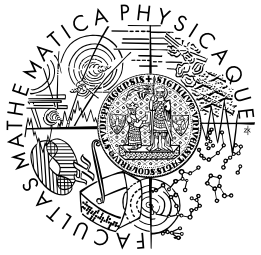

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Adam Filandr

Latency aware deployment in the edge-cloud environment

Department of Distributed and Dependable Systems

Supervisor of the master thesis:	doc. RNDr. Petr Hnětynka, Ph.D.
Study programme:	Computer Science
Study branch:	Software and Data Engineering

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I would like to thank my supervisor and consultants doc. RNDr. Petr Hnětynka, Ph.D., RNDr. Iveta Hnětynková, Ph.D., doc. RNDr. Tomáš Bureš, Ph.D., doc. Ing. Lubomír Bulej, Ph.D., and prof. RNDr. Tomáš Skopal, Ph.D. for their advice and time. I also want to give special thanks to my family and friends for their continued support.

Title: Latency aware deployment in the edge-cloud environment

Author: Bc. Adam Filandr

Department: Department of Distributed and Dependable Systems

Supervisor: doc. RNDr. Petr Hnětynka, Ph.D., Department of Distributed and Dependable Systems

Abstract: The goal of this thesis is to propose a layer on top of edge-cloud, in order to provide soft real-time guarantees on the execution time of applications. This is done in order to satisfy the soft-real time requirements set by the developers of latency-sensitive applications. The proposed layer uses a predictor of execution time, in order to find combinations of processes with, which satisfy the soft real-time requirements when collocated. To implement the predictor, we are provided with information about the resource usage of processes and execution times of collocated combinations. We utilize similarity between the processes, cluster analysis, and regression analysis to form four prediction methods. We also provide a boundary system of resource usage used to filter out combinations exceeding the capacity of a computer. Because the metrics indicating the resource usage of a process can vary in their usefulness, we also added a system of weights, which estimates the importance of each metric. We experimentally analyze the accuracy of each prediction method, the influence of the boundary detection system, and the effects of weights.

Keywords: edge-cloud performance prediction cluster-analysis regression-analysis

Contents

1. Introduction

Cloud computing is the technique of using a network of remote servers to provide on-demand delivery of computing power, database, storage, applications, etc. This computing technique is becoming an increasingly important part of today's technological ecosystem, with many major companies providing such systems (Google, Microsoft, Amazon, IBM). The advantages of using cloud systems when compared to in-house solutions include convenience, scalability and low cost. Some well-known examples of usage of cloud computing include applications such as Netflix, Facebook, Dropbox, platforms such as Google App Engine, Windows Azure, and infrastructures such as Amazon Web Services' EC2.

A large variety of different applications can be developed using cloud computing, however, there are also categories of applications, which are hard to develop using traditional cloud systems. One of them is the category of latency-sensitive applications, which is of interest to us. These applications might include software for self-driving cars, Internet of Things, or augmented reality applications. In traditional cloud computing systems, the servers hosting the user activity are physically located in large data centers serving entire regions (for example, a data center serving western Europe), which is one of the main problems for latency-sensitive applications. The distance between the client and the data center is not trivial, thus neither the latency is trivial, and clients from different parts of the region might experience different latency. Also, the traditional cloud systems may not guarantee the performance of a hosted application. As a result, applications can be slowed down by other applications running on the same server, which again increases latency and produces jitter as the slowdown is not constant.

Edge-cloud computing[1] is a new computing paradigm, which fights the latency issues by decentralizing the data centers into micro data centers that are geographically closer to the client. By itself, this is not enough to satisfy the requirements of latency-sensitive applications, and at least soft execution time guarantees have to be provided, which is the focus of our work.

This calls for an additional layer on top of the edge-cloud, which manages deployment and redeployment of services in edge-cloud based on the specification of timing requirements of individual processes running in the cloud. The goal of this thesis is to propose such a layer, which, given the knowledge of performance and resource consumption of the processes, predicts the execution time of a set of processes running on a server and finds optimal deployment (with regards to the applications' real-time requirements on execution time).

The thesis is structured as follows. Chapter 2 focuses on the concept of edge-cloud computing and the problem we want to solve in greater detail. Chapter 3 provides an analysis of our situation and the proposed layer. In Chapter 4 we present our approach to predicting the execution time and in Chapter 5 the implementation details. Finally Chapter 6 provides evaluation of our approach and Chapter 7 concludes the thesis.

2. Cloud and edge-cloud computing explained

In this chapter, we explore and compare cloud and edge-cloud computing. Then we examine the relation of both paradigms to latency-sensitive applications and explain in detail what problem our thesis focuses on.

2.1 Cloud computing

Cloud computing started gaining prominence since its inception around 2005[2]. It has been identified as the next computing paradigm in the evolution from mainframes to PCs, networked computing, the internet and grid computing, with effects as profound as the move from mainframes to PCs[3].

The US National Institute of Standards and Technology (NIST) defines cloud computing as[4]: *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

Cloud computing systems are usually implemented as centralized data centers, in which the cloud provider manages servers, which host the customers' activity. To support all of the required features, a range of other existing technologies are combined, such as parallel computing, distributed computing, virtualization, containerization, orchestration and many more. As a result, we can think about the cloud system as a collection of hardware and software running in a data center that enables cloud computing[5].

A customer connects to the cloud system, usually through the Internet, and is provided with various cloud services. One of the implications of the NIST definition is, that the cloud computing paradigm transforms computing power into a service with certain key properties, for which a customer pays according to use. This is in contrast with computing power being hardware, for which the customer pays an upfront cost. Computing power as a service is not a new idea (e.g., renting a server), thus the properties which these services are required to satisfy, in order to be classified as cloud services, is what makes cloud computing novel and appealing[6]:

- On-demand self-service: A customer can acquire server time, storage, or other computing capabilities automatically without the need of other human intervention.
- Broad network access: The customer accesses the cloud services over the network through standard mechanisms using a client platform (mobile, laptop, etc.).
- Resource pooling: The provider pools the computing resources to serve multiple customers. Different physical and virtual resources are dynamically assigned and reassigned according to customer demand.

- Location independence: The customer has no knowledge of the exact location of the provided resources (the infrastructure is transparent to him). However, the customer might be able to specify the broad location in which he wishes to acquire the service (e.g., western Europe).
- Rapid elasticity: The acquired services can be automatically provisioned or released in order to scale according to demand.
- Measured service: Resource usage (such as storage, bandwidth use, etc.) is monitored, controlled and reported in order to better inform the customer and the provider and to enable the optimization of resource use.

The services provided to the customer are typically divided by the level of responsibility, which the provider adopts, into three categories[7]:

- Software-as-a-Service (SaaS): Customers of this cloud service only make use of an application hosted by the cloud provider, which can be accessed using clients such as a web browser or dedicated application. The customer does not manage the application or the infrastructure (such as servers, operating systems, storage, etc.). Thus the service, for which the customer pays, is the ability to use a given software application. Examples of this model include DropBox, Slack, Microsoft Office 365 and Google Apps for Work.
- Platform-as-a-Service (PaaS): Customers of this cloud service can release their applications, developed using programming languages, libraries and tools supported by the cloud provider in a hosting environment. Users can again access these applications through various clients. Similarly to SaaS, the customer does not have control over the underlying infrastructure, but the customer has control over his deployed applications. Thus the service, for which the customer pays, is the ability to develop and deploy applications using a platform managed by the cloud provider. Examples of this model include Microsoft Azure and Google App Engine.
- Infrastructure-as-a-Service (IaaS): The cloud provider manages the processing, storage, networking, power, security, virtualization and other aspects of the infrastructure, which the customer uses to run arbitrary software. The customer has limited or no ability to manage underlying physical infrastructure. Thus the service, for which the customer pays, is the ability to use remote hardware managed by the cloud provider. Examples of this model include Amazon Web Services' EC2 and S3.

Cloud computing systems are also divided into different categories depending on who owns and who uses them[4]:

- Private cloud: The cloud system is created for and used by a single organization. This organization, some third party, or a combination of them own the system.
- Community cloud: The cloud system is created for and used by a specific community (e.g., academic community). An organization from this community, some third party, or a combination own the system.
- Public cloud: The cloud system is created for general use by the public. It is owned by a business, academic, or government organization or a combination of those.

- Hybrid cloud: This cloud system is a composition of more than one distinct cloud system (public, community or private), which are bound together by a technology that enables the portability of data and applications.

Advantages and disadvantages

Key advantages of cloud computing consist of scalability, the gradual payment model, cost-efficiency and ease of use.

Key disadvantages of cloud computing consist of issues with downtime, security, privacy, limited control over infrastructure, flexibility and vendor lock-in. Vendor lock-in is the problem of transferring activity from one cloud provider to another. It can be divided into risks of transfer of data, applications, infrastructure and human resource knowledge risk. In regards to latency-sensitive applications, the main problem of cloud computing is the significant distance latency between the user and the data center and the lack of guarantees on the performance of an application.

It is clear that without sufficient planning and carefully executed development, the use of a cloud computing system can become problematic. However, as we can see by the wide adoption of this computing paradigm, the positives outweigh the negatives significantly.

2.2 Edge-cloud computing

One can think of edge-cloud computing as an extension of the idea of cloud computing explored in the previous section. Multiple definitions of edge-cloud computing exist, such as:

- *Edge-cloud computing is where compute resources, ranging from credit-card-size computers to micro data centers, are placed closer to information-generation sources, to reduce network latency and bandwidth usage generally associated with cloud computing.*[8]
- *Edge-cloud computing consists of any computing and network resources along the path between data sources and cloud data centers.*[9]

In general, the main idea behind edge-cloud computing is to decentralize the computing power from a few large data centers into many smaller computational units (micro data centers (MDCs), cloudlets, devices, etc.) closer to the users (the data generators). This is done in order to reduce the latency between the user and the data center and to distribute the network usage into multiple locations.

When compared, edge-cloud computing could be, in the broad sense, almost identical to cloud computing, with the only difference being that the data centers are decentralized. Technically, the two systems would differ significantly, however, the main concepts of cloud computing still apply, such as the properties of cloud services, the concepts of SaaS, PaaS and IaaS and the differentiation of cloud systems into the public cloud, private cloud, etc.

