given set of components which are dedicated to a specific workflow, set of capabilities or an application domain. Modules can associate their components with services and by this form a functional unit. Each Angular application has at least a root module which provides a bootstrap mechanism for the application. Each module also defines the routing between its components. Examon Web utilizes modules heavily and each clearly separable functionality is defined as a module.

Components are defined using a `class` that contains the data and logic for their template. Components handle the data manipulation and store component-specific data entered by a user.

Templates are a combination of HTML and Angular markup that modifies the contents of the HTML document before and after it is displayed.

Services handle data and logic which is not directly linked to a specific view and are shared across multiple components or even modules. Services are injected into components using dependency injection mechanism.

The core of Examon Web resides in the `AppModule` module which, using various components, handles users sessions, Examon Web module's discovery and other globally needed functionality such as navigation bar. Handling of user access is done via `LoginComponent` and `LogoutComponent` with the help of `AuthService` service which manages sessions in the local storage.

`ApiInterceptor` is an extension to regular `HttpClient` which adds the `Authorization` header with a session token (if available) to each HTTP request sent to back-end and handles HTTP response codes appropriately.

All modules except the Cluster module utilize HTTP **REST** API made available by the back-end and this API is described in detail in section 5.2.1

5.1.1 Homepage

The homepage is the initial page which user sees upon successfully logging in. The list of available modules obtained from the root router is displayed to which a user can navigate next. The homepage can be seen in Figure 5.2.

5.1.2 User Management

User management is handled by the `UsersModule` which can add, edit and remove users. Users dispose of certain role specified by a role number where 0 indicates an *administrator* account with no restrictions apart from deleting themselves. The second role, defined as a

¹Specifically Angular version 4.3.

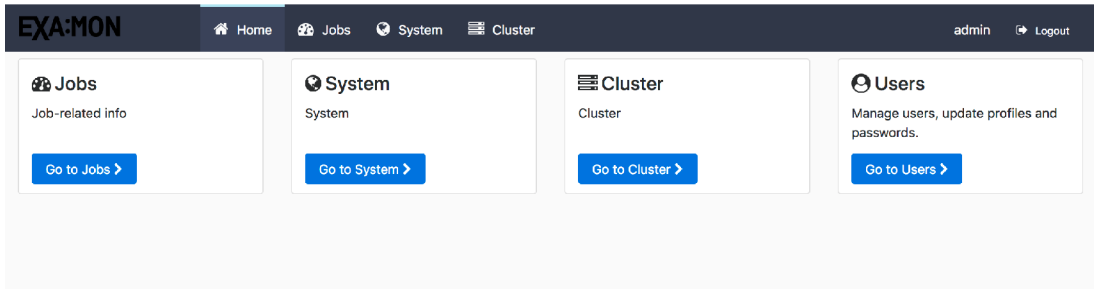


Figure 5.2: Homepage with all available modules.

user account can access all features but with limited privileges. *Guest* users are capable of view-only operations while not being able to access sensitive content.

These roles can be redefined and extended up to 255 different roles thanks to the design of both front-end and back-end.

The back-end and front-end can be set to a special state called *Setup Mode* where the `SetupComponent` is used when the back-end responds with HTTP 442 status code. This status code is not defined in the HTTP standard and is application specific. The HTTP code states that the API has not been set up yet and needs an initial administrator account. The front-end handles this state by redirecting the user to the setup page where they can set the username and password for the initial administrator account.

The initial listing of users can be seen in Figure 5.3. The overview provides the username, email, name, surname and the role of a user and two actions buttons to view/edit the specified user and to delete the user. The delete button is not available for the main administrator account or the currently active user and only administrators can delete an account.

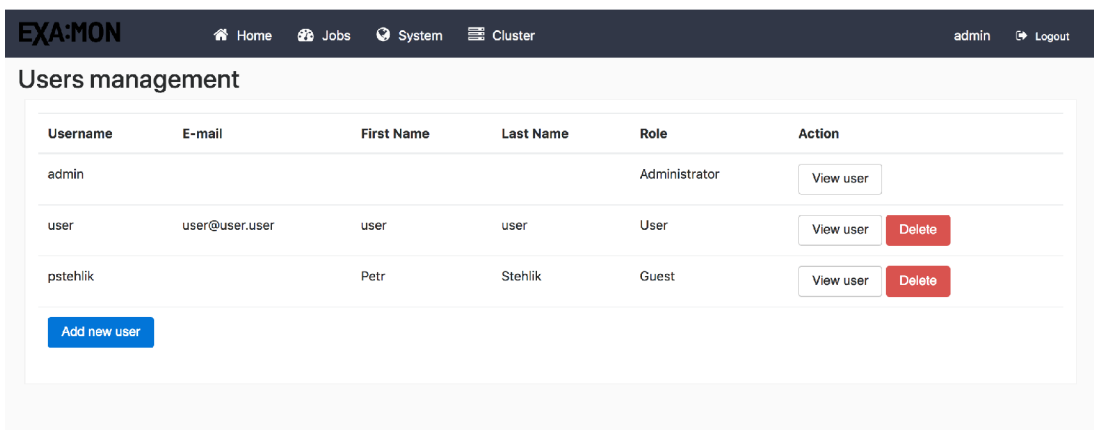


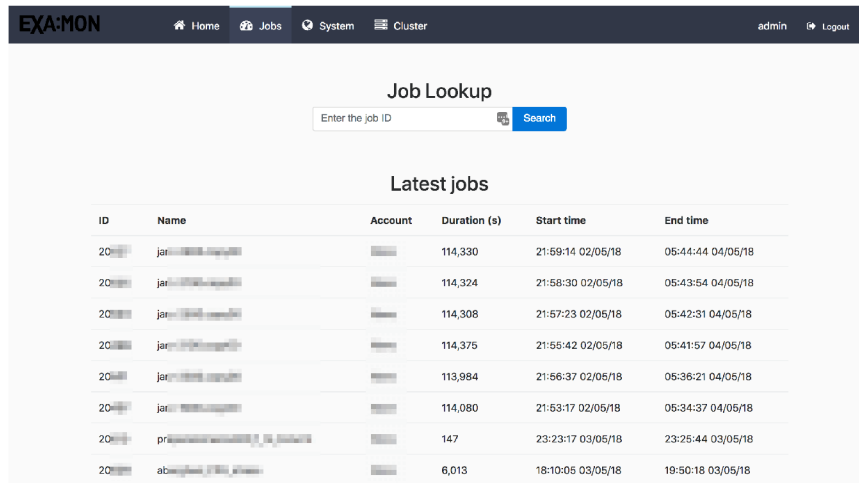
Figure 5.3: The user listing showing all three types of users.

5.1.3 Job Module

The next major module is focused on the jobs submitted to the supercomputer by users. All jobs are collected by the Examon framework and stored in the Cassandra database.

On the initial job page, the user with lower than administrator privileges only sees jobs submitted with their username (to protect the privacy of other users since the job

managers collect detailed information up to environment variables which can be used to hold confidential data such as credentials to various services).



ID	Name	Account	Duration (s)	Start time	End time
20	jar-...	...	114,330	21:59:14 02/05/18	05:44:44 04/05/18
20	jar-...	...	114,324	21:58:30 02/05/18	05:43:54 04/05/18
20	jar-...	...	114,308	21:57:23 02/05/18	05:42:31 04/05/18
20	jar-...	...	114,375	21:55:42 02/05/18	05:41:57 04/05/18
20	jar-...	...	113,984	21:56:37 02/05/18	05:36:21 04/05/18
20	jar-...	...	114,080	21:53:17 02/05/18	05:34:37 04/05/18
20	pr-...	...	147	23:23:17 03/05/18	23:25:44 03/05/18
20	ab-...	...	6,013	18:10:05 03/05/18	19:50:18 03/05/18

Figure 5.4: Latest job listing showing all recently submitted and finished jobs on the cluster with the option to click through each one of them or look up a job by its ID.

The listing as seen in Figure 5.4 displays the job ID, name, account, duration in seconds, start and end time of the job. These are only the basic data to help determine the desired job. By clicking on a row of a specified job the user is directed to the job detail dashboard.

The user can also look up a job by its ID which will redirect them to the job detail dashboard.

The job detail dashboard offers 3 views. The general overview of the job displaying various info about the job, a dashboard displaying performance-oriented metrics in time-series charts and a dashboard focused on energy consumption displaying time-series charts as well.

The *general overview* informs the user about crucial information of the job such as the number of required nodes, cores, memory, GPU together with the job ID and name, account, user and several others as seen in Figure 5.5.

The dashboard also provides the user with information about the queue, start and end times of the job, often useful to see how long the job took to execute. The state of the job and aggregated metrics are displayed as well telling the user the totals of CPU and GPU power, average temperature and utilization of the hardware used during the runtime of the job. The last piece of information on this dashboard is a chart showing the average core load during the execution.

The *performance dashboard* can be seen in Figure 5.6. It collects and displays several pre-selected metrics crucial to determine the overall performance of the job and detect possible problems in forms of time-series charts. The displayed metrics are:

- node utilization
- instructions per seconds
- CPU frequency
- system, memory, and IO utilization

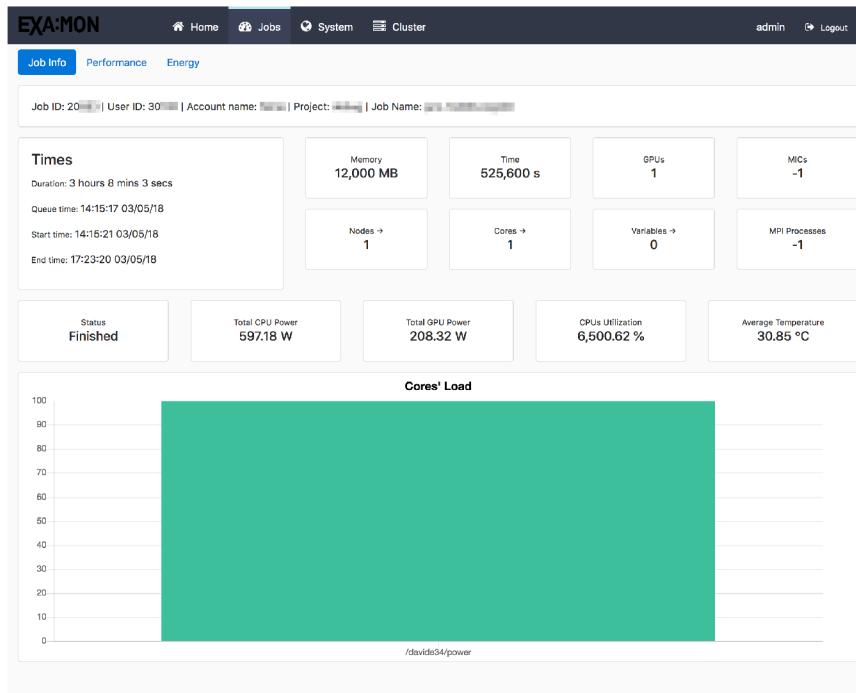


Figure 5.5: The general overview with information about a specific job.



Figure 5.6: Performance dashboard presenting the user performance related metrics in time series charts.

- front and back-end bound instructions shares
- **CPU** power saving states (C3 and C6)

The *energy dashboard* is very similar to the performance one only displaying different metrics:

- power consumption
- **CPU** temperature
- **CPU**, DRAM and other components' power

5.1.4 System Module

The system module is made of only the *system dashboard* which displays the cluster's load, temperature and power consumption during the last 30 minutes. Together with the time series charts, there are also single number metrics showing the average and latest values of each metric as seen in Figure 5.7.

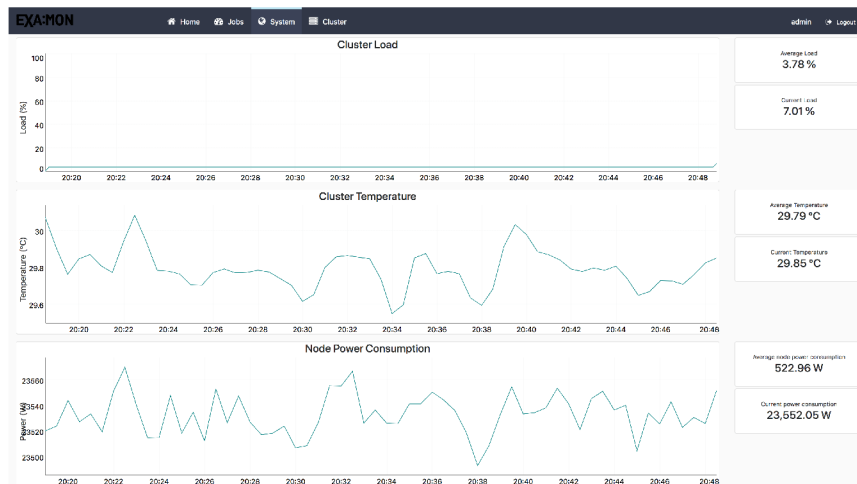


Figure 5.7: The system dashboard offers the overview of the whole cluster in easy to consume charts.

5.1.5 Cluster Module

The cluster module offers a single view on the monitored cluster. Depending on the setup, the cluster view can be a rendered interactive 3D model as shown in Figure 5.8 made using Blend4Web library and a Blender model which can be interacted with. The other option is an HTML generated 2D view of the cluster (seen in Figure 5.9). Both options use a colour-coded scale to easily determine the hotspots and possible problems in the cluster only by quickly taking a look at the model. The colour range starts at deep blue indicating the minimum of received values ending at red signifying the maximum value recorded.

The user can choose from different metrics to be displayed including but not limited to:

- ambient temperature
- **CPU** load, power, temperature

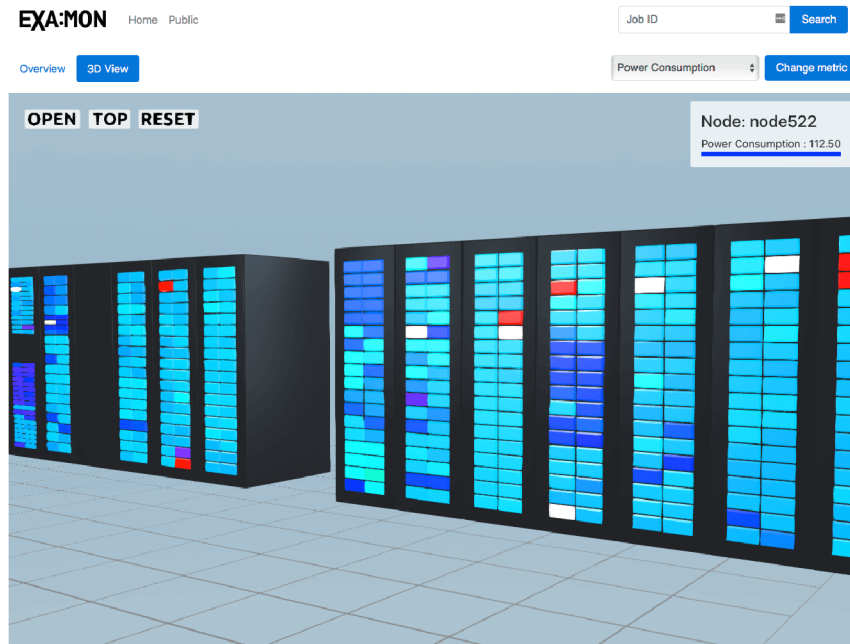


Figure 5.8: The interactive 3D model of the Galileo supercomputer fed by the live metrics data.



Figure 5.9: The 2D HTML generated model of D.A.V.I.D.E. supercomputer supplied with data via WebSockets.

- system, memory or IO utilization

The data to the model is delivered via the WebSocket interface with a simple subscription-based model which is elaborated in section 5.2.2. The front-end uses the Socket.IO library to implement WebSocket communication which also enables us to utilize the room functionality as described later.

5.1.6 Front-end Adaptations

During the development, several alternative versions of the Examon Web’s front-end were produced. The first fully functional front-end was specifically designed for the Galileo supercomputer² located in CINECA, Bologna, Italy. This version included the interactive 3D model of the cluster itself and was the cornerstone for further development both front-end and back-end. It lacked the subscription mechanism for receiving live data from the MQTT stream and because of that performance issues occurred during the deployment process where the back-end was not able to handle the incoming MQTT data. This version utilized WebSockets for live job monitoring where the user could browse through jobs currently running on the cluster and monitor them in real time.

The second version was tailored for the newly installed D.A.V.I.D.E. supercomputer³ in the same facility. This supercomputer is built using the POWER8 processors and therefore the formerly monitored metrics had to be adjusted due to the different CPU architecture. Moreover, the 3D model was replaced with an HTML generated overview since there was no 3D model available for the cluster. D.A.V.I.D.E. also replaced the PBSPro job manager with SLURM which does not publish job information before or during the submission of a job but only after the job has been finished. This led to the removal of the live job monitoring on the Examon Web front-end.

The third and last version dropped almost all functionality in favour to maximize the effectivity of the front-end. The single purpose was to annotate a dataset made of various metrics collected during the operation of the Galileo supercomputer. The front-end listed only jobs available in the dataset and reduced the job dashboard to a single view with the metrics selected for job behaviour analysis. The only additional part was the annotation component which was used to mark a metric or the whole job as suspicious and afterwards this annotation was stored in a database. This version was crucial for creating a representative dataset designed for machine learning described in next sections.

5.2 Back-end

The back-end is created in the Python language with the help of several libraries. The major framework used in the back-end is Flask [46]. It is a micro-framework used for building various web applications. In relation to this thesis, Flask is used as a web server which makes available the HTTP REST API together with the WebSocket interface which is handled by a Flask-SocketIO [20] extension.

Another Python package used for developing back-end is an author-made package called muapi [50]. The name stands for “modular User-oriented REST API”. This library provides a Flask-based application with user and session management, simple MongoDB or SQLite database connector, automatic REST API module discovery and a configurator interface.

²<http://www.hpc.cineca.it/hardware/galileo>

³<http://www.hpc.cineca.it/content/davide>

The following endpoints are made available by the muapi package:

- / [GET] – list all routes and HTTP methods available
- /authorization [GET, POST, DELETE]
 - GET – check the validity of a user session located in the `Authorization` HTTP header field
 - POST – obtain a session by providing correct user credentials
 - DELETE – invalidate a session
- /users [GET, POST]
 - GET – get a list of all users (requires at least the *user* role)
 - POST – add a new user (requires the *administrator* role)
- /users/<user ID> [GET, POST, DELETE] – all methods require at least *user* role
 - GET – get a user specified by their ID
 - POST – update a specified user
 - DELETE – delete a specified user (requires the *user* role or *administrator* role in case of deleting a user other than themselves)

Modules are a crucial part of the muapi functionality. One can create a module by instantiating the `Module` class available in the package and placing it in a directory specified by a configuration file loaded by the application. Afterwards, the module is registered, imported and made available by the server. Further information about modules, their creation and usage can be found in muapi’s wiki documentation⁴.

In the following text, the endpoints specific to this thesis are described. First, the **REST** endpoints are laid out and afterwards the WebSocket communication together with how the **MQTT** data stream is handled via the subscription-based model.

5.2.1 REST API

The API is a set of muapi modules separated by functionality very similar to the front-end design. Each module has a specific set of use-cases and utilizes various approaches to the given problem. The first module described is the *job* module handling job lookup and retrieval, afterwards the *Kairos* module fetching data from the KairosDB data source, then the **MQTT** module together with the WebSockets subscription-based model is presented and finally the *classifier* module is briefly described as the core of classifier is in the training phase described in Section 5.4.

Job Module

Job module handles all requests regarding the job information retrieval. The following endpoints are available via the API:

- /jobs/latest [GET] – fetch the latest jobs from database

⁴<https://github.com/petrstehlik/muapi/wiki/Creating-a-Module>

- `/jobs/<job_id>` [GET] – fetch job information specified by its ID

All endpoints utilize the Cassandra database connector where all job-related information is stored. The connector is configured using the muapi configuration interface. The data source contains two tables made of the same data but with different sets of keys. The first table is indexed only by the job ID used in the `/jobs/<job_id>` endpoint when querying specific job only with the knowledge of its ID.

The second table uses a compound key made of the user ID, start time and job ID. This tuple needs to be partly specified when querying this table with the `ALLOW FILTERING` property. The `/jobs/latest` endpoint differentiates between different users. In case of the administrator role, the database is queried without the user ID specified. Otherwise, the user ID is obtained from the PAM interface [35] and only jobs with the specified user ID are returned. This ensures no information leakage is probable because the user ID is obtained via an independent API.

The latest 100 jobs are always fetched unless there are fewer jobs present in the database. The jobs are then ordered by the start time and returned as a response.

The need for a unified model representing a job arose during development because of different fields present in case of the PBSPro and SLURM job managers. These differences are unified in the `Job` model which returns all information in the unified format.

Kairos Module

The metrics data are obtained via the Kairos module which utilizes the forked `pyKairosDB` [3] package extending the original `pyKairosDB` [36] package with HTTP Basic auth and other features. This package encapsulates requests to the KairosDB `REST` API handling the timestamp manipulation and raw HTTP requests. Moreover, an aggregation Python module was developed to update the KairosDB query with aggregation parameters. These parameters are used during querying various metrics with the different sampling rate and unifies the rate throughout requests.