There are also other architectures of edge-cloud computing systems possible, such as edge-cloud being a layer, in which only partial computation occurs

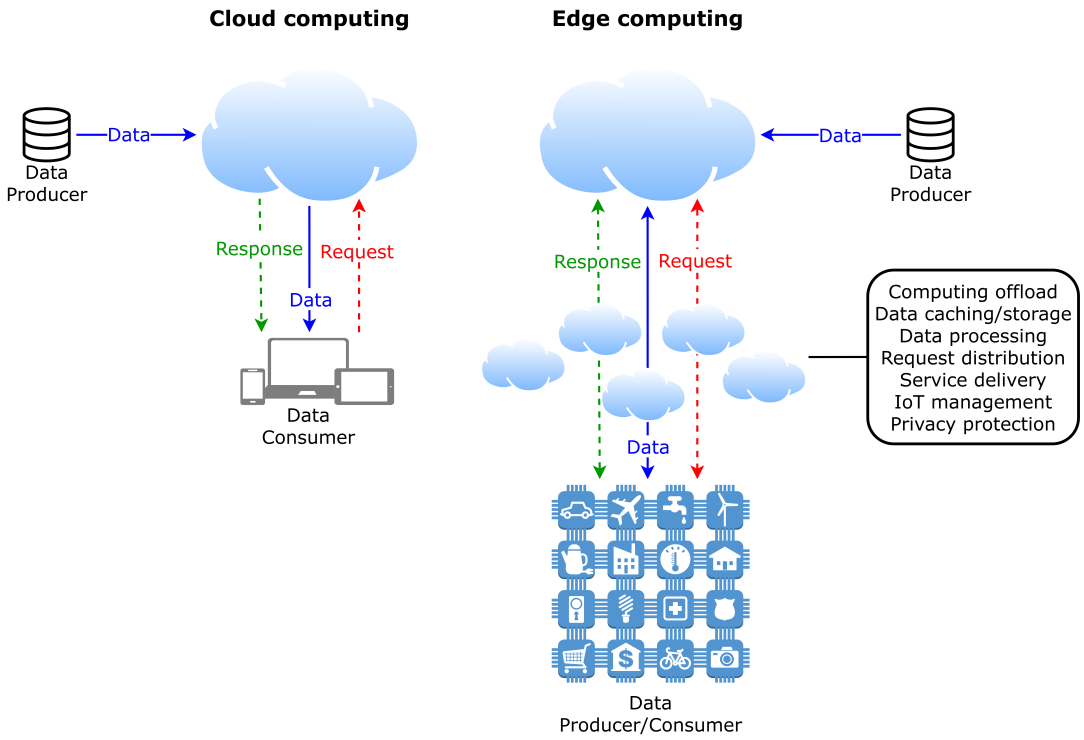

Figure 2.1: Cloud computing compared to three-layer edge-cloud computing[9].

between the user and the centralized data center, such as in Figure 2.1. Thus cloud computing and edge-cloud computing do not have to be exclusive but can complement each other. However, in this thesis, we only consider the edge-cloud systems with two layers – the MDCs and the users. For our purposes, the three-layer design does not make much difference.

Importance of edge-cloud computing for the future

One of the main reasons behind the advent of edge-cloud computing is the change of users from being only data consumers into being both data consumers and data producers[9]. The amount of data created by people and machines on the edge of the network is steadily increasing, and with that comes increased demand in data processing and other related computationally demanding tasks.

There is a strong incentive to use cloud computing to process the data in order to save the battery life of the devices generating this data and increase the processing speed. However, to process all this data in centralized data centers would create a heavy load on the network infrastructure. Also, latency-sensitive applications would experience problems due to the vast distances between the user and the data center. As a result, edge-cloud is needed to satisfy this increased demand in computing.

Examples of promising edge-cloud use cases include IoT data processing, implementation of smart home/city, augmented reality and self-driving cars/drones[9] as visualized in Figure 2.2.

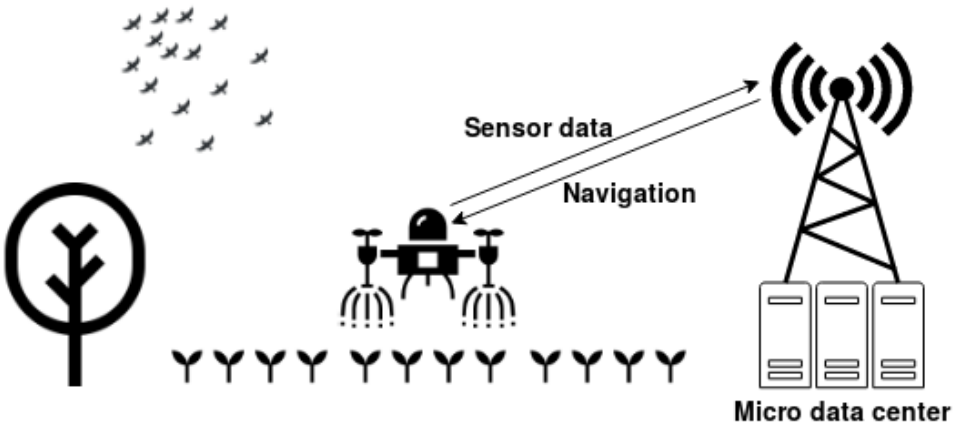

Figure 2.2: Ilustration of a farming drone equipped with sensors and simple computer using an edge-cloud self-driving software.

Advantages and disadvantages

Key advantages of edge-cloud computing consist of decentralization, low latency (due to the smaller distance between the user and the MDC) performance improvement and battery saving due to computation off-loading. Also, the same advantages of cloud computing apply to edge-cloud computing. Several demonstrations of advantages of computation offloading to the edge-cloud exist, such as:

- Reducing 20 times the running time and energy consumption of tested applications[10].
- Reduction of the response time of a face recognition application from 900 to 169 ms by moving the computation from cloud to the edge-cloud[11].
- Reduction of response time by 80-200ms and reduction of energy consumption by 30%-40% of wearable cognitive assistance[12].

Key disadvantages of edge-cloud computing consist of problems with privacy, security, availability, cost and consistency. In regards to latency-sensitive applications, the main problem of edge-cloud computing is the lack of guarantees on the performance of an application. Also, edge-cloud computing suffers from the same disadvantages as cloud computing.

2.3 Guaranteeing the performance of latency-sensitive applications

In the 5G Automotive Vision white paper[13], the 5G-PPP (initiative between the European Commission and European ICT industry) introduces a multitude of technologies focused on integrating the automotive and the telecom industry. These technologies include mostly latency-sensitive applications such as cooperative collision avoidance of cars, augmented reality windows, utilization of sensors distributed across the city for self-driving, traffic synchronization and many more.

Importantly, the relocation of complex tasks from computing units in vehicles and IoT to a remote server is argued to be possible and recommended in 5G infrastructures. In order to use this remote computing, it is necessary to provide low latency and high reliability.

To implement this remote computing, cloud computing is insufficient due to having extensive and variable distance latency (time needed to transfer a request from the user to the data center).

As a result of using decentralized data centers located closer to the user, the edge-cloud significantly reduces the distance latency. However, the base architecture of an edge-cloud system does not provide a guarantee on round trip time (RTT). RTT is the time needed to get a response for a request issued by a user. RTT consists of two times the distance latency (to and from the server) and the time needed to calculate the response.

The issue is that developers are expected to have soft real-time requirements on the RTT for their latency-sensitive applications. For example, developers can demand, that the RTT for their self-driving or augmented reality application to be less than 70ms, otherwise the applications would be unusable (e.g., self-driving car would not react fast enough).

Imagine an edge-cloud provider having an MDC composed of several servers, which covers a single city with distance latency guaranteed to be less than 20 ms, and the service provided is in the form of hosting latency-sensitive applications. If the provider allocated applications on servers using a trivial strategy (e.g., randomly), it is very likely, that the applications would slow each other down to the point of exceeding the required RTT, making the applications unusable.

The focus of our thesis is to solve this problem and provide a soft guarantee on the RTT for applications running in the edge-cloud. Soft guarantee on the RTT means that a request issued to the edge-cloud system will return in a specified amount of time with a specified probability (e.g., a request to apply a filter on an image will return in 30ms with 90% probability).

It is important to note that we do not focus on providing full real-time guarantees (every request finishes in a specified amount of time). The reason being, that such guarantees are the domain of real-time programming, which comes at a very high price of forcing developers to a low-level programming language, limited choice of libraries and the use of a relatively exotic programming model of periodic non-blocking real-time tasks.

3. Situation analysis

In this chapter, we first propose an edge-cloud layer to provide soft guarantees on the RTT. Then we examine the way we expect the processes to run in the edge-cloud nodes. In the end, we focus on the form and measurement of the provided data, which influences the solution to the problem.

3.1 Proposed layer

As highlighted in Chapter 1 and further explained in Section 2.3, the focus of our thesis is to provide soft-guarantee on the RTT for applications in the edge-cloud system.

Two factors are contributing to RTT in an edge-cloud system – the distance between the user and the MDC and the amount of slowdown caused by interference between processes. In order to minimize the latency caused by distance, edge-cloud MDCs are moved closer to the user. As a result, only the minimization of process interference needs to be solved.

It is expected that the developers use standard high-level programming languages (such as Java or Python) and that the cloud provider uses standard cloud technologies (such as Docker and Kubernetes). Therefore the minimization of interference between processes is achieved by deployment of suitable combinations of processes on individual servers, not by the use of low-level real-time programming languages or real-time programming models.

As a result, the cloud provider faces a dilemma: how to deploy processes to the servers available in a given MDC in such a way that would achieve:

- Minimal interference between the collocated processes
- Satisfaction of the soft real-time requirements
- Minimal number of servers used (maximize the resource utilization of servers)

To solve this problem, we propose a new layer for the edge-cloud composed of a predictor of the execution time of processes and a solver, as visualized in Figure 3.1. The edge-cloud architecture would then consist of:

- Central datacenter – decides which processes should run in which MDC
- Predictor and solver – accept deployment request of a set of processes to an MDC and finds suitable combinations of processes to collocate on the individual servers
- MDCs – run combinations of processes on its servers

3.1.1 Finding optimal deployment using predictor

The problem of deployment and redeployment can be solved relatively easily using a predictor of execution time. The solver iterates through a set of process combinations and requests a prediction for each combination. The predictor issues a prediction stating the predicted performance and whether the combination

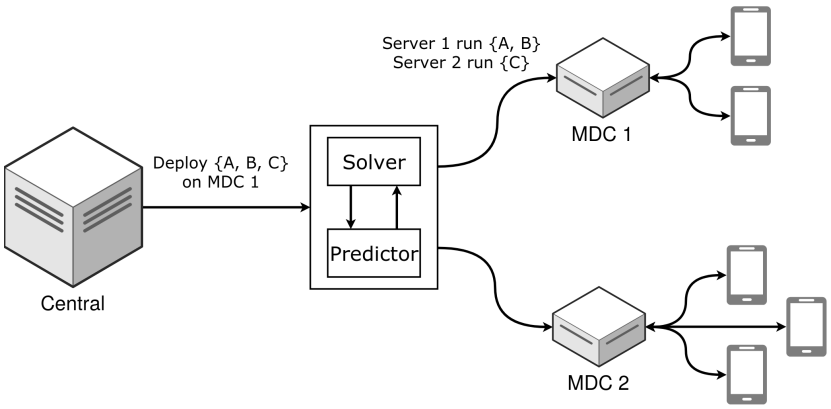

Figure 3.1: Ilustration of the edge-cloud architecture with the proposed layer.

satisfies the soft real-time requirements or not. The solver can then select a deployment with the fewest used servers or with combinations of processes with the best performance.

A trivial solver would iterate through all combinations of processes and use the predictor to decide if a given combination satisfies the soft real-time requirements. A non-trivial solver design could be, for example, in the form of formulating and solving the optimal deployment as a constraint satisfaction problem.

However, a more complex solver can only decrease the number of combinations to be predicted (thus lower the time needed to find the deployment). To provide the soft real-time guarantees, the quality of the predictor is the decisive factor. Thus our thesis focuses primarily on the predictor and not on the solver.

To estimate if a set of collocated processes satisfy the soft real-time requirements on the RTT, we need to predict the percentile of the execution time of each of the processes. The reason being, that if a process is required to have, for example, execution time shorter than 30 ms in 90% of calculations, we can express the same requirement as having the 90th percentile of execution time shorter than 30 ms.

As a result, the construction of a sufficiently accurate predictor of the percentile of the execution time is needed to satisfy the goals of this thesis. Section 3.3 describes the data provided to the predictor, the design of the predictor is described in Chapter 4, and the implementation details are explained in Chapter 5.

3.2 Process and node behavior

The expected behavior of processes can be summarised as follows:

- The processes run in isolated containers
- The containers can be moved between nodes in the same MDC
- The processes receive requests from client applications on the user's devices or from other processes
- The processes can communicate with other processes running in the same MDC (processes can depend on other processes)
- The processes can not communicate with processes running in other MDCs

Even though the processes are isolated in containers, we expect them to share the CPU. Thus instead of each process acquiring one CPU core for itself, the

computational load is distributed among all cores equally. The same principle also applies to memory, disk and other shared resources. Even if the node had multiple disks or memory cards, we still regard them as one resource. This is important as we do not have to deal with the issue of assigning applications on different CPU cores, disks or memory cards.