The module offers the following endpoints, all of which with the only HTTP GET method available:

- `/kairos/health` – get health status of KairosDB cluster
- `/kairos/status` – get status of KairosDB cluster which returns the deadlock state and datastore availability
- `/kairos/metrics` – list all metric names in the database
- `/kairos/tags` – list all tag names in the database
- `/kairos/tagvalues` – list all tag values in the database
- `/kairos/core` – fetch metric data with core level aggregation (lowest level meaning no vertical aggregation is done)
- `/kairos/cpu` – fetch metric data on cpu level (aggregate cores by `CPU` sockets)
- `/kairos/node` – fetch metric data on node level (aggregate metric by node tag)
- `/kairos/cluster` – fetch metric data with cluster level aggregation by cluster tag (full vertical aggregation)

The endpoints fetching metric data share common GET parameters required for successfully querying the database. The parameters `from` and `to` specify the querying time window in UNIX milliseconds timestamps. `metric` parameter sets the metric name for which to query, this parameter can be set multiple times in the request which will result in multiple queries made to the database each with a different metric name specified. Using the `node` or `core` parameter the query will be limited to a given set of nodes or cores. The `cluster` parameter is set in the back-end configuration since the Examon Web is always deployed for a specific cluster.

The optional parameter `aggregate` can be set in order to reduce the size of returned data and to omit probable gaps due to missing data in the database. The sampling rate is set in seconds with aligned start time and values in the time frames are averaged.

After successfully querying the data from KairosDB REST API, the data can be returned in raw format by setting the `raw` parameter in the request. Otherwise, the obtained data is processed in order to be easily parsed by the front-end's charting library and rendered. The post-processing is done on the back-end because of large datasets which can extremely slow down the front-end rendering.

An example query with all available features can look like this:

```
/kairos/node?
  node=davide10&node=davide11&
  from=1525363805000&to=1525369818000&
  metric=PCIE\_Proc1\_Power&metric=PCIE\_Proc0\_Pwr&
  aggregate=10
```

The metric data do not include confidential data and therefore no authentication is required to make requests for this module.

MQTT Module

MQTT module makes available two REST endpoints which utilize the MQTTManager described in section 5.2.2 together with the subscription mechanism:

- `/metric/<metric>` [GET] – get metric data
- `/metric/nodes` [GET] – get list of nodes of collected data

Both endpoints were used mainly during the development and currently are not used in any part of the front-end.

All other endpoints in this module are WebSocket ones, specifically:

- `subscribe-metric` – subscribe to a metric
- `unsubscribe-metric` – unsubscribe from a metric

Both endpoints use the `/render` namespace to distinguish from other WebSocket endpoints in the back-end.

Classifier Module

The pinnacle of this thesis is the classifier module used for classifying the jobs based on their effectivity of the execution. Using a predefined set of metrics and a trained backpropagation neural networks determine the likeliness of suspicious behaviour of the job's execution.

The module makes available two endpoints, one of which is used for annotating the dataset and the other for obtaining the result.

- `/classifier/<job_id>` [GET] – get the likeliness of suspicious behavior
- `/classifier/<job_id>` [POST] – set annotations for metrics and whole job

The GET method utilizes the KairosDB database where cluster-level metric data is fetched and evaluated without aggregation. Each metric is split into windows containing 80 values due to the design of the networks (see section 5.4). Afterwards, the calculated annotations are gathered and the average, minimum and maximum values of the whole dataset for the metric. These calculated metric values are then passed to the final network which annotates the whole job. The results are sent back to the front-end to be interpreted by the user.

The POST method expects a JSON object with metric names and the `jobber` with values 0 for non-suspicious behavior or 1 for suspicious behavior. These values are then stored in an SQLite database with the job ID as the primary key. This data was used for dataset creation. The process of annotation is described in section 5.3.

5.2.2 MQTT and WebSocket Communication

The data gathered by Examon framework is distributed via MQTT protocol to the broker where it is processed and stored in a database cluster. The MQTT broker is publicly accessible, and therefore, Examon Web can utilize this data in order to display metric data with a minimum delay.

All published MQTT topics use the key value scheme for topic names meaning a key is followed by its value. This way a precise structure of the metric can be reconstructed upon receiving an MQTT message.

The data can be split into two categories: 1) job data which comes from the job manager and 2) metric data comes from various publishers deployed on each node of a supercomputer.

Job data can be sent within one or three MQTT topics depending on the job manager. PBSPro MQTT publisher uses three topics to signal the state of a job:

- `jobs_runjob` – when a job is submitted to the job manager’s queue
- `jobs_exc_begin` – execution of the given job starts
- `jobs_exc_end` – a job finishes and is cleared from the manager

Each MQTT message payload includes the job ID and several other information about the job such as requested resources.

The SLURM job manager publishes only a single MQTT message after the job is finished and cleared off the queue with the topic ending with `jobs_info` value. The payload is a sum of all information similar to the set published by the PBSPro manager. Both job managers publish the messages in a topic in the following format:

```
org/<organization>/cluster/<cluster name>/<job's topic>
```

For the job data, a `JobManager` class was developed used to gather job data from the MQTT broker mainly because of multiple messages coming from PBSPro manager. This

data is kept in the internal structure and is handled once all the required messages are received. The behaviour of `JobManager` can be modified using callback methods built in the class.

Metric data is published with key-value topics as well. The format is more complex than job-related data in order to maintain minimal payload size of the messages because of large volumes of these messages published by each publisher. The `MQTT` topic is in following format⁵:

```
org/<organization name>/
cluster/<cluster name>/
node/<node name>/
plugin/<plugin name>/
chnl/data/<item>/<item number>/<metric name>
```

During the deployment of the Examon framework on D.A.V.I.D.E. supercomputer the topic was extended with several other keys:

```
org/<organization name>/
cluster/<cluster name>/
node/<node name>/
plugin/<plugin name>/
chnl/data/<item>/<item number>/
cmp/<compartment>/
id/<item id>/
unt/<unit>/<metric>
```

The message payload is always in format `value; UNIX timestamp`.

Metric data is used for the cluster visualization feeding the 3D or 2D model with live data. Further development might include utilizing this data for live data in job-related charts.

The metric data manager in the first deployment was subscribed to all metric-related topics but the `MQTT` data stream was extremely large and unable to be handled by a single server with low hardware resources (2 Intel Xeon E3 cores and 4 GB RAM) and the `MQTT` message handling overloaded the server all the time. Therefore, the further development was needed and the subscription-based mechanism was devised, see Figure 5.10 for the schema of this mechanism.

Once the user visits the cluster dashboard of Examon Web, a subscription message is sent via the front-end's WebSocket connection to the back-end. The back-end keeps a list of subscribed metrics and the number of users subscribed.

If the announced metric cannot be found in the subscribed ones but is present in available metrics a new `MQTT` topic subscription is made using a predefined topic to the metric and new Socket.IO room is created with the same name as the metric. Otherwise, the number of subscribers is increased and the WebSocket connection joins the metric room.

Back-end also runs the `MQTTManager` similar to the `JobManager` presented earlier with the additional methods supporting subscribing and unsubscribing to `MQTT` topics solving the performance issues of the previous version.

Once the back-end is subscribed to the selected `MQTT` broker's topic, initial data is sent to the front-end. It is a bulk of collected metric data gathered during the initialization

⁵Line breaks are inserted for better readability.

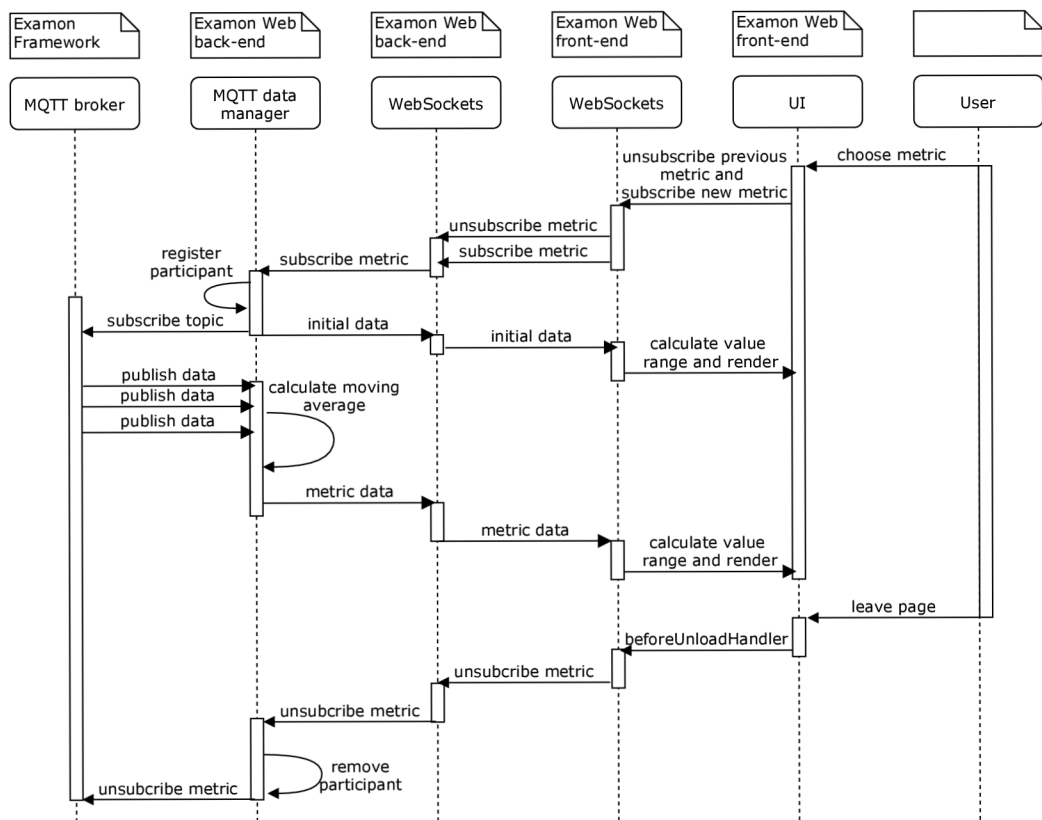


Figure 5.10: Flow model of WebSocket-MQTT subscription mechanism.

so front-end can render at least partial data for the user right after subscription to the topic. When the manager receives data, the topic and payload are parsed and the received value is averaged using a weighted moving average as presented in section 4.2.2. During the reception of the metric data, the minimum and maximum values are stored as well, and used by the front-end to calculate the colour range for the model.

The user can change the displayed metric or leave the page. On these actions the unsubscribe message is sent via WebSocket where back-end decreases the number of participants in a given metric room and if the number has reached zero the whole room is deleted and the `MQTTManager` unsubscribes from the `MQTT` topic at the broker. The averaged metric data is kept in memory for future subscribers in order to send the initial data to the front-end.

5.3 Dataset Creation

The backpropagation networks require an annotated dataset for their training. Currently, no suitable dataset was found to fit the needs of this specific case of analyzing time series data based on the fluctuation and absolute levels.

Two datasets similar in purpose were produced. For both datasets, the conditions the jobs must fulfil are that they occupied the whole node (multiples of 16 cores) and the execution must take at least 10 minutes. The maximum of the execution time was extended from 60 minutes to 24 hours as it is limited by the PBSPro job manager to 24 hours. This time extension allowed us to collect more data.

The distribution of job runtimes can be seen in Figure 5.11 for the total of 22 791 jobs submitted to the job manager during the time period from 2/11/2017 to 20/11/2017. The histogram clearly shows that the majority of jobs is shorter than 10 minutes most of which is shorter than 60 seconds. These jobs are relatively cheap to run a debug and are not representative enough for the dataset. For both datasets, the jobs were run on the Galileo supercomputer where the Examon framework monitored most of the cluster.

Both datasets were labelled manually by examining each time series metric individually and as a set as well. Next follows a set of examples representing the spotted suspicious behaviour and an example of a good job run.

In essence, if the job was balanced and ran well the `load_core` metric was set close to 100 % during the whole run and the `C6` metric close to 0 %. An example of this behaviour can be seen in Figure A.6.

The `back_end_bound` metric could vary during the runtime but if the metric was unbalanced it meant a non-uniform cache access which was labelled as suspicious. Other metrics were dependent on each other and it was easily spotted if the job was suspicious or not.

In many cases, the `load_core` metric was set around 50 % which meant half of the used cores were fully utilized while the other half was not used at all. This was also marked as suspicious. An example of such a job can be seen in Figure A.7.

Other suspicious behaviour was when a job had no `load_core` at all and the `CPU` was in `C6` state during the whole runtime as seen in Figure A.4.

Less often, jobs with sudden drops in utilization were spotted. These jobs are suspicious as well because of a possible indication of a problem in regards to unpredictable data loading or other similar issues. This behaviour can be seen in Figure A.5.

Other examples of suspicious behaviour include a long startup period with 0 % `load_core` at the beginning or the other way a premature end or long result storing period with 0 % `load_core` at the end.

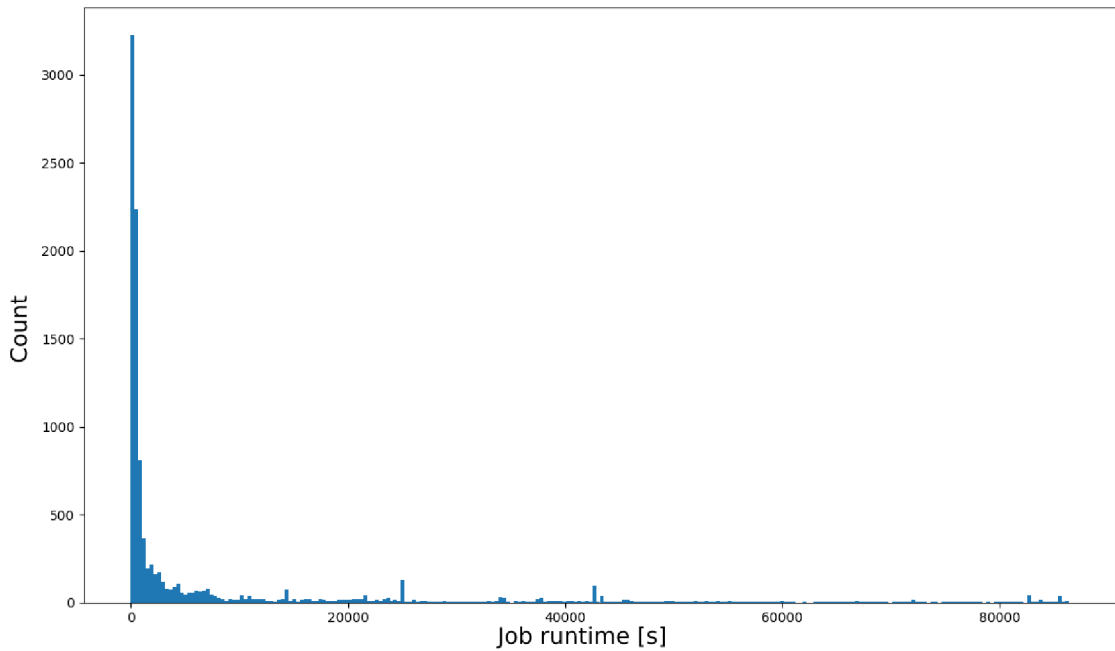


Figure 5.11: Histogram chart of job runtimes. Jobs with runtime less than 60 seconds are not plotted with the volume of 12 120 jobs.

First Dataset

The first, initial, dataset was made of 32 jobs with the additional condition of runtime between 10 and 60 minutes. The jobs were collected between dates 31/10/2017 and 20/11/2017. This dataset was representative enough for proof of work stage of developing the neural networks with fast training sessions and fast manual labelling.

A rough dataset consisted of 442 jobs but during the data acquisition stage, only the selected 32 jobs had all needed metric data present in the KairosDB cluster. Again the runtimes are mostly placed within the 10 to 20 minutes runtimes as seen in Figure 5.12.

In the end, the total of 19 jobs was labelled as suspicious and the remaining 13 jobs as non-suspicious. Some of their metrics were labelled as suspicious but the job as a whole not. Eventually, this helped to remove a certain amount of false positive labels from the jobber network.

Second Dataset

The second, considerably larger dataset consisted of 3373 candidate jobs to be evaluated. After the initial fetching stage about a half (1532) of the jobs had the required metric data. The data is first fetched on a per-node aggregation level but the resulting dataset was too large (around 40 GB) to be analyzed during the training period. The second round of retrieval was done using the original cluster-level aggregation which resulted in a dataset considerably smaller (around 160 MB). This dataset contains 1172 jobs. The number of jobs is smaller due to errors during the fetching of data since the data source was mounted on a low resource server which, during large queries, often failed to retrieve the data and crashed.

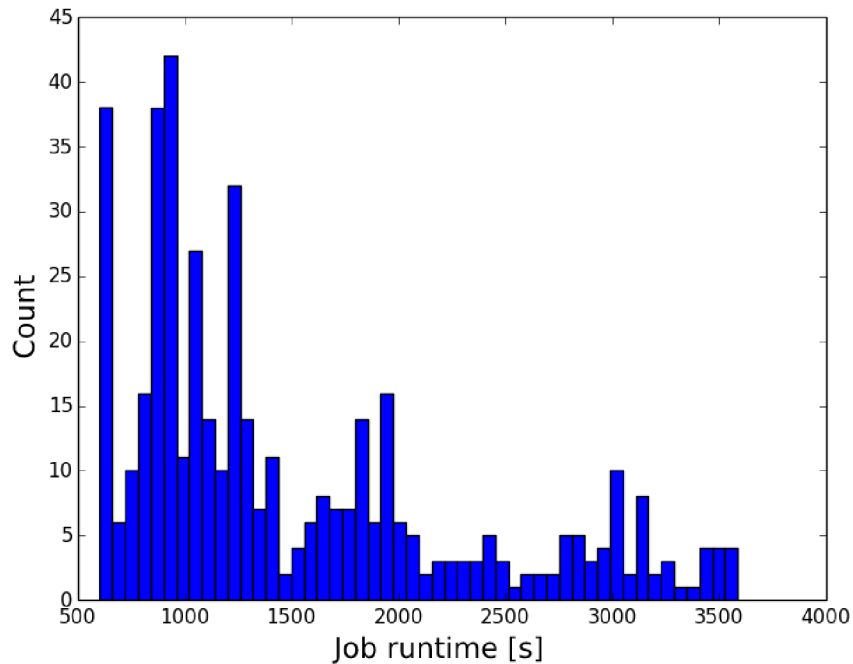


Figure 5.12: Histogram chart of job runtimes for the first small unfiltered dataset.

With the final dataset, the labelling phase was done on 500 jobs which is more than enough samples for training a neural network and took more than 20 hours time-wise. The remaining unlabelled jobs can be labelled in the future in order to provide a very representative dataset for time series machine learning algorithms.

This dataset will be anonymized and made publicly available after this thesis is defended as a cooperation result of CINECA, UNIBO and BUT.

5.4 Job Classification

The networks are divided into two categories—metric and job. The metric networks are configured the same for all metrics and the job network is connected to the outputs of each metric network. This way the labels of a metric network can be extracted, optionally modified and then set as input to the job network.

The backpropagation network was developed without any non-system library in order to be easily deployed on any machine with Python language. Libraries like Keras [7] or scikit-learn [42] were considered during development. For the sole purpose of backpropagation network, these libraries can be replaced with custom-made code.

The design of the network is split into two classes: `Network` and `Neuron`. The `Neuron` class facilitates the work of a single neuron in the network keeping its state in the class instance. The `Network` class takes care of the backpropagation and gradient descend algorithms and the import and export of a configured network.

Initial experiments consisted of various network configuration, one of which produced the best results. The final network configuration is 80 input neurons with 3 hidden layers consisting of 20 neurons, then 4 and finally 3 neurons with one output neuron. Eighty input neurons provide good performance/evaluation compromise. Larger input vectors can

result in extremely long runs and smaller input vectors might result in poor outputs of the networks.

The `jobber` network was designed similarly with the difference in input layer consisting of 12 neurons (each for one metric network), 4 and 3 neurons in the hidden layers and one output neuron.

All networks are independent of each other, therefore certain parallelization could be made during the training period. During the training period, all labelled data is loaded into memory and then served to each network. All networks were logging their progress in training by log outputs to the standard output which afterwards could be analyzed.

All networks were set to train a maximum of 25 000 epochs or until the sum error had reached 1.0. With the smaller dataset present, the networks were trained with 27 jobs keeping the other 5 jobs as the evaluation dataset. The larger dataset was split into 350 training jobs and 150 jobs as the evaluation sample.

The gradient descent algorithm was modified with momentum addition [47] in order to surpass local minimums and generally to achieve faster convergence where γ was set to 0.8 after the trial and error experiments.

$$\Delta \vec{w}_p = -(\gamma w_{p-1} + \mu \nabla E_p) \quad (5.1)$$

The dataset was randomly shuffled after each epoch in order to limit overfitting on the training dataset. The final error rates were about 15 % higher compared to non-shuffled dataset but during evaluation, the networks with shuffled dataset produced better success rate (about 5 % better).