Rate of requests

The expected rate of requests is 100% – when a process finishes the current calculation, it starts a new calculation immediately. The reason is, that when using the edge-cloud system, a new clone of a process would be spawned only after the previous one can not handle the request rate anymore. Thus at any time, there would be maximally one clone of a process without a 100% request rate. It is preferable to still treat it as a 100% request rate process during the prediction as the request rate can climb unexpectedly.

Process dependency

The processes running in a given MDC can depend on other processes running in the same MDC, for example, a face recognition application (depender) can use a database (dependee). In this situation, we expect the dependee process to run with an acceptable slowdown. If this condition cannot be guaranteed, then both the depender and the dependee can not be assigned as the performance of the depender can be unpredictably affected.

We regard the communication latency between processes in the same MDC to be trivial. On the other hand, we expect that processes do not communicate across MDCs, as the communication latency would be automatically too high (if the communication latency would also be trivial, these two MDCs could be merged into one).

Distance latency

The communication latency caused by the distance between the user and the MDC does not necessarily have to be trivial. The information about the expected latency (probably the highest possible in regards to the served area by given MDC) can be provided and accounted for in order to provide the soft guarantee on RTT.

3.3 Provided knowledge

Data provided to implement the predictor is in the form of a data matrix describing the behavior of a process when running alone or alongside a combination of other processes. In this matrix, columns describe various properties (such as execution time, number of instructions executed, number of written blocks on disk, etc.) and rows represent runs of the application (a run represents one calculation of a request). This data matrix allows us to examine the behavior over multiple runs, which can be used to infer statistical information and also reveals low probability behavior, such as the writing of disk buffers. An example is provided in Table 3.1.

run	elapsed	instructions	cache-misses	read_sectors	written_sectors	io_time
1	5506	42787936272	58142200	0	8976	170
2	5467	42788015318	58033451	0	7432	158
3	5496	42787914647	58113190	0	6608	94
4	5545	42787790092	58147487	0	4688	66
5	5470	42787883516	58149216	0	5680	114
6	5515	42787801186	58045890	0	6592	125

Table 3.1: An example of possible data matrix of a process.

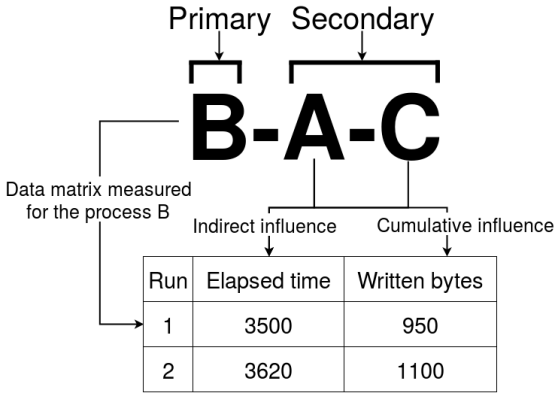

Figure 3.2: Ilustration of cumulative influence on a property.

3.3.1 Data matrix naming convention

Let us consider we had processes A, B and C at our disposal. The data matrix of a process allocated alone is called *single data matrix* and is denoted only by the ID of the process (e.g., data matrix A). The data matrix of a combination of processes is called *combination data matrix* and is denoted by the IDs of processes separated by a dash.

For example, B-A-C means a measurement of a situation, in which three processes were collocated on a given node. In the combination, we distinguish the primary process and secondary processes. The primary process is the first ID in the combination (B), the secondary processes are the rest (A and C). The primary process is the process, for which we measure the data matrix and secondary processes are influencing the primary process. Thus the result of measuring the B-A-C combination is a data matrix with the information about process B (execution time, written sectors, etc.), which is influenced by the secondary processes A and C. For example, the execution time could be a second longer when compared to the measurement of single B.

It is important to note that some of the properties (e.g., number of written bytes on disk) can be cumulatively influenced by the secondary processes. Imagine that process A writes 1000 bytes per 3.5 seconds on disk. Thus even thought process B does not write anything on disk, the B-A-C data matrix will have the property influenced. This is unfortunately how the data is provided to us, as apparently, this behavior is unavoidable for some properties. This concept is illustrated in Figure 3.2.

3.3.2 Lack of node or process-specific information

At first glance, it seems like we do not get any information about the nodes on which the processes run, however, that is not entirely true. The data matrix of a given process is measured on a particular type of node (hardware and software configuration) and is only used for prediction on the same type of node. This way, the properties of the node are reflected on the data matrices about the process (e.g., in the length of execution time).

We are also not provided with any specific data about the process, such as if the process uses a particular library, uses sequence write on disk, etc. The reason is that such information provided by the developer is unreliable.

The lack of more specific information about the node or the processes is on purpose. The predictor needs to function on multiple types of nodes and it is undesirable to structure the predictor around the knowledge of, for example, particular pieces of hardware.

It is important to note that the data matrix of a process can vary from node to node. The reason is, that some hardware and OS configurations do not allow the measurement of certain properties (e.g., the information about CPU cache), or provide the measurement of additional properties.

The lack of specific information about the nodes and processes and the varying data matrix of a process on different nodes can enhance the difficulty of designing the predictor greatly. However, when the predictor is constructed with these limitations, it becomes much more flexible and also future-proof. If, for example, new measurable properties became available on a new server architecture running a new type of application, the predictor should be automatically able to take advantage of such a data matrix and provide prediction, even if this situation is entirely unaccounted for.

3.3.3 How is the data matrix measured

Measured processes are run in containers with a 100% request rate (their calculation is repeated after each completion), which simulates how would these processes run in the edge-cloud node. For each run of the primary process, properties are measured into a row containing various information about the use of CPU, memory, disk, etc. One difference is that after a certain number of runs, the computer is restarted in order to minimize the effects of different computer initializations (e.g., loading of libraries).

The measurements of a data matrix last for several hours (in our case between one and two hours). The length of measurement must be sufficiently long in order to acquire a statistically significant number of runs. We expect that dependencies are run alone on separate computers in the same MDC during the measurement in order to guarantee acceptable slowdown.

Number of measurements

Due to the number of possible combinations of processes and length of the measurement, it is expected that only a small percentage of all combinations will be measured for a given edge-cloud system. The number of data matrices

in a given arity of processes (combination of five processe has an arity of five) is calculated as:

$$DM_a = \binom{(n+a-2)!}{a-1} \times n \tag{3.1}$$

Where DM_a is the number of data matrices in a given arity a and n is the number of available processes. The number is calculated as the combination with repetition of secondary processes, times the number of possible primary processes. This is due to the fact, that for each combination of secondary processes we can choose one primary process.

Imagine we had 17 different processes and we could collocate up to 5 processes. The number of quintuples alone already amounts to $(\binom{20}{4} \times 17) = 82365$. If a single measurement lasts around 1 hour (which is our case), we would need at least 82 365 hours (3 431 days) of machine time to measure them all.

This puts another limitation on the design of the predictor, as we can expect only a small percentage of all combinations to be measured.

3.4 Overview of the whole process

The process of providing a soft real-time guarantee (as illustrated in Figure 3.3) starts with a developer submitting their process to the cloud provider for evaluation. The cloud provider then measures necessary data matrices and provides them to the predictor. After that, the developer can request the deployment of their process to the edge-cloud. The cloud provider then uses the deployment solver to find a suitable deployment of all processes on the MDC. The deployment solver repeatedly uses the predictor to estimate the percentile of the execution time of collocated processes in order to find a deployment, which satisfies the soft real-time requirements of developers. Finally, the cloud provider deploys the processes according to the suitable deployment plan.

It is important to note that the predictor is expected to be periodically updated with new data matrices in order to improve the precision of predictions. The cloud provider should also monitor the real performance of deployed processes in order to confirm the predicted execution times or request a redeployment.

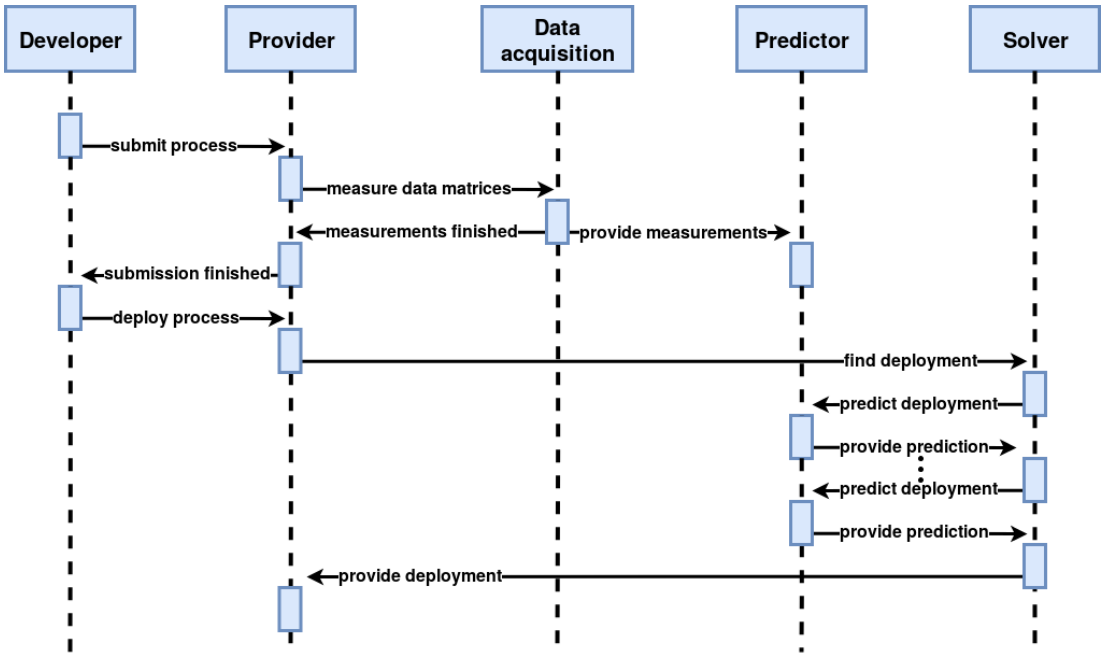

Figure 3.3: Overview of providing soft real-time guaranties on process execution time.

4. Predicting a percentile of process execution time

In this chapter, we first provide an overview of the predictor's architecture. Then we focus on related theory needed to understand the design of the predictor, such as similarity, cluster analysis, weights and regression analysis. After that, we describe the individual prediction methods. The implementation details are explained in Chapter 5.

4.1 Predictor architecture overview

As described in Section 3.1, we need a predictor, which predicts a given percentile of the elapsed execution time of a process when this process is collocated alongside other processes.

As mentioned in Section 3.3, the predictor needs to be general enough to work on multiple hardware and software configurations of nodes. The predictor can only utilize data matrices, which describe the behavior of a process or a combination of processes on a given type of a node. Consequently, the predictor does not use any specific information about the nodes (such as the type of CPU used) or the processes (such as which library does the process use). The form of data matrices is heavily influenced by the configuration of a node – two data matrices of the same process measured on two different nodes can have very different measured properties due to different hardware.

A trivial prediction solution is to run the combination of processes on a test-node and measure the slowdown for each process, thus creating a database of combinations with known slowdowns. Then, before assigning the combination to an edge-cloud node, a simple database lookup would suffice to decide, if the slowdown is acceptable or not. As explained in Section 3.3.3, the measurement of a large number of combinations is challenging, which makes this solution acceptable only with a small number of different processes and different node types.