During experiments, various configurations regarding the learning rate, maximum epochs and the momentum gradient descend presence were evaluated. The best results were generally achieved with previously mentioned configuration but certain metrics provided better results without the momentum gradient descend. Each individual config was evaluated and the best resulting configurations were combined to create a well-trained set of networks.

The final evaluation of best-performant network configurations is listed in Table 5.1. Each network was evaluated separately on the same dataset and then the full network set was presented with the input data, the metric networks evaluated the data and the outputs without modification were presented to the `jobber` network which was eventually evaluated. This single complex evaluation shown in the „complex jobber“ row is the final result.

An example of training period and the resulting error can be found in Figure A.8 where we can see multiple networks surpassing a local minimum and then finding, probably, the global minimum. These error rates were extracted from the logs as mentioned earlier.

The success rates were in the range between 70 and 100 % with most of the networks between 80 and 90 %. The higher success rate is often hard to achieve and would require deep and long-term evaluation and experiments with a larger and more diverse dataset. In the end, the resulting complex job evaluation success rate is 84 %, which can be considered a very good result.

5.5 Job Anomaly Classification

The job anomaly classification network using decision trees was after a careful consideration scrapped due to the lack of context and background information about the evaluated jobs. This network is a great start point for the further work.

Table 5.1: The final evaluation result of all trained networks.

Network name	Correctly classified	Incorrectly classified	Success rate (%)
C6res	129	21	86
C3res	150	0	100
load_core	127	23	84.67
ips	121	29	80.67
Sys_Utilization	122	28	81.33
IO_Utilization	150	0	100
Mem_Utilization	119	31	79.33
CPU_Utilization	114	36	76
L1L2_bound	140	10	93.33
L3_bound	104	46	69.33
front_end_bound	112	38	74.67
back_end_bound	108	42	72
jobber	145	5	96.77
complex jobber	126	24	84

During dataset labelling a few of possible scenarios were found, which can be summed up in several categories to set the basic leaves for the decision tree:

- unbalanced CPU utilization
- the job exited with non-zero return value
- no CPU utilization during the whole job
- non-uniform memory access
- long startup and finalization periods
- premature job ending
- sudden performance drops

5.6 Summary

The final output of this work can be considered quite wide in terms of various fields which are incorporated in it. Starting from the UI and front-end as a whole which utilizes the most modern and up-to-date technologies and paradigms, continuing to a newly made independent Python package publicly available on Python Package Index (PyPi) [16] with a full-featured back-end HTTP REST API capable of handling user and securing access to private information up to the well trained backpropagation neural network capable of classifying a job based on time series data together with a large labelled dataset which can help other researchers in their machine learning applications.

The main focus of this thesis is the final set of neural networks even though the path to fully working trained networks was quite difficult laid with many obstacles in the path mainly in the availability of the needed data and its extraction from the database which itself took more than 5 days due to technical difficulties which appeared only under extreme server load. The final configuration of all metric networks is the same with the 80-20-4-3-1 neurons in each layer. In the end, the resulting complex job evaluation success rate is 84 %, which can be considered a very good result but still one which can be worked upon and improved.

Chapter 6

Conclusions

The Examon Web was successfully presented, designed and implemented. The needed theoretical background was explained with references to underlying literature. Examon Web was set in between current state-of-the-art tools for **HPC** system monitoring with its unique place.

It was designed using modern approaches in web development such as single-page application design, **REST** API and WebSocket interface while integrating Internet of Things technologies, namely **MQTT** together with big data analysis for data acquisition.

Anomaly detection of suspicious jobs was successfully verified in a proof of concept work done during a course on soft computing and then implemented using the second, larger, dataset with the success rate of 84 % of correctly labelled jobs using only metric data of it. The implementation consists of 12 metric networks labelling metrics such as core load, **CPU** utilization, C6 **CPU** state share or back-end bound instructions shares. The final network which produces the definitive answer about the suspicious behaviour of the job takes the output of all 12 metric networks as its input and labels the job itself. The results from the metric networks are also presented to the user so the user can see what was labelled as suspicious.

Examon Web is an expansion layer to the Examon framework. With this web-based tool, the Examon framework gains completely new use-cases and audience not just amongst the research community but also in the **HPC** users community. This fact serves a critical role in the popularization of Examon itself and its possible expansion to many **HPC** facilities around the world.

Another output of this thesis is the manually labelled dataset of time-series data which can be used for research purposes other than job classification such as trend finding analytic tool or a time-series prediction tool which can be used in other fields of research as well.

The final part of the soft-computing tool was not developed due to the lack of background information of the jobs themselves and without it, any proper labelling could not be done. This part is only discussed with the suggested output labels of the decision tree.

6.1 Current Deployment

Examon and Examon Web is currently deployed on the D.A.V.I.D.E. supercomputer located in CINECA, Bologna, Italy. It had also been successfully deployed on the Galileo supercomputer for several months before being decommissioned in November 2017 and

replaced by the aforementioned D.A.V.I.D.E. The Galileo supercomputer was afterwards updated and re-instantiated in the same facility for further use.

In next months, Examon will be also deployed on a part of the Marconi supercomputer which will be the largest deployment of Examon so far.

6.2 Contributions & Impact

Examon and Examon Web were presented at the IT4Innovations' 1st User's Conference together with a poster showing the features of Examon and Examon Web as seen in Figure A.9.

Two papers about Examon Web were submitted one of which was evaluated as an innovative approach to HPC system monitoring but was unfortunately rejected in face of great papers of other researchers at the HUST 2017 conference. The second paper, submitted to ISC 2018, was reviewed as intriguing and the work as very promising but again due to other great papers and low acceptance rate the paper was rejected.

The whole implementation of Examon Web and all its parts are available online¹ as open-source and anyone can contribute to the project to extend its functionality. The labelled dataset will be anonymized and then published online as open-source as well which will help with the promotion of participating parties, namely BUT, UNIBO and CINECA and it can be used by other researchers to create interesting works.

6.3 Further Work

Further work can be focused in two directions. First is the expansion of Examon Web and making it more general for any cluster and data that are published by the Examon framework. This will include very specific and vast configuration options while maintaining the user-friendliness which is extremely hard to achieve.

The second direction is focused on the soft-computing part of the thesis, mainly on the decision trees implementation. The classification networks can be also improved to achieve better, more precise results with over 90 % success rates.

¹<https://github.com/petrstehlik/examon-web>

Bibliography

- [1] Barth, W.: *Nagios: System and network monitoring*. No Starch Press. 2008.
- [2] Benedict, S.: Energy-aware performance analysis methodologies for HPC architectures: An exploratory study. *Journal of Network and Computer Applications*. vol. 35, no. 6. 2012: pp. 1709 – 1719. ISSN 1084-8045.
- [3] Beneventi, F.: pyKairosDB. 2018. [Online; accessed 27/02/2018]. Retrieved from: <https://github.com/fbeneventi/pyKairosDB>
- [4] Beneventi, F.; Bartolini, A.; Cavazzoni, C.; et al.: Continuous learning of HPC infrastructure models using big data analytics and in-memory processing tools. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. March 2017. pp. 1038–1043. doi:10.23919/DATE.2017.7927143.
- [5] Birrittella, M. S.; Debbage, M.; Huggahalli, R.; et al.: Intel® Omni-path architecture: Enabling scalable, high performance fabrics. In *High-Performance Interconnects (HOTI), 2015 IEEE 23rd Annual Symposium on*. IEEE. 2015. pp. 1–9.
- [6] Carasso, D.: Exploring splunk. *published by CITO Research, New York, USA, ISBN*. 2012: pp. 978–0.
- [7] Chollet, F.; et al.: Keras. 2015.
- [8] CINECA: MARCONI: migration from PBSPro to SLURM scheduler. 2017. [Online; accessed 22/12/2017]. Retrieved from: http://www.hpc.cineca.it/center_news/marconi-migration-pbspro-slurm-scheduler
- [9] Computing, A.: Moab HPC Suite. 2015.
- [10] Computing, A.; Computing, G.: Torque Resource Manager. 2017. [Online; accessed 18/11/2017]. Retrieved from: <http://www.adaptivecomputing.com>
- [11] Dementiev, R.; Willhalm, T.; Bruggeman, O.; et al.: Intel Performance Counter Monitor. 2017. [Online; accessed 16/10/2017]. Retrieved from: <http://www.intel.com/software/pcm>
- [12] Dongarra, J.; London, K.; Moore, S.; et al.: Using PAPI for hardware performance monitoring on Linux systems. In *Conference on Linux Clusters: The HPC Revolution*, vol. 5. Linux Clusters Institute. 2001.

- [13] Ezzati-Jivan, N.; Dagenais, M. R.: Multi-scale navigation of large trace data: A survey. *Concurrency and Computation: Practice and Experience*. vol. 29, no. 10. 2017.
- [14] Fette, I.: The websocket protocol. 2011.
- [15] Fielding, R.: Representational state transfer. *Architectural Styles and the Design of Network-based Software Architecture*. 2000: pp. 76–85.
- [16] Foundation, P. S.: Python Package Index. 2018. [Online; accessed 015/04/2018]. Retrieved from: <https://pypi.org/>
- [17] Gimenez, A. A.; Gamblin, T.; Jusufi, I.; et al.: MemAxes: Visualization and Analytics for Characterizing Complex Memory Performance Behaviors. *IEEE transactions on visualization and computer graphics*. 2017.
- [18] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press. 2016. <http://www.deeplearningbook.org>.
- [19] Gormley, C.; Tong, Z.: *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. „O’Reilly Media, Inc.“. 2015.
- [20] Grinberg, M.: Flask SocketIO. 2018. [Online; accessed 04/01/2018]. Retrieved from: <https://flask-socketio.readthedocs.io/en/latest/>
- [21] Han, J.; Pei, J.; Kamber, M.: *Data mining: concepts and techniques*. Elsevier. 2011.
- [22] Intel: Tuning Applications Using a Top-down Microarchitecture Analysis Method. 2017. [Online; accessed 25/10/2017]. Retrieved from: <https://software.intel.com/en-us/vtune-amplifier-help-tuning-applications-using-a-top-down-microarchitecture-analysis-method>
- [23] Isaacs, K. E.; Giménez, A.; Jusufi, I.; et al.: State of the art of performance visualization. *EuroVis 2014*. 2014.
- [24] IT4Innovations: IT4Innovations Extranet. 2017. [Online; accessed 12/11/2017]. Retrieved from: <https://extranet.it4i.cz/>
- [25] Jette, M. A.; Yoo, A. B.; Grondona, M.: SLURM: Simple Linux Utility for Resource Management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag. 2002. pp. 44–60.
- [26] Lakshman, A.; Malik, P.: Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*. vol. 44, no. 2. 2010: pp. 35–40.
- [27] Lee, S.; Kim, H.; k. Hong, D.; et al.: Correlation analysis of MQTT loss and delay according to QoS level. In *The International Conference on Information Networking 2013 (ICOIN)*. Jan 2013. ISSN 1550-445X. pp. 714–717. doi:10.1109/ICOIN.2013.6496715.
- [28] Locke, D.: Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library*. 2010.