In Section 3.3, we also mentioned the advantage of having a general predictor. The main advantage is that this predictor should be able to work on new and unexpected hardware configurations of nodes, predicting unexpected processes with new and unexpected properties in the data matrix. This sounds very similar to the use cases for the latest deep learning models. The reason why we did not use deep learning for our predictor lies in the amount of information available to us. It is usually necessary to utilize data points in the numbers of millions, however, as we will see in Chapter 6, the number of combinations we were able to obtain is in the thousands from less than twenty processes. Thus we consider the risk of unsuccessful deep learning due to limited data to be too high.

As a result, we decided to use a more traditional approach to design the predictor. The core of our approach consists of cluster analysis, weights training, regression analysis, and boundary detection. The prediction process is visualized in Figure 4.1.

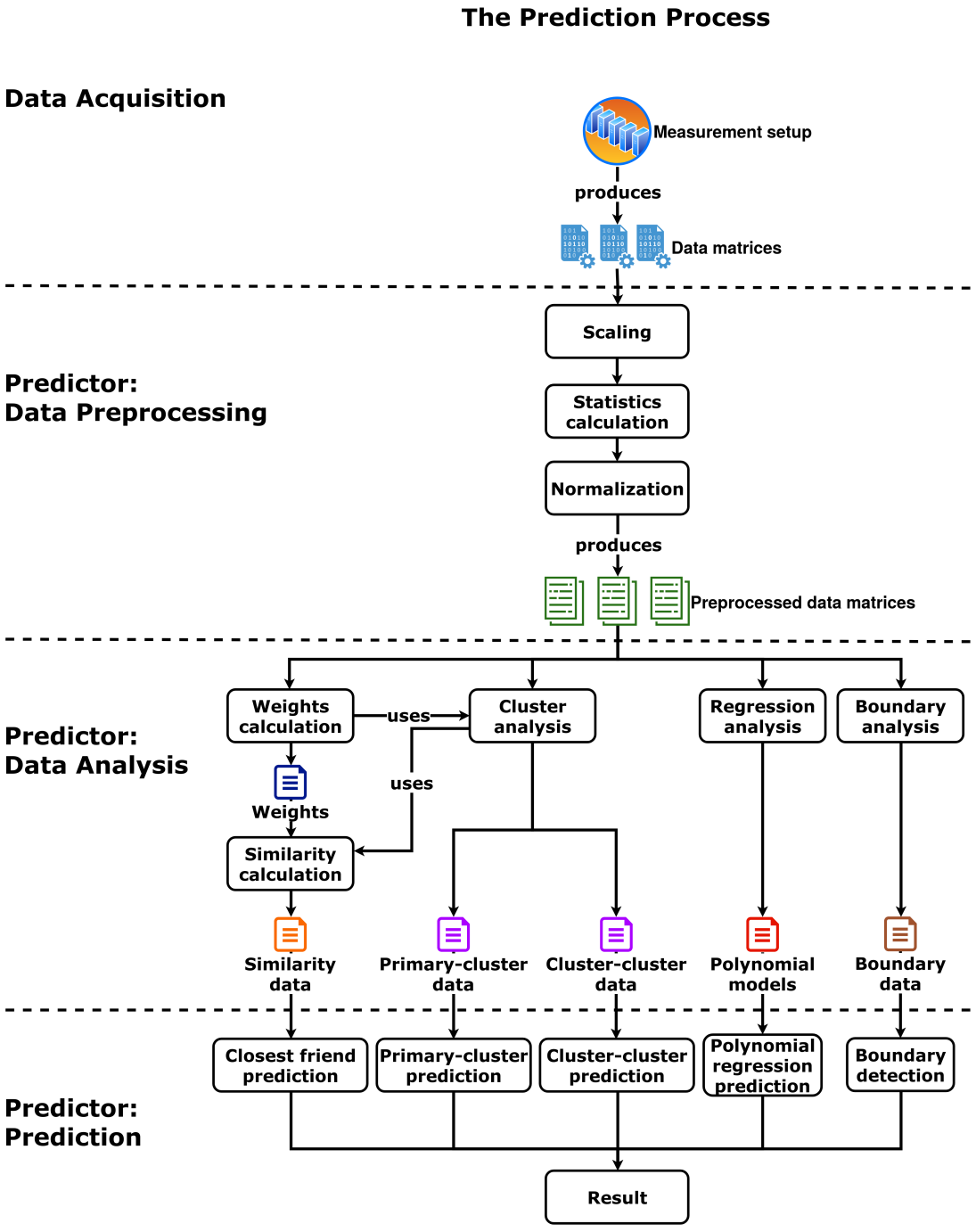

Figure 4.1: Schema of the prediction process.

Data acquisition

In the first phase, it is necessary to obtain data matrices, which will be provided to the predictor. We utilized an existing measurement software that ran on a small server farm. This step is executed a long time before any actual prediction is calculated due to the time required for measurement.

Predictor: Data preprocessing

After we finish the data acquisition, we need to preprocess the data matrices into a more suitable form. The result of preprocessing is scaled and normalized statistical information about the measured properties of processes, such as mean, median, deviation, etc. Further explanation is provided in Section 5.1. This step is done in advance in order to reduce the prediction calculation time.

Predictor: Data analysis

The preprocessed data matrices are analysed in order to uncover:

- Similarity between processes (Section 4.2)
- Relationships among the processes using cluster analysis (Section 4.3)
- Usefulnes of various measured properties using weights (Section 4.4)
- Trends in slowdown of processes using regression analysis (Section 4.5)
- Limits on node resource usage using a boundary system (Section 4.7).

The approaches to data analysis are evaluated in Chapter 6. This step is also done in advance to reduce the prediction calculation time.

Predictor: Prediction

Finally, we calculate a multitude of predictions for a given combination of processes by using the results of all previous phases. This step is designed to be computationally simple in order to achieve fast calculation times and is described in Section 4.6. The individual prediction methods are evaluated in Chapter 6. The user of the predictor is provided with a prediction from the most accurate method that was able to predict a given combination with the provided data.

4.2 Similarity between processes

It is important to define the concept of similarity between processes first, as this concept is used multiple times in our solution. The whole meaning of similarity depends heavily on the description of processes and the way this information is interpreted. For example, we could say that two processes are similar if they have similar lengths of execution time. This is, of course, not suitable for our purposes.

We are interested in the similarity of behavior and resource usage of the processes. Imagine we had two processes, which write some data on the disk and a third process, which only calculates some mathematical functions. In our sense of similarity, the two processes using disk are more similar to each other than to the third process.

In our case, the description of processes is in the form of single data matrices. Combination data matrices are not used as we only want to know the information about processes running alone without the interference of other processes.

It is possible to devise many ways of interpreting the information in single data matrices in order to identify the similarity of behavior of processes. We chose a few approaches, which we describe in Section 5.2 and compare in Chapter 6. For now, the key thing to understand is that similarity between processes means similarity in their behavior and resource usage.

4.3 Cluster analysis

Cluster analysis is one of the key concepts of our solution. The general idea of clustering is to find relationships between objects by placing similar objects in groups. The objects in said groups should be more similar to each other than to objects in other groups. In our case, the objects are individual processes, which can run in the edge-cloud system.

To provide a motivational example, imagine we had a group of processes and data matrices of all pairs of processes (all double data matrices). If we ran a cluster analysis, the resulting clusters could, for example, correlate with the process resource usage, dividing them into CPU-heavy, memory-heavy and disk-heavy clusters. We hypothesize that similarly behaving processes should influence other processes in a similar manner. Thus the influence between members of two clusters should be similar (e.g., disk-heavy processes slow other disk-heavy processes down by a certain percentage of execution time but are neutral to CPU-heavy ones). Now imagine that we were provided with a new process with only a single data matrix measured. Just by placing this process into its respective cluster, we can predict the interaction between this process and others. As a result, the knowledge of groups of similar processes offers additional information, which can be used to uncover new relationships between processes and predict their interaction.

It is important to note that the clusters could also group processes by other means than just resource usage. Imagine we were provided with six CPU-heavy facial recognition processes and the clustering algorithm divided them into two clusters – three versions of one process and three versions of some other process. Thus even though the resource usage is very similar, the versions of the processes are much more similar to each other, which divides the set of processes into two clusters.

As a result of cluster analysis, each process is assigned a cluster (denoted by a number), into which it belongs. The utilization of information obtained by cluster analysis is explored in greater detail in Section 4.6.2 and 4.6.3.

Cluster analysis is not a single algorithm but a problem to be solved, for which there are many methods available. The choice of a particular method influences the result of cluster analysis heavily. The explanation of the methods we used is provided in Section 5.3.

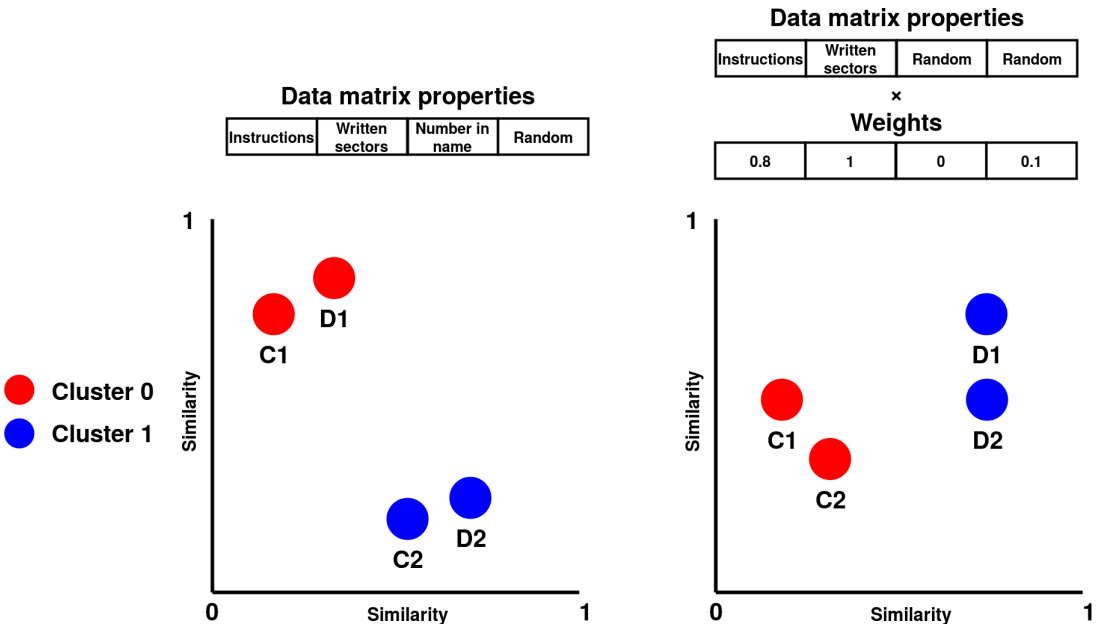

Figure 4.2: Illustration of the improvement of similarity calculation and cluster analysis caused by the introduction of weights.

4.4 Weights

The concept of weights relates directly to the concepts of similarity between processes and cluster analysis. The data matrix, which is provided as the description of the processes, only provides information about properties of the process (e.g., the number of instructions executed etc.). However, it would be useful to have information indicating which of these properties are actually useful in order to determine the similarity of behavior and to analyze clusters.

To illustrate this issue on an example, suppose that an error occurred during the measurement of data matrices, causing some properties to have random values. We measured CPU-heavy processes C1, C2 and disk-heavy processes D1, D2. Without the ability to determine which properties are useful and which are useless, these erroneous properties would negatively impact the calculation of similarity between processes and consequently negatively impact the cluster analysis.

As a result, weights indicating the usefulness of various properties for similarity calculation are introduced. Weights are in the form of a vector, where each value corresponds to one property in the data matrix. The values in this vector range between 0 and 1 and indicate the importance of a given property for the similarity calculation, as illustrated in Figure 4.2.

We hypothesize that with these weights, we can improve the similarity calculation, reduce noise in the data matrices, and consequently improve the cluster analysis. They also solve one of the challenges we are faced with, namely the issue of having different data matrices on different types of nodes. As we can not be sure that the measured properties of processes are correct or useful, we need a system of determining the importance of these properties.

It is important to highlight that weights can differ on different nodes (due to possibly different data matrices between the nodes) in their length and value.

Thus weights are node-specific and their value depends on available processes.

Similarly to cluster analysis, there is no single algorithm used to find such weights, but many approaches can be valid. The implementation of the approach we chose is described in Section 5.4.

4.5 Regression analysis

Regression analysis is another key concept of our solution and is used to find trends in the slowdown of processes using only the information about the length of execution time in single and combination data matrices.

To provide a motivational example, imagine we only had a single process A in the edge-cloud system. We then measured A running alone (90th percentile of the execution time equal to 10 sec), A-A (15 sec) and A-A-A-A (25 sec). Using this information, we can express a prediction of the percentile of the execution time of processes A as:

$$A_{percentile} = t_0 + t_1 secondary \tag{4.1}$$

This form of prediction is a case of linear regression prediction, where the percentile of the execution time of process A ($A_{percentile}$) is the *dependent variable* and the number of secondary processes A (*secondary*) is the *independet variable*. In the model (equation), the dependent variable is a linear combination of the coefficients t_0 and t_1. The most logical estimation of coefficient t_0 is the percentile of execution time of the process A running alone (10 sec) and 5 sec for t_1. Using this formula, the prediction of percentile of execution time for process A in the combination A-A-A is 20 sec.

This concept can be extended for multiple processes into *multiple linear regression* with multiple independent variables. For example if we had processes A and B, the regression models would be:

$$A_{percentile} = x_0 + x_1 A + x_2 B \tag{4.2}$$

$$B_{percentile} = y_0 + y_1 A + y_2 B \tag{4.3}$$

Where A and B are the numbers of secondary processes A and B respectively and x_0, x_1, x_2, y_0, y_1 and y_2 are the coefficients, which need to be calculated.

4.5.1 Polynomial regression analysis

Polynomial regression analysis is a special case of multiple linear regression, where the model is created as the nth degree polynomial of the independent variables. We can consider the previous examples to be a first-degree polynomial regression. Thus the second-degree polynomial model of the example with single process A is:

$$A_{percentile} = t_0 + t_1 secondary + t_2 secondary^2 \tag{4.4}$$

As a result, instead of approximating the percentile of execution time of a process using a line, we can now use a curve as illustrated in Figure 4.3. Similarly,

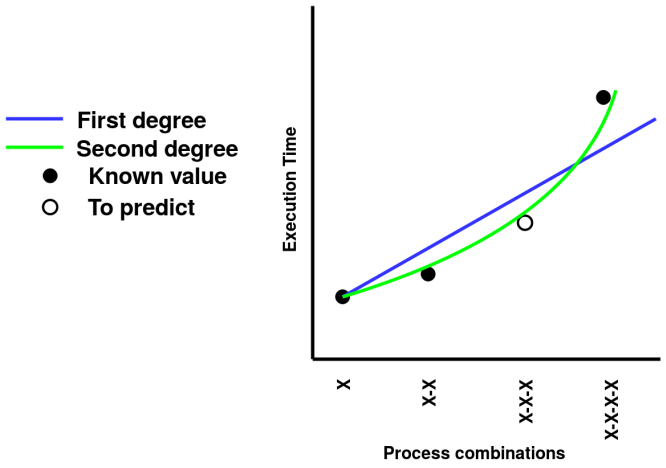

Figure 4.3: Ilustration of of first and second degree polynomial regression models.

the second-degree polynomial model of the example with processes A and B for $A_{percentile}$ is:

$$A_{percentile} = x_0 + x_1 A + x_2 B + x_3 A^2 + x_4 B^2 + x_5 (A \times B) \tag{4.5}$$

For $B_{percentile}$, the model is similar, just with different coefficients. In our solution, we use the polynomial regression to predict the percentile of the execution time of processes. The challenge is how to estimate the values of the coefficients (x_0, x_1, etc.) using the information about the execution time in combination data matrices, which is explained in Section 5.6. The degree of the polynomial is decided experimentally in Section 6.5.

4.6 Prediction methods

We chose to implement a multitude of prediction methods. We hypothesize that individual methods can perform better or worse in certain conditions and thus it is suitable to implement and compare more of them in order to find the one which suits us best. We also hypothesize that various prediction methods can complement each other – for example, when the best performing method does not have suitable data for given prediction, the second-best performing method can step in and provide a prediction.

4.6.1 Closest friend prediction

Hypothesis

Closest friend prediction is the simplest method, which only utilizes the information about the similarity of processes. Imagine we want to predict the 90th percentile of the A-B-C combination. We lack the data matrix for this combination, however, we already measured A-B1-C or A-B-C1, where B1 and C1 are previous versions of their respective processes. Intuitively, the data matrices of old versions could serve as suitable predictions for the A-B-C combination, as their behavior should be quite similar.

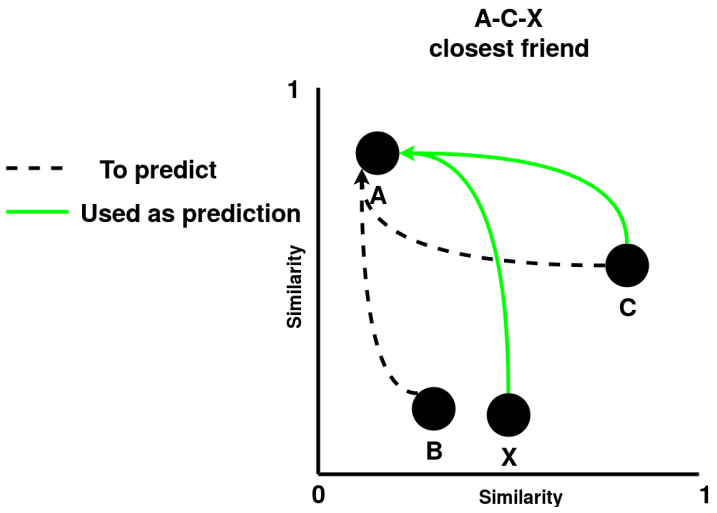

Figure 4.4: Illustration of the closest friend prediction method.

If we extend this idea, we hypothesize that any combination can be predicted by a data matrix of some other combination, if the secondary processes in this alternative combination are the most similar to the original ones. This concept is illustrated in Figure 4.4.

Procedure

During the prediction phase:

1. Prepare alternative combinations by exchanging processes in the provided combination with their most similar counterpart. For example, alternative combinations for A-B-C (if B is most similar to X and C is most similar to Y) are A-B-C, A-C-X, A-B-Y and A-X-Y.
2. Sort the alternative combinations by their distance from the original combination. The distance of an alternative combination is measured as a sum of distances between the original and alternative processes in the combination.
3. Iterate through the sorted alternative combinations and perform a database lookup for the data matrix. If found, return the requested percentile of elapsed execution time.

4.6.2 Primary-cluster prediction

Hypothesis

Imagine that due to the lack of data matrices, the closest friend prediction could not be calculated. We thus need to access more information, from which to calculate the prediction, by going upwards in abstraction – instead of examining how a combination of processes influence a process, we examine how a combination of clusters on average influences a process. As a result, the primary-cluster prediction method should be less accurate than the closest-friend prediction, however, due to the larger amount of available data, it can complete a prediction more often.

We hypothesize that the influence of processes from secondary clusters on the primary process can be analyzed as the average percentual change of percentile of

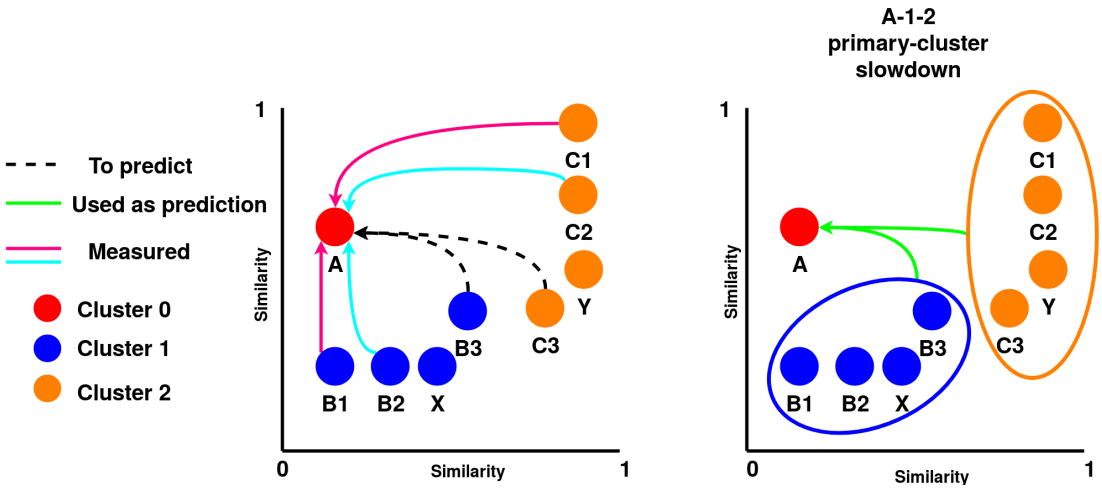

Figure 4.5: Illustration of the primary-cluster prediction method.

the execution time of the primary process. We call it the primary-cluster slowdown and denote it similarly to the data matrix naming convention, except secondary processes are replaced by cluster numbers.

To give a simple example (illustrated in Figure 4.5), consider that processes A belongs to cluster 0, processes B1, B2, B3, and X belong to cluster 1 and processes C1, C2, C3 and Y belong to cluster 2. We have data matrices A-B1-C1, A-B2-C2 and we want to predict A-B3-C3. B3 is most similar to X and C3 to Y, which blocks the closest friend prediction. When using the primary-cluster prediction, we calculate that percentile of A is 10% slower in A-B1-C1 and percentile of A is 20% slower in A-B2-C2. Then the A-1-2 primary-cluster slowdown is 15%. The prediction of A-B3-C3 is then calculated as the percentile of the execution time of single A extended by primary-cluster slowdown (15%).

Procedure

1. During the analysis phase:
   (a) Analyse clusters of processes and assign each process a cluster into which it belongs.
   (b) Calculate primary-cluster slowdown for all available combinations of primary processes and clusters.

2. During the prediction phase:
   (a) Identify into which clusters do the secondary processes in the requested combination belong. In other words, transform the combination from A-B-C into A-0-1, for example.
   (b) Perform a database lookup into the previously calculated primary-cluster slowdowns. If the requested slowdown was calculated, return the percentile of execution time of the primary process running alone extended by the primary-cluster slowdown.