- [29] Massie, M. L.; Chun, B. N.; Culler, D. E.: The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*. vol. 30, no. 7. 2004: pp. 817 – 840. ISSN 0167-8191.
- [30] McNabb, L.; Laramée, R. S.: Survey of Surveys (SoS)-Mapping The Landscape of Survey Papers in Information Visualization. In *Computer Graphics Forum*, vol. 36. Wiley Online Library. 2017. pp. 589–617.
- [31] Mehrotra, K.; Mohan, C. K.; Ranka, S.: *Elements of artificial neural networks*. MIT press. 1997.
- [32] de Melo, A. C.: The new linux 'perf' tools. In *Slides from Linux Kongress*, vol. 18. 2010.
- [33] Mikowski, M. S.; Powell, J. C.: Single page web applications. *B and W*. 2013.
- [34] Minyard, C.: IPMI—A Gentle Introduction with OpenIPMI.
- [35] Morgan, A. G.; Kukuk, T.: *The Linux-PAM System Administrators' Guide*. 2006.
- [36] N, P.: pyKairosDB. 2018. [Online; accessed 27/02/2018]. Retrieved from: <https://github.com/pcn/pyKairosDB>
- [37] Ödegaard, T.: Grafana, The Leading Graph And Dashboard Builder For Visualizing Time Series Metrics. 2016.
- [38] Oetiker, T.: RRDtool. 2005. [Online; accessed 20/02/2018]. Retrieved from: <https://oss.oetiker.ch/rrdtool/>
- [39] Olups, R.: *Zabbix 1.8 network monitoring*. Packt Publishing Ltd. 2010.
- [40] Otto, M.; Thornton, J.; et al.: Bootstrap. *Twitter Bootstrap*. 2013.
- [41] Passint, R.; Thorson, G.; Galles, M.: Hybrid hypercube/torus architecture. May 8 2001. uS Patent 6,230,252. Retrieved from: <https://www.google.com/patents/US6230252>
- [42] Pedregosa, F.; et al.: Scikit-learn: Machine learning in Python. *Journal of machine learning research*. vol. 12, no. Oct. 2011: pp. 2825–2830.
- [43] Pfister, G. F.: An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O*. vol. 42. 2001: pp. 617–632.
- [44] Rai, R.: *Socket. IO Real-time Web Application Development*. Packt Publishing Ltd. 2013.
- [45] Reinders, J.: VTune performance analyzer essentials. *Intel Press*. 2005.
- [46] Ronacher, A.: Flask (A Python Microframework). 2018. [Online; accessed 04/01/2018]. Retrieved from: <http://flask.pocoo.org/>
- [47] Ruder, S.: An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*. 2016.

- [48] Showerman, M.: Real Time Visualization of Monitoring Data for Large Scale HPC Systems. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE. 2015. pp. 706–709.
- [49] Stefanov, K.; Voevodin, V.; Zhumatiy, S.; et al.: Dynamically reconfigurable distributed modular monitoring system for supercomputers (DiMMon). *Procedia Computer Science*. vol. 66. 2015: pp. 625–634.
- [50] Stehlík, P.: muapi (modular user-oriented REST API). 2018. [Online; accessed 15/05/2018]. Retrieved from: <https://github.com/petrstehlik/muapi>
- [51] UNIBO: DEI UNIBO Website. 2018. [Online; accessed 20/12/2017]. Retrieved from: <http://www.dei.unibo.it/en>
- [52] Vanderkam, D.; Allaire, J.; Owen, J.; et al.: dygraphs: Interface to ‘Dygraphs’ Interactive Time Series Charting Library. *R package version 0.5*. 2015.
- [53] Victor Savkin, V. B.: Angular. 2018. [Online; accessed 04/02/2018]. Retrieved from: <https://angular.io/>
- [54] Voevodin, V.; Voevodin, V.; Shaikhislamov, D.; et al.: Data mining method for anomaly detection in the supercomputer task flow. In *AIP Conference Proceedings*, vol. 1776. AIP Publishing. 2016.
- [55] Works, A. P.: PBS Professional®14.2 Administrator’s Guide. 2017.
- [56] Works, A. P.: PBS Professional®14.2 Plugins („Hooks“) Guide. 2017.
- [57] Yasin, A.: A top-down method for performance analysis and counters architecture. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE. 2014. pp. 35–44.

Appendix A

Appendices

A.1 PBS Pro Hooks Lifecycle

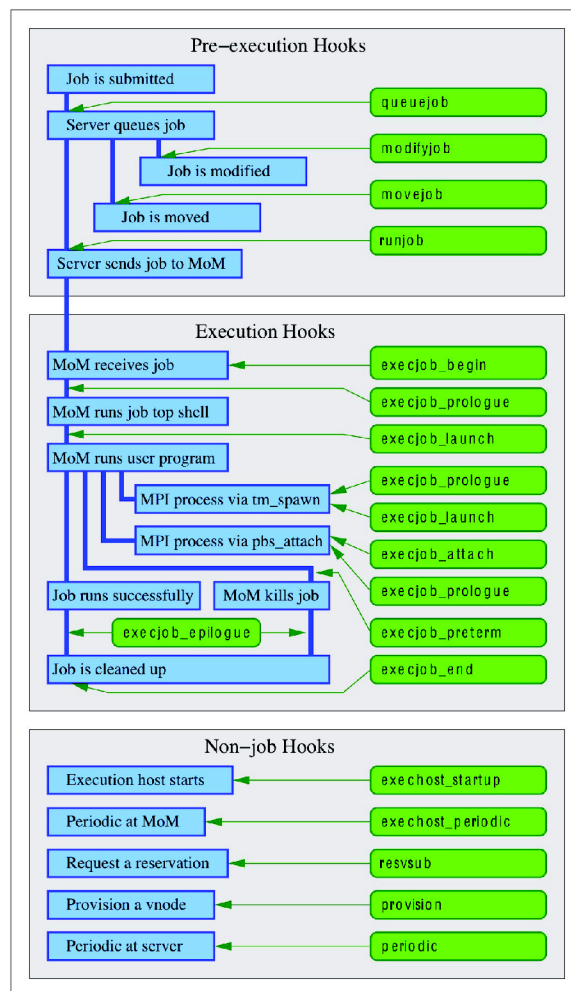


Figure A.1: Simplified view of hook trigger timing. Taken from [56].

A.2 Examon Web Job Info Wireframe

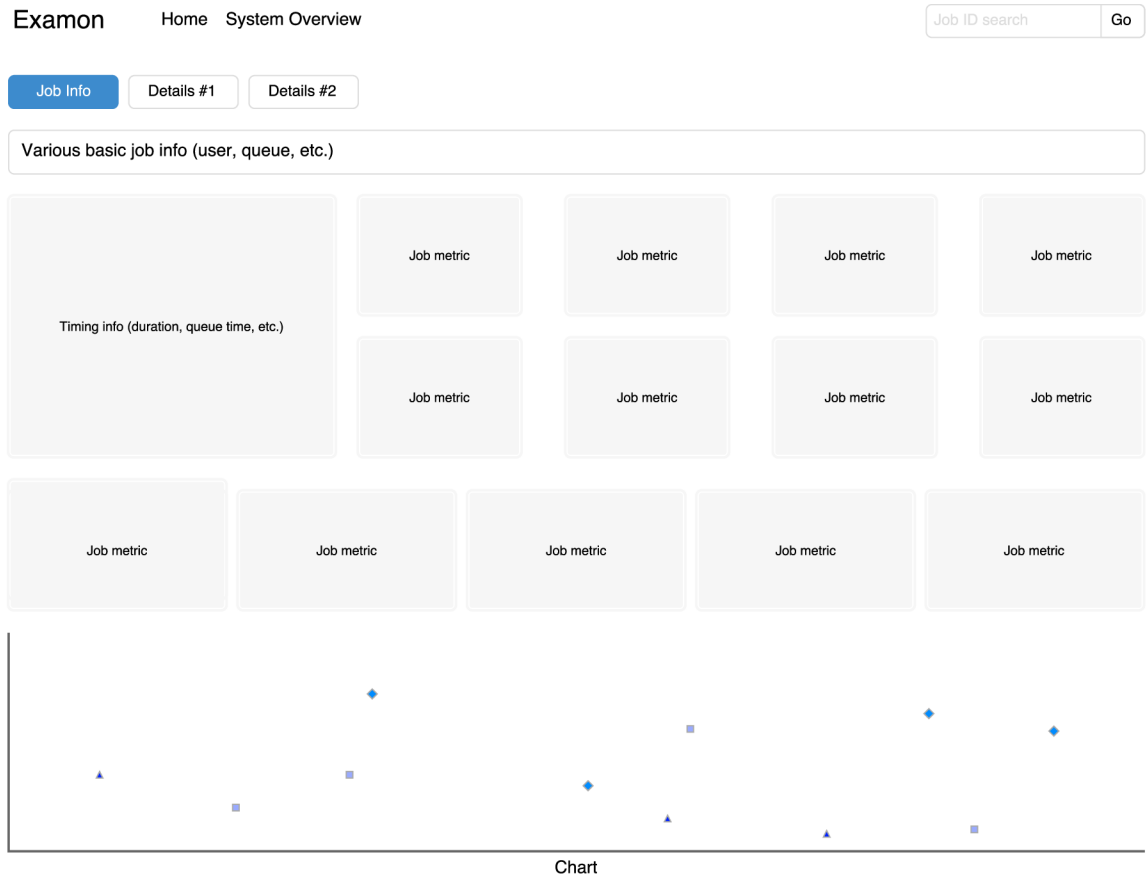


Figure A.2: Low-fidelity wireframe of job overview dashboard with all the job info and one chart showing core loads.