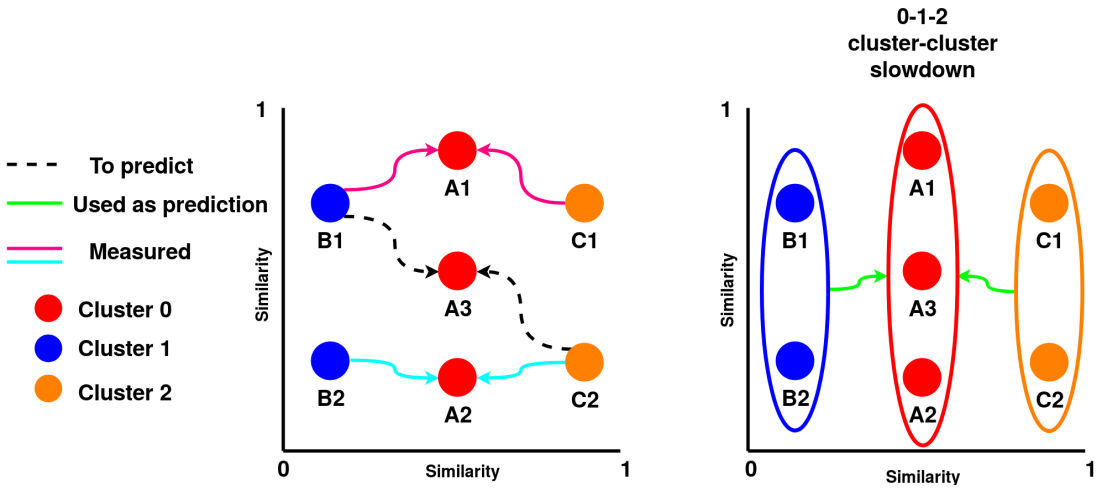

Figure 4.6: Illustration of the cluster-cluster prediction method.

4.6.3 Cluster-cluster prediction

Hypothesis

The cluster-cluster prediction method is a direct extension of the thought process behind the primary-cluster prediction. If we lack the data even for primary-cluster prediction, we can again go a step upwards in the abstraction and focus on the interaction between the cluster, into which the primary process belongs, and the clusters, into which the secondary processes belong. The cluster-cluster prediction method should be less accurate than the primary-cluster prediction, however, due to the larger amount of available data, it can complete a prediction more often.

We hypothesize that this interaction between primary and secondary clusters can be analyzed as an average percentual change of percentile of the execution time of processes from the primary cluster when collocated with processes from the secondary clusters. We call it cluster-cluster slowdown and denote it similarly to the primary-cluster notation, except the primary process is also replaced by cluster number. The prediction is then calculated by adding the cluster-cluster slowdown to the original percentile of the execution time of the primary process in the predicted combination.

To give a simple example (illustrated in Figure 4.6), consider that processes A1, A2 and A3 belong to cluster 0, processes B1 and B2 belong to cluster 1 and processes C1 and C2 belong to cluster 2. We have data matrices A1-B1-C1, A2-B2-C2 and we want to predict A3-B1-C2. When using the cluster-cluster prediction, we calculate that the percentile of A1 is 10% slower in A1-B1-C1 and the percentile of A2 is 20% slower in A2-B2-C2. Then the 0-1-2 cluster-cluster slowdown is 15%. The prediction of A3-B1-C2 is then calculated as the percentile of the execution time of single A3 extended by cluster-cluster slowdown (15%).

Procedure

1. During the analysis phase:
   (a) Analyse clusters of processes and assign each process a cluster into which it belongs.

(b) Calculate cluster-cluster slowdown for all available combinations of primary and secondary clusters.

2. During the prediction phase:
   (a) Identify into which clusters do the processes in the requested combination belong. In other words, transform the combination from A-B-C into 2-0-1, for example.
   (b) Perform a database lookup into the previously calculated cluster-cluster slowdowns. If the requested slowdown was calculated, return the percentile of execution time of the primary process running alone extended by the cluster-cluster slowdown.

4.6.4 Polynomial regression prediction

To calculate the polynomial regression prediction, we build a regression model for each one of the processes during the analysis phase. Each model is a polynomial of a certain degree, which consists of independent variables and coefficients. The coefficients are calculated during the analysis, therefore, to calculate the prediction of the percentile of the execution time of a process, we only need to fill the independent variables.

During the prediction phase, the values of independent variables are calculated from the numbers of distinct secondary processes in the predicted combination of processes, as explained in Section 4.5.

Procedure

1. During the analysis phase:
   (a) For each process:
      i. Gather all data matrices with the specified primary process
      ii. Calculate the regression model
2. During the prediction phase:
   (a) Calculate the values of the independet variables from the predicted combination
   (b) To form the prediction, apply the model using the independent variables

4.7 Operational boundary

When deploying processes on the nodes, we want to collocate as many processes as possible, as long as the RTT satisfies soft real-time requirements. This is done in order to reduce the number of nodes needed. However, if a combination of processes exceeds the computational capacity of a computer, their behavior can become highly unpredictable. There are many ways in which a set of processes can exceed the capacity of a computer: they can overload the CPU, require too much memory, use too much network bandwidth, etc.

To give an example, imagine we collocated five processes, which write 50 MB per second on disk, while the capacity of the computer is to write 200 MB per

second on the disk. We would exceed the capacity by 50 MB per second, which would result in a slowdown in the execution time of the processes.

As a result, a problem arises – how can we be confident that the predictor is reliable when predicting a combination. It is reasonable to assume that a predictor probably can not predict a combination that exceeds the computational capacity of a computer by 500% and that such a combination probably exceeds the RTT requirements anyway. There are, of course, exceptions, such as that we measured in advance exactly this extreme combination, and now the predictor can predict it with 100% accuracy – this situation, however, should be very rare.

As a result, we need to decide if a given combination of processes can be reliably predicted. There are two solutions to this problem:

1. Dynamic solution: Design a special estimator, which would for any combination of processes evaluate the capability of the predictor to predict this combination and estimate confidence in this prediction.
2. Static solution: Detect that a combination exceeds an operational boundary (e.g., 200% of computer resources) and mark the prediction as potentially unreliable.

We chose to implement the static solution and leave the dynamic solution as future work. By setting the operational boundary, the user is given greater control over the predictor and can avoid many problematic situations. The design of the boundary detection system is explained in Section 5.7. In Chapter 6, we examine the accuracy of the predictor, both with and without the use of boundary detection, in order to evaluate its effects.

5. Implementation

In this chapter, we focus in greater detail on the implementation of certain aspects of the solution, as described in Chapter 4. During the implementation, we focused on finding more than one way to solve a problem (e.g., more than one normalization function, similarity function, clustering algorithm, etc.). The reason being, that we do not know in advance which combination of methods is most suitable for us, and as a result, we want to test out multiple combinations experimentally in Chapter 6.

5.1 Data preprocessing

The purpose of the preprocessing is to extract useful statistical information from the provided data matrices for similarity search between applications and to determine the percentile of the slowdown of a primary process in a combination of processes. The results of preprocessing are stored in order to reduce the calculation time of prediction.

It is important to note that preprocessing differs for single data matrices and combination data matrices. This is due to the fact that some properties can be cumulatively influenced by secondary processes and thus do not offer useful information. Unfortunately, the only reliably usable information in the combination data matrix is the length of run time. Thus only single data matrices are scaled and normalized.

5.1.1 Scaling

Initially, each row in the data matrix provides exact values of various properties of a process (e.g., number of instructions in a calculation). It is necessary to scale some of these values by their respective run time in a given row in order to distinguish resource consumption of long-running processes from short-running processes. As a result, a portion of the properties in a row will be expressed per millisecond (e.g., instructions per millisecond), which is used later for similarity search between processes. The user can configure which properties should be scaled.

To give an example, without scaling a process M1, which calculates matrix addition, and a second process M2, which calculates the same matrix addition two times, would seem like processes with very different behavior without property scaling (M2 would have the values of scaling properties double). However, with scaling, we can see that those two processes behave essentially the same, just the second process runs longer.

As explained in Section 3.2, each process calculation on the node will be immediately repeated after completion. If we imagine two scenarios, M1-M1 and M1-M2, the influence of the secondary process on the primary process should be essentially the same in both cases (just two parallel matrix additions collocated). Scaling is thus necessary in order to calculate the similarity of behavior between processes.

5.1.2 Statistics

With the properties scaled, we can extract statistical information from the data matrix. For the single data matrices, the statistics are calculated for each property and consist of:

- Mean
- Median
- Standard deviation
- Percentile
- Max
- Min

The result is again a matrix as each statistic is calculated for every property.

For the combination data matrices we only analyze the execution time property and calculate the relative slowdown – the difference between the percentile of the execution time of primary process running alone and in combination, measured as the percentage of the primary percentile execution time:

$$\frac{percentile(combination\_time) - percentile(primary\_time)}{percentile(primary\_time)} \tag{5.1}$$

5.1.3 Normalization

The final step in the preprocessing phase is to normalize the data, which transforms the values of a property from its original range into values between 0 and 1. This is needed as the values of different properties can be in very different ranges (some properties are in single digits, other in millions). Without normalization, this would distort the similarity search between processes greatly, as the properties with large ranges would have a bigger impact. Different normalization methods can have different influence on the overall prediction method. We chose to implement the two most popular methods and compare them in Chapter 6.

Thus the result of data preprocessing is a set of preprocessed data matrices (one for each process), which consists of normalized statistical information (illustrated in Table 5.1).

	instructions	**cache-misses**	**read_sectory**	**written_sectors**	**io_time**
mean	0.8	0.47	0	1	1
median	0.75	0.48	0	1	1
deviation	0.02	0.01	0	0	0.02
...	...	...	...	...	...

Table 5.1: An example of a preprocessed single data matrix.

Min-max normalization

For every calculated statistic and every property we find the *max* and *min* value among all processes. The normalized value of the property in a particular statistic

of a given process X is then calculated as:

$$normalized\_value_{s,p}(X) = \frac{original\_value_{s,p}(X) - min_{s,p}}{max_{s,p} - min_{s,p}} \tag{5.2}$$

Where s represents a statistic and p represents a measured property.

Z-score normalization

For every calculated statistic and every property we find mean μ and standard deviation δ among all processes. The normalized value of the property in a particular statistic of a given process X is then calculated as:

$$normalized\_value_{s,p}(X) = \frac{original\_value_{s,p}(X) - \mu_{s,p}}{\delta_{s,p}} \tag{5.3}$$

Where s represents a statistic and p represents a measured property.

5.2 Calculating similarity between processes

As explained in Section 4.2, the meaning of similarity between two processes depends on the description of said processes and on the interpretation of the provided description. In our case, we form the description of processes from preprocessed single data matrices. Combination data matrices are not used as we only want to know the information about processes running alone without the interference of other processes.

We implemented two distance measures, which is just an inverse of similarity (clustering algorithms rely on distance instead of similarity), and both of these measures use only a subset of the vectors available in preprocessed single data matrices to describe a process. This is done in order to reduce the calculation time.

Application of weights

During similarity calculation, we can also utilize weights, which indicate the importance of various properties. The application of weights is straightforward – we execute elementwise multiplication between each vector used to describe a process and the weights. Because the values in weights range between 0 and 1, this will cause some properties to contribute less or more towards the final similarity value.

5.2.1 Average pair correlation distance

To calculate the average pair correlation distance, we describe the process using a set of vectors – the max, min and percentile vector from the preprocessed single data matrix. The correlation distance is then calculated between every pair of vectors from the two sets and the results are averaged.

$$corr\_dist(u, v) = \frac{(u - \bar{u}) \cdot (v - \bar{v})}{||u - \bar{u}||_2 \times ||v - \bar{v}||_2} \tag{5.4}$$

Where $||u||_2$ is the euclidean norm of the vector, $\bar{u}$ is the mean of elements of the vector and $u \cdot v$ is the dot product of the vectors. The final average pair correlation distance is then calculated as:

$$APCorrelation(U, V) = \frac{\sum_{u \in U} \sum_{v \in V} corr\_dist(u, v)}{|U| \times |V|} \tag{5.5}$$

Where U and V are the two sets of vectors describing the two processes, between which we calculate the average pair correlation distance.

5.2.2 Frobenius distance

To calculate the Frobenius distance, we describe the process by a matrix consisting of the mean, median and standard deviation vectors taken from the preprocessed single data matrix. The Frobenius norm is then calculated from the difference of the matrices of the two processes, for which we calculate the Frobenius distance.

$$Frob\_dist(X, Y) = ||X - Y||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (X_{i,j} - Y_{i,j})^2} \tag{5.6}$$

Where X and Y are the matrices representing the processes.

5.3 Calculating clusters

In Section 4.3, we introduced the idea of cluster analysis, and in Sections 4.6.2 and 4.6.3 we explained how the information of clusters of processes can be used to predict the execution time of processes. In this section, we describe the chosen clustering algorithms

Requirements

We are interested in *hard clustering*, which means that the relationship between the object and the cluster is binary (the object either belongs or does not belong into the cluster). This is in contrast with *fuzzy clustering*, where objects belong to all clusters to a certain degree, which is not suitable for us.

Additionally, we require *strict partitioning clustering*, which means that each object belongs to exactly one cluster. Thus we do not allow outliers or overlapping of clusters. The reason is that overlapping and outliers would bring too much complexity to the prediction method. However, the exploration of these concepts could be suitable for future work.

The last requirement is the automatic detection of the number of clusters in the dataset. Consequently, we exclude algorithms such as k-means, where the user has to specify the number of clusters to detect. The reason being that the number of clusters is very hard to estimate in advance in our situation, and various sets of processes can produce vastly different numbers of clusters.

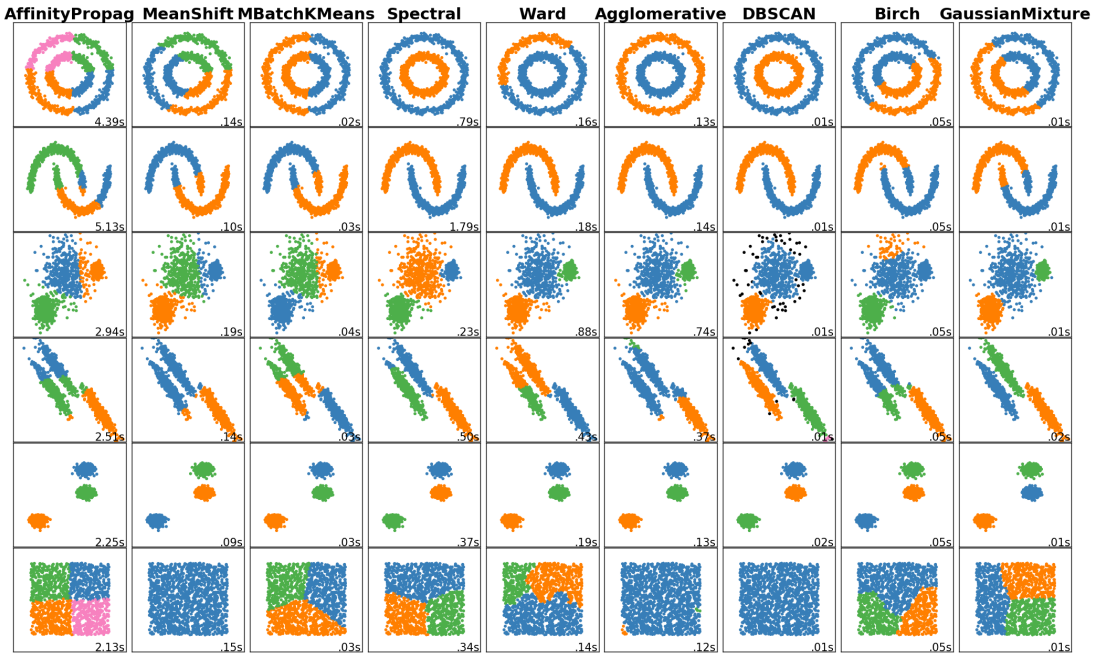

Figure 5.1: Comparison clustering algorithms provided by scikit using a toy dataset[14].

Selected algorithms

To be confident in the correctness of used clustering algorithms, we chose to use the implementation provided by Python scikit library[14].

The overview of all clustering algorithms provided by scikit using a toy dataset can be seen in Figure 5.1. In this overview, we can see that different clustering methods have very different outcomes. It is thus very important to choose a suitable algorithm for our purposes.

Imagine that the distances between data points in the toy dataset in Figure 5.1 are distances in the behavior of processes. We hypothesize that the clustering provided by Affinity propagation and mean-shift is most suitable if we want to cluster processes by the similarity of their behavior. The rest of the algorithms provided by scikit do not automatically detect the number of clusters, and the way they form clusters does not fit our needs.

For example, in the first row, we do not want to detect the outer and inner circle, but we want to split both circles into partitions. The same applies to the second row, where we do not want to detect two separate moons. Similarly, in the last row, we would like to split the square into four parts (in this case, the mean-shift does not cluster ideally).

To calculate the clusters, we provide the clustering algorithms with a distance matrix calculated using preprocessed data matrices and our similarity measures. A distance matrix is a matrix containing the information about the distance between each pair of processes.

5.3.1 Affinity propagation

Affinity propagation[15] is based on finding exemplars around which clusters are then formed. This is done through a technique called message passing, during which information is exchanged between data points until convergence. This information represents the ability of a data point to be the exemplar of other data points. The exemplars are iteratively chosen until they remain unchanged for some number of iterations, or a maximum number of iterations was reached.

The time complexity is $O(N^2T)$, where N is the number of processes and T is the number of iterations until convergence. The memory complexity is $O(N^2)$.

5.3.2 Mean-shift

Mean-shift[16] is a centroid based algorithm, which updates candidates to be the mean of the points in a region. We can understand this algorithm by thinking of our data points to be represented as a probability density function, where higher density regions will correspond to the regions with more points. In clustering, we need to find clusters of points, i.e., the regions with higher density in our probability density function. This is done by iteratively shifting the centroids towards higher density regions. The time complexity is also $O(N^2T)$.

5.4 Weights training

As introduced in Section 4.4, we need weights in order to determine the importance of properties measured in data matrices, in regards to the similarity of process behavior and cluster analysis.

To illustrate the idea behind the solution of training the weights, imagine we had processes C1, C2 and C3 as CPU-heavy processes and D1, D2 and D3 as disk heavy processes. We ran a cluster analysis and the result was three clusters [C1, D1], [C2, D2] and [C3, D3]. The reason being that in the data matrices one of the measured properties has random values. Thus the calculation of similarity of processes is negatively impacted, and consequently, also the clustering is negatively impacted.

We want to tell our training model, that the correct clustering is [C1, C2, C3] and [D1, D2, D3]. With this information, the training model can adjust the weights and try again, while hopefully lowering the importance of the erroneous property. In the end, we should end up with weights, which adjust the importance of properties in such a way, that the resulting clustering is correct.

In this scenario, the processes C[1,3] and D[1,3] are the training dataset. The correct clustering is the ground truth, which is assigned by a human. Thus we compare the results of cluster analysis on the training dataset against ground truth labeling. This way, we can determine if the weights improve the calculation of similarity of process behavior. Using the labeling in the ground truth dataset, we essentially determine, what does a similarity of process behavior mean to us, and the weights then shift the calculation of similarity in order to satisfy it. This method of weights training can be viewed as a case of supervised learning.

It is important to note that this approach is heavily dependent on the quality of the training data. If we provided ground truth, which was wrongly labeled, the weights would train to distort the similarity search. The ground truth, which we used in our solution, is described in Section 6.3.

Thus the process of training the weights is:

1. Create a training dataset from the ground truth.
2. For some number of iterations:
   (a) Use the latest weights and execute cluster analysis on the dataset.
   (b) Compare the result of cluster analysis to the ground truth labeling and assign it a score.
   (c) Based on the score, generate new weights with the goal of maximizing the score.

The score measures how well did the cluster analysis perform with the weights. The implementation of scoring is explained in Section 5.5. Thus the problem of weights training is transformed into an optimization problem with the goal of maximizing score through weights. Such an optimization problem has many common solutions, the algorithm which we implemented is described in Section 5.4.1.

5.4.1 Simulated annealing

Simmulated annealing is a method used to find global optimum of a function. In our situation, the function $f : R^n -> R$ is the cluster analysis combined with scoring method, which takes the weights of length n as argument and produces single number (the score). In order to explain how simulated annealing works, let us compare it to hill climbing. Let us define:

- $f : R^n -> R$ function for which we want to find global maxima
- $x, y \in R^n$ vectors
- $G : R^n -> R^n$ weights generation function

The algorithm for hill climbing (when searching for maxima) goes as follows:

1. $x = empty\_weights$
2. *For some number of iterations:*
   (a) $y = G(x)$
   (b) If $f(y) > f(x)$ then $x = y$

Note that the weights generation function produces new weights based on the previously best-performing ones. The disadvantage of hill climbing is that there is a chance of weights getting stuck in local maxima. In simulated annealing, we sometimes allow the acceptance of weights with a worse score, instead of always accepting weights with a better score. The hypothesis is that this will allow us to jump out of local maxima and find the global one, or at least get closer to it.

For this purpose, we define *acceptance criterion*, which determines if weights, which result in a worse score, will be used as the basis for next weights generation.

The acceptance criterion changes during the algorithms run, which is called *cooling schedule*. A cooling schedule is specified by the initial *temperature*, *decrementfunction* for lowering the value of temperature, and the final value of temperature at which the annealing stops[17]. With high temperature, the acceptance criterion accepts worse weights relatively frequently, but as time goes on and the temperature cools down, the acceptance criterion behaves more and more like hill climbing.

The idea is that we jump out of local maxima at the begining and towards the end we climb the global maxima (or value close to the global maxima). Let us define:

- $f : R^n -> R$ function for which we want to find global maxima
- $x, y \in R^n$ vectors
- $G : R^n -> R^n$ weights generation function
- $P : (R, R, R) -><0, 1>$ acceptance criterion
- T temperature
- C decrement function

The algorithm for simulated annealing goes as follows:

1. $x = empty\_weights$
2. While T>1:
   (a) $y = G(x)$
   (b) $x = y$ with the probability of $P(f(x), f(y), T))$
   (c) $T = C(T)$

When implementing this algorithm, we need to solve mainly three different issues, which are specific to the kind of function for which we want to find global optima: how to implement weights generation function, acceptance criterion, and decrement function.

The weights generation function tak is defined as:

- $G(weights) = weights + rand(-1, 1) * [\ldots, 0, dist, 0, \ldots]$

Where $rand(...)$ is a function which randomly chooses one of the provided numbers, $dist = 1/3$ and the position of $dist$ in the vector $[\ldots, 0, dist, 0, \ldots]$ is also random. The values of a particular weight can not get below 0 or exceed 1. Thus weights range between zero, low, medium and high importance.

The acceptance criterion is implemented as :

$$P(f(x), f(y), T) = \begin{cases} 1 & \text{if } f(x) =< f(y) \\ exp\left(\frac{f(y)-f(x)}{T}\right) & \text{if } f(x) > f(y) \end{cases} \tag{5.7}$$

The decrement function is implemented as $C = T * cooling$, where $cooling$ is a constant (usually 0.99) and starting temperature T is set to 100000. The temperature at which simulated annealing stops is $T = 1$.

Inspiration for these functions was drawn from[17].

5.5 Scoring clusters

As explained in Section 5.4, we need to assign a score to a clustering when compared to a ground truth dataset. The choice of a scoring function influences the training of weights greatly. It is thus important to try multiple functions and choose the best one experimentally.