A.3 Examon Web Cluster Overview Wireframe

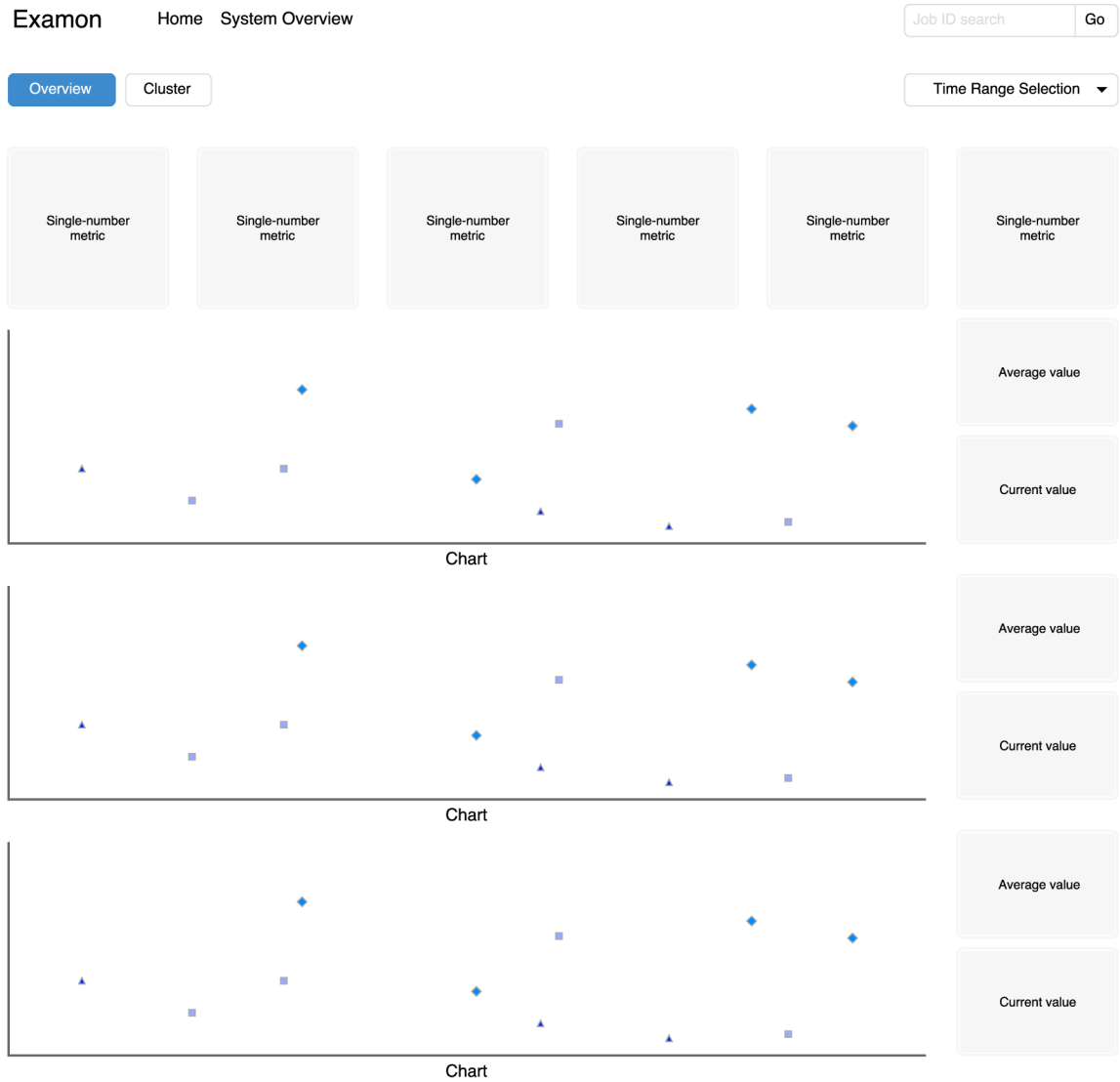


Figure A.3: Low-fidelity wireframe of cluster overview dashboard with several single-number metric boxes and charts with average and current values next to the right.

A.4 Example Jobs Spotted During Labelling

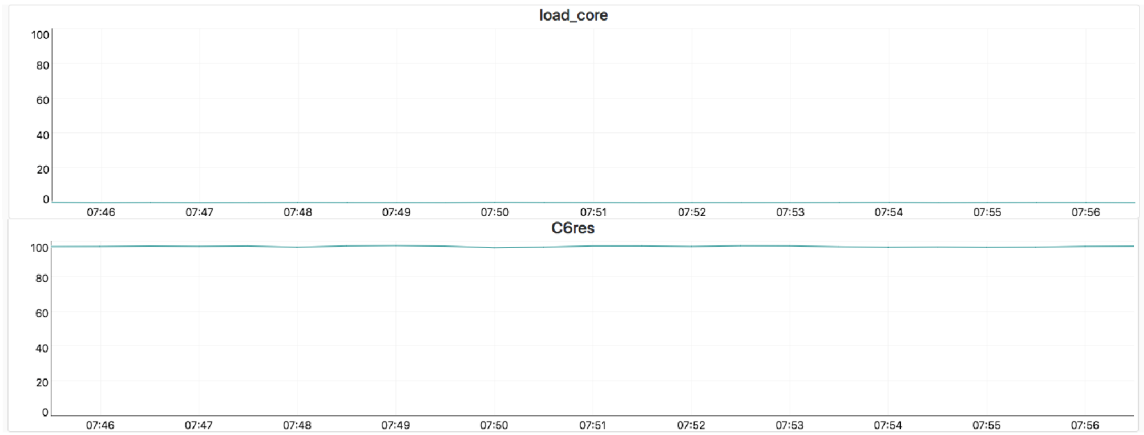


Figure A.4: A job with no load the CPU in C6 state during the whole runtime. Other metrics are not included.

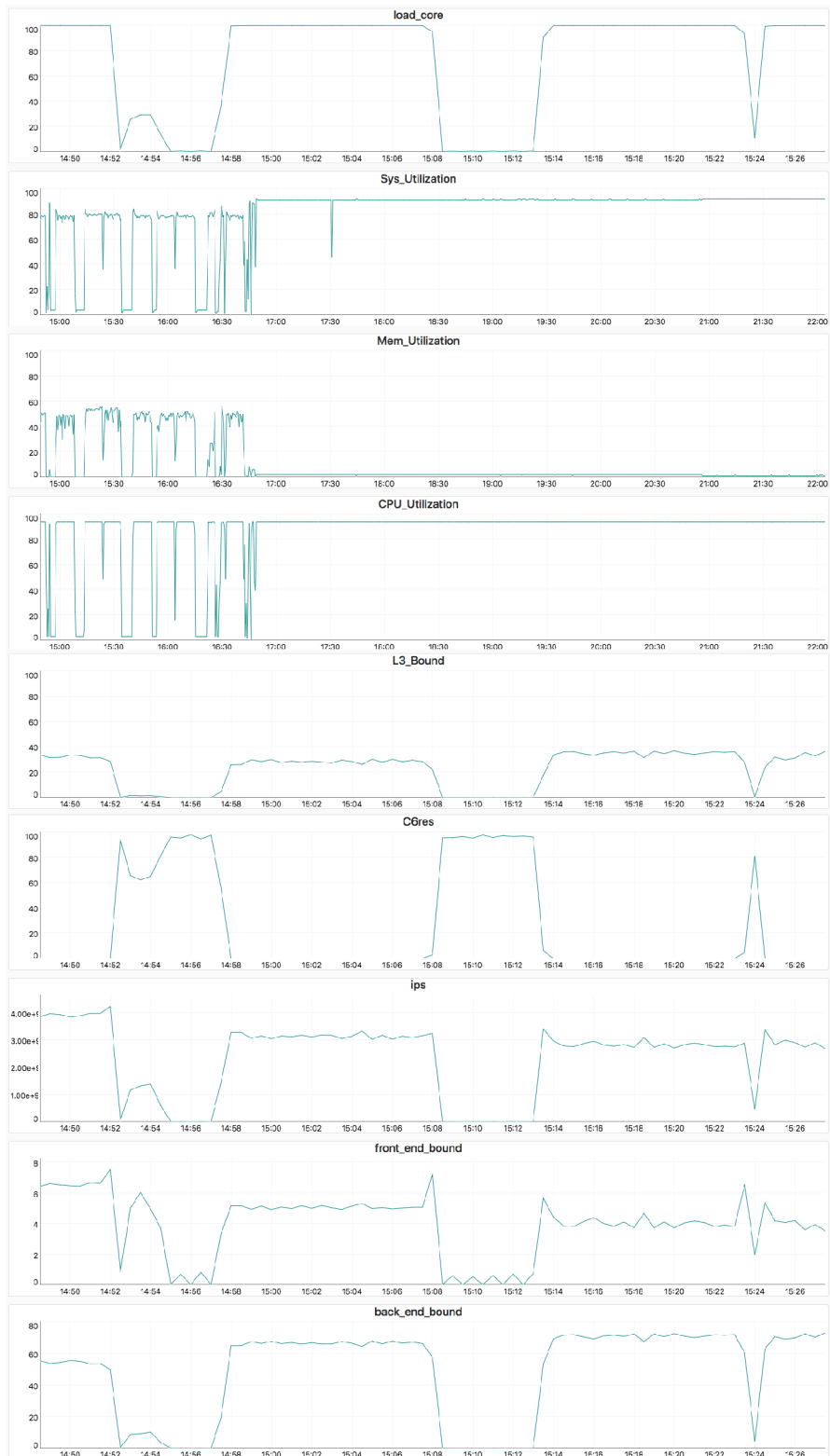


Figure A.5: A job with suspicious drops in utilization. Unsuspicious metrics were removed.



Figure A.6: A job with balanced behavior utilizing all allocated resources. Unsuspecting metrics were removed.

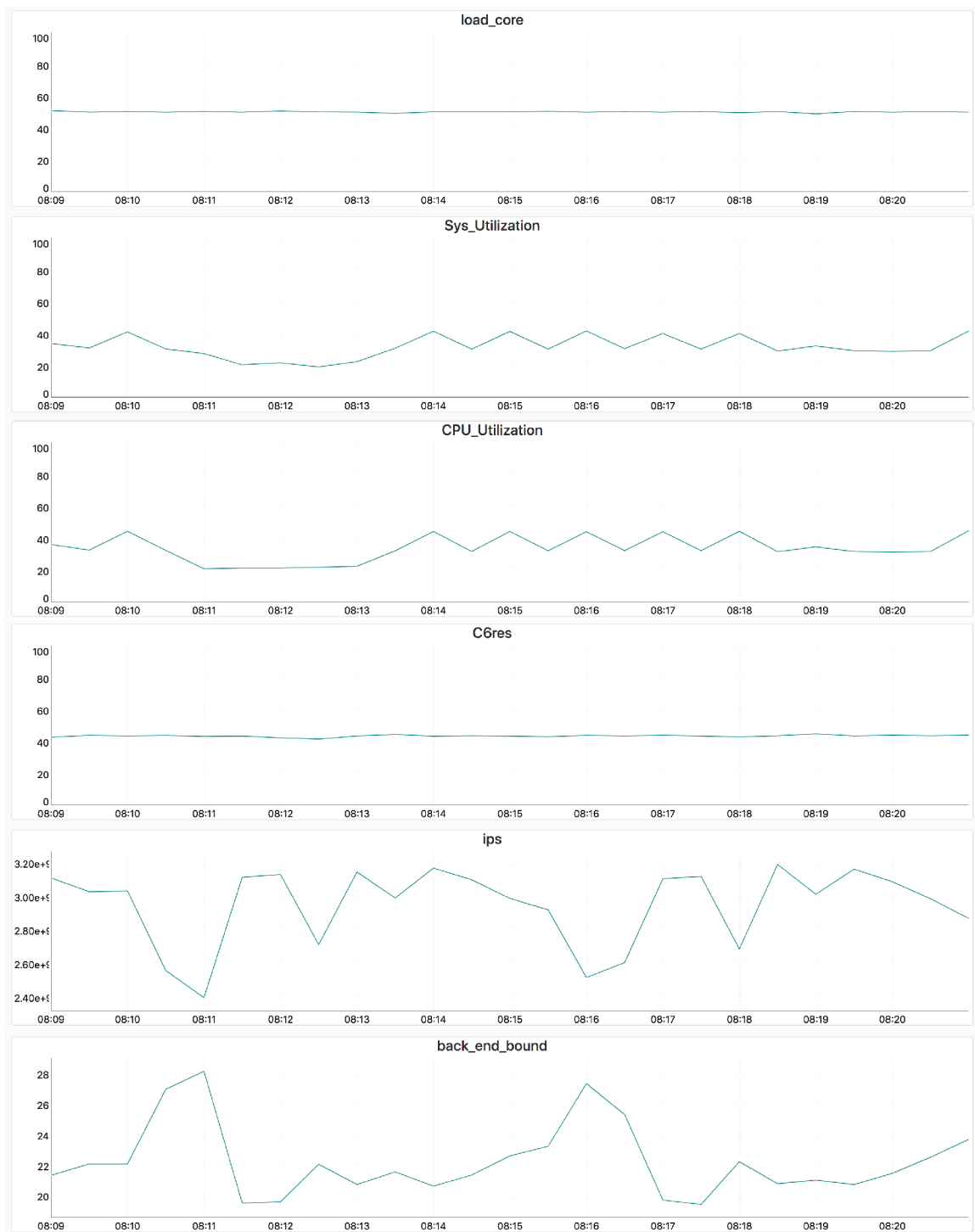


Figure A.7: A suspicious job with 50 % core utilization. Unsuspicious metrics were removed.

A.5 Networks' Error Rates

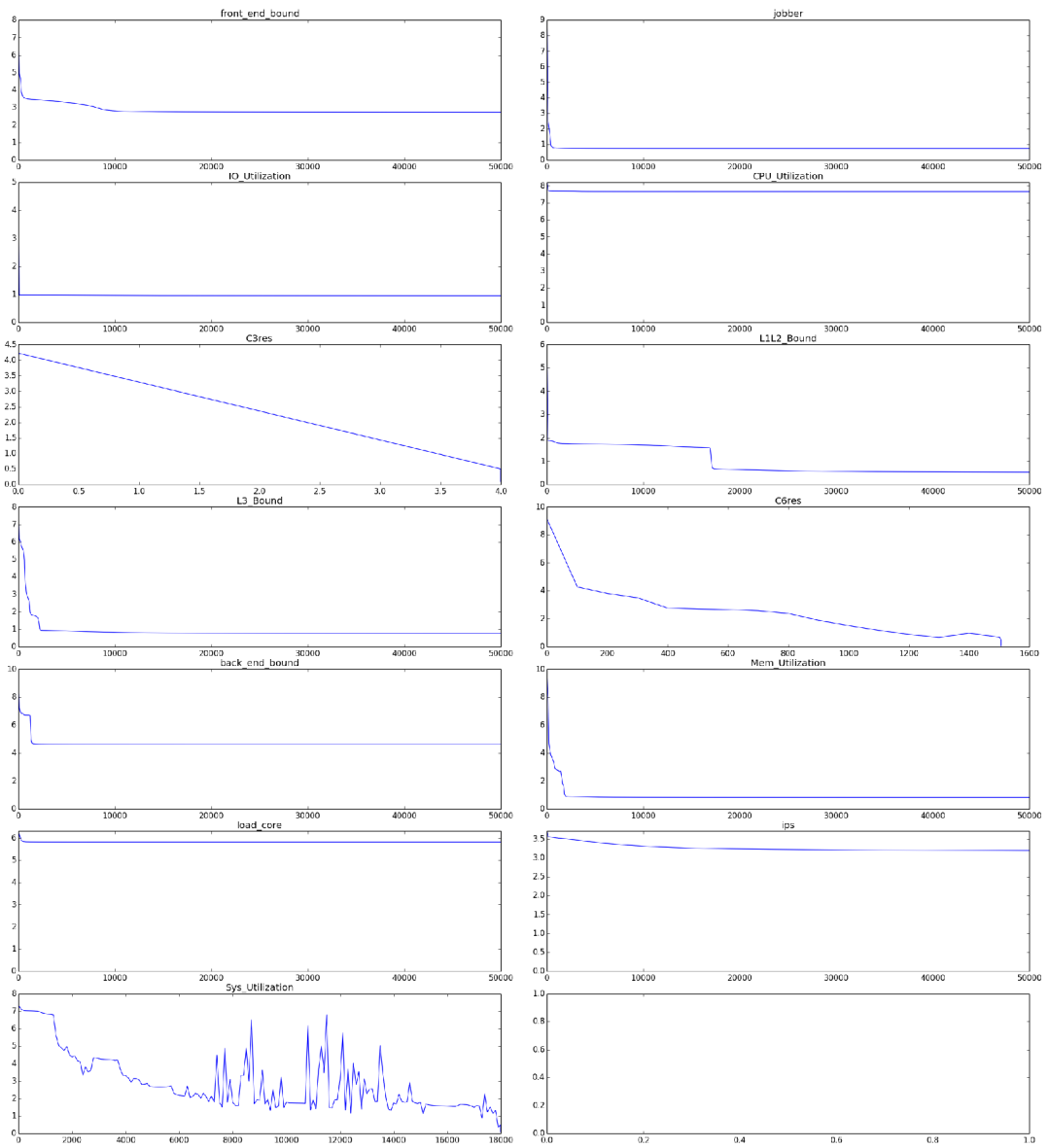


Figure A.8: The sum error rates during the training period of all networks.

A.6 Poster for IT4Innovations' 1st User Conference

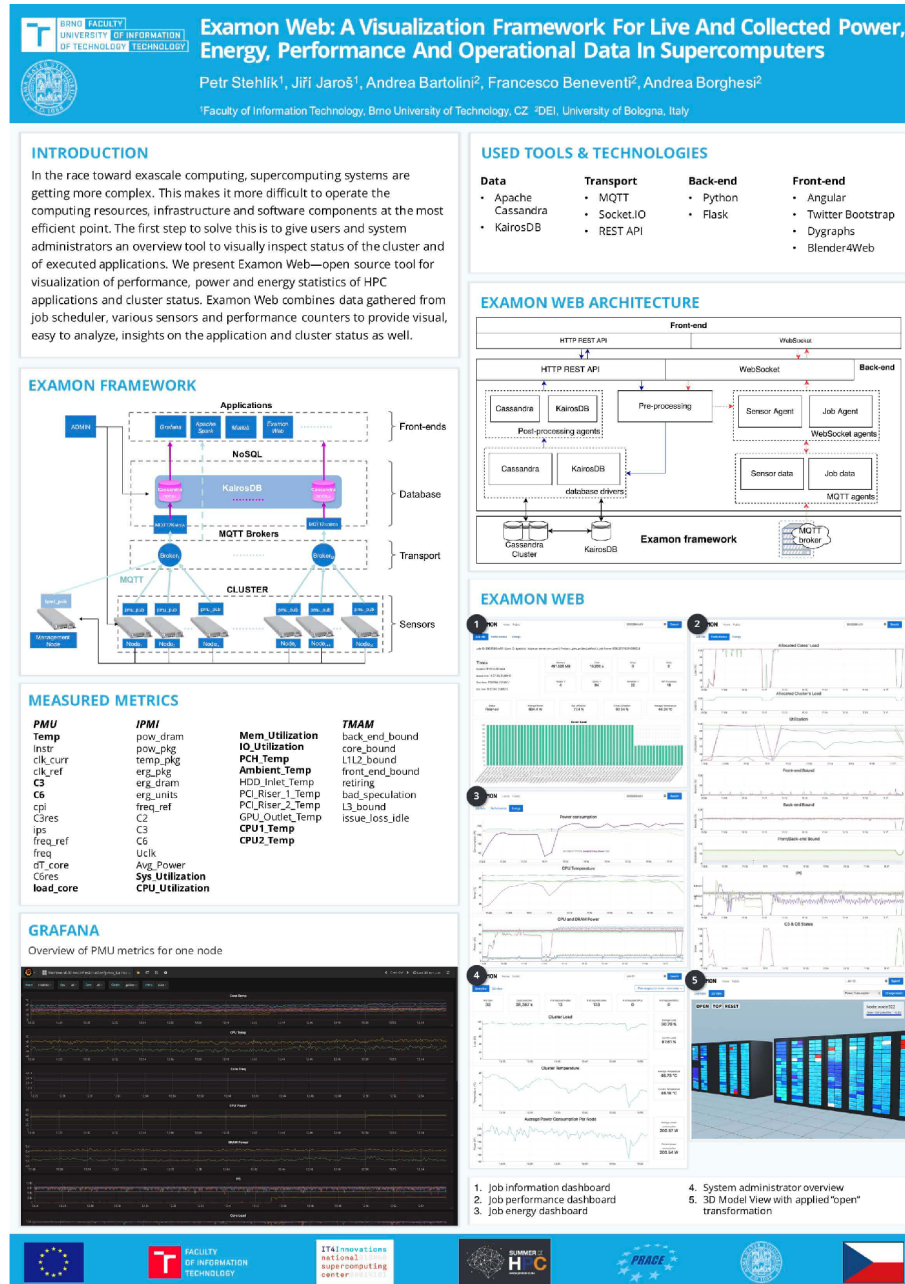


Figure A.9: The poster presented at the IT4Innovations' 1st User Conference.

A.7 Contents of the Attached Media

- `docs` – L^AT_EX source code of the thesis
- `xsteh14.pdf` – thesis in PDF file
- `examon-web` – source code with GIT history of Examon Web containing all 3 versions of front-end in GIT branches
- `dataset` – large partially annotated dataset used for training neural networks
- `poster.pdf` – poster presented at the IT4Innovations' 1st Users' Conference in full resolution