To give an example, imagine we had a scoring function, that would return a score of 1, if the number of clusters matched between the calculated and ground truth clustering, and otherwise gave a score of 0. The trained weights would then try to push the clustering algorithm into detecting a certain amount of clusters while disregarding any requirements for the similarity of behavior of the processes. Thus the final clustering when using weights trained with such scoring would have the correct number of clusters, however, the contents of the clusters could be completely wrong.

As a result, we need to use such scoring functions, that can project the similarity of behavior into the score. As already mentioned, the ground truth dataset is labeled by the behavior of the processes. Thus the scoring function has to take this labeling into account in an intelligent manner.

However, such a scoring function can not be as trivial as counting the number of errors in labeling. The evaluation metric should not take the absolute values of the cluster labels into account but rather if this clustering defines separations of the data similar to the ground truth[14].

Selected scoring functions

To be confident in the correctness of used scoring functions, we again chose to use the implementation provided by Python scikit library[14].

We selected two scoring functions, namely Fowlkes-Mallows score and V-measure. Both functions range between 0 and 1, where 1 is perfect clustering and 0 completely wrong clustering.

5.5.1 Fowlkes-Mallows score

The Fowlkes-Mallows score is calculated as the geometric mean of the pairwise precision and recall[18].

$$FM = \frac{TP}{\sqrt{(TP + FP) \times (TP + FN)}} \tag{5.8}$$

Where TP is the number of pairs of processes, which belong to the same cluster in both detected and ground truth clusters (true positives). FP is the number of pairs of processes, which belong to the same cluster in the ground truth, but not into the same detected cluster (false positives). FN is the number of pairs of processes, which belong to the same detected cluster, but not into the same cluster in the ground truth (false negatives).

5.5.2 V-measure

V-measure is calculated as the harmonic mean of homogeneity and completeness of the clustering[19]

$$v = 2 \times \frac{h \times c}{h + c} \tag{5.9}$$

Homogeneity and completeness scores are given by:

$$h = 1 - \frac{H(C|K)}{H(C)} \tag{5.10}$$

$$c = 1 - \frac{H(K|C)}{H(K)} \tag{5.11}$$

Where C are the detected clusters, K the ground truth clusters, $H(C|K)$ is the conditional entropy of the ground truth and detected clusters, and $H(C)$ is the entropy of the detected clusters.

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \times \log\left(\frac{n_{c,k}}{n_k}\right) \tag{5.12}$$

$$H(C) = -\sum_{c=1}^{|C|} \frac{n_c}{n} \times \log\left(\frac{n_c}{n}\right) \tag{5.13}$$

Where n is the number of processes, n_c and n_k the number of processes in cluster c and ground truth cluster k, and $n_{c,k}$ number of processes from k assigned to c. $H(K|C)$ and $H(C)$ is defined symmetricaly.

5.6 Calculating polynomial regression models

As explained in section 4.5, we need to calculate the regression model for each process. More specifically, we need to calculate the coefficients of the polynomial of independent variables. The model for a process X is expressed in Equation 5.14.

$$X_{percentile} = x_0 + x_1 Y + x_2 Z + x_3 Y^2 + x_4 Z^2 + x_5 (Y \times Z) \dots \tag{5.14}$$

Where Y and Z are the number of some secondary processes Y and Z and $X_{percentile}$ is the percentile of execution time of process X. Our goal is to calculate the coefficients x_0, x_1, etc. The number of the coefficients depends on the number of distinct processes, which can be collocated with process X, and the degree of the polynomial. To calculate the coefficients, we need to first gather all data matrices with the primary process X. The value of the percentile of execution time in each data matrix can be expressed as an equation:

$$X_{percentile1} = x_0 + x_1 Y_1 + x_2 Z_1 + x_3 Y_1^2 + x_4 Z_1^2 + x_5 (Y_1 \times Z_1) \dots \tag{5.15}$$

Where the values of $X_{percentile1}$, Y_1 and Z_1 are known and the coefficients are unknown. Similarly, we would have equations for $X_{percentile2}$, $X_{percentile3}$, etc. depending on the number of measured combination data matrices. We can transform this set of equations into two vectors and a matrix.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad (5.16) \qquad A = \begin{bmatrix} Y_1 & Z_1 & Y_1^2 & \dots \\ Y_2 & Z_2 & Y_2^2 & \dots \\ & \dots\dots\dots & & \\ Y_m & Z_m & Y_m^2 & \dots \end{bmatrix} \quad (5.17)$$

$$b = \begin{bmatrix} X_{percentile1} & X_{percentile2} & \dots & X_{percentile(m)} \end{bmatrix} \quad (5.18)$$

Where the matrix A has dimensions $m \times n$. The vector b holds the values from the right side of the equations – percentiles of execution time of the process X when collocated with various combinations of processes. The matrix A holds the values from the right side of the equations (the polynomial values of the numbers of secondary processes) and the vector x holds values of the coefficients. Our goal is to find:

$$argmin_x ||Ax - b||_2 \quad (5.19)$$

Where $||_2$ is the euclidean norm. In other words, we want to find vector x, which after multiplying the matrix A, will produce a vector of execution times, which has the smallest distance from the original vector of execution times b.

To calculate the coefficients, we chose the method of non-negative least squares (NNLS). Standard least squares minimizes the sum of squares of the residuals (the difference between measured and calculated percentile of execution time) for each equation. The problem is that standard least squares can yield negative coefficients, and consequently, such coefficients could predict a negative slowdown in percentile of execution time. Because negative slowdown does not make sense in our case, we opted for NNLS, which is a constrained version of least squares, where negative coefficients are forbidden.

For the calculation of NNLS, we used the solution provided by sklearn[20]. After having the coefficients (the regression model) calculated for a given process, the prediction of a given combination of processes is formed simply by assigning the numbers of the secondary processes into the model.

The polynomial regression prediction can, in theory, predict any combination of processes. However, we consider the prediction to be unsucessful if the result is extremely high – in our case, higher than twenty times the percentile of the execution time of the process running alone.

5.7 Ensuring operational boundary

As explained in Section 4.7, in our solution, we want to detect the situation in which a combination of processes exceeds a user-specified computational capacity of a computer (e.g., 200% of the capacity).

In order to do so, we first define what is the maximal value of a property: *Maximal value of a measured property on a given computer is a value, which is impossible to exceed cumulatively on the computer with any number of running processes.* To use the example of a number of written bytes per second on disk, the

maximal value of this property could be 100 MB per second due to the hardware performance.

It is important to note that processes could have upper limits on the usage of various resources – for example, a process could be forbidden to allocate more than 2 GB of memory by the OS. The maximal value of a property is not the highest value, which a process can reach, but the highest value, which a computer can reach.

These maximal values then represent the 100% capacity of the computer in regards to the measured properties. We determine if a combination of processes exceeds a specified percentage of the capacity of the computer as follows:

1. For each property:
   (a) Sum the average value of that property from all processes in the combination
   (b) If the sum is lower than the specified percentage of maximal value of the property, this property does not exceed the boundary
2. The combination of processes does not exceed the boundary if each property lies below the boundary

As a result, in order to establish the operational boundaries, we need to approximate the maximal values of properties in the data matrix. It is important to note that these maximal values vary between types of nodes due to the difference in measured properties, as explained in Section 3.3.

5.7.1 Approximating maximal values of properties

There are multiple possible approaches to approximate the maximal values, and they can be split into two categories – approaches, which use the information about the hardware and software (configuration) of the computer, and those, which do not use this information. In our solution, we use the second category.

Let us first explain why we did not choose to primarily use the information about configuration of the computer. Imagine we knew details about the CPU, such as clock rate, architecture, etc. We could, in theory, estimate the maximal number of instructions, which a multi-threaded process can use in a unit of time. However, such an estimate would be unrealistic, as the performance of a computer depends on the combination of all hardware and software components. To estimate all properties, which indicate a resource usage, would require a very complex model and understanding of the computer. This would soon become unrealistic with large numbers of properties and unexpected components on new node types.

The approach we chose uses only information about the processes, which means data matrices with various measured properties. We hypothesize that we can approximate the maximal values of these properties through measurement of the performance of extreme processes (benchmarks) on the computer. As a backup option, the user can also supply the predictor directly with the information about maximal values of various properties to deal with limitations on resource usage by the OS and other problems with measurement. The primary way of approximating the maximal value of these properties should, however, be automatic.

The predictor comes with a set of benchmarks. When a user wants to predict on a new type of node, the predictor requests the measurements of provided benchmarks on this node. These benchmarks aim to use all of the capacity of various resources on the computer. As a result, we are provided with data matrices, which have various properties measured, hopefully, close to their maximal values.

Additionally, if a process supplied by the user exceeds the benchmarks in some property, the maximal value of this property will be estimated by this process instead of the benchmarks. Thus even if the benchmarks are not designed ideally, we can expect that the processes running in the edge-cloud system will eventually supply us with information about the maximal values of various properties. The probability that one process from the large numbers of processes running in the edge-cloud system will come close to the maximal value is expected to be reasonably high. On top of that, if the supplied benchmarks do not focus on some kind of property (e.g., usage of graphic cards), the user processes will, which makes this approach future-proof.

The description of the benchmarks which we used during the evaluation is provided in Section 6.3.

Excluding properties

Some properties can be excluding in the sense, that they exploit the same resource. For example, if we had separate properties for reads and writes on the disk, these two properties would exclude each other – two processes, one reading and the other writing 60 MB per second to/from disk would exceed the capacity of 100 MB per second disk.

To reflect this behaviour, it is needed to introduce new property into the data matrix aggregating excluding properties into one, which is the responsibility of the provider of the data.

User provided approximation

The usage of some properties can be restricted by the OS, such as the maximal number of allocated bytes by a process. As a result, the approximation of this property using benchmarks would become impossible. For example, a process could maximally allocate 4 GB of data, but the capacity of the computer is 32 GB.

As a result, it is permitted that the user can provide their approximation of maximal values of properties. The user can also specify which properties should be used to detect the boundary. For example, if one of the measured properties was non-limiting, the limit of such property can be set to infinite.

6. Evaluation

In this chapter, we first describe what the goal of the evaluation is. After that, we introduce the dataset used for evaluation, and at last, we describe and evaluate the experiments.

6.1 Goal of the evaluation

In Section 5, we introduced multiple normalization, similarity, clustering, scoring and optimization algorithms. The reason for such a range of algorithms is that each one can influence the cluster analysis of processes and calculation of weights, which consequently influences the prediction methods. Different combinations of algorithms thus need to be evaluated experimentally, as it is very hard to estimate, which combination of algorithms will result in the best prediction performance.

The first goal of the evaluation is to find out which combination of algorithms has the best performance on a supplied dataset of real applications. We also examine the best performing combination in depth both inside and outside of the operational boundary.

The second goal of the evaluation is to demonstrate that our system for weights training is able to improve the prediction performance if the supplied dataset gets corrupted by erroneous process properties.

It is important to note that each experiment and graph generation is automated and is intended to be a part of the predictor. The reason being that it is important to supply the user of the predictor with information about the prediction performance.

6.2 Measured dataset

First, we need to describe the dataset used for experiments and the configuration of the servers, which were used for measurements.

In total we have 17 processes at our disposal and measured:

- 17 single data matrices (100% of all singles)
- 289 double data matrices (100% of all doubles)
- 1500 triple data matrices (57.6% of all triples)
- 1500 quadruple data matrices (9.1% of all quadruples)
- 3000 quintuple data matrices (3.6% of all quintuples)

The equation for the total number of data matrices in a given arity is described in Section 3.3.3. The measured combinations are chosen randomly from all possible combinations in a given arity. Because the measurement of one data matrix lasts for one hour, this data set required 6306 hours (263 days) of machine time. Seven identical servers were used for the measurement of data matrices, thus the measurement took around 38 days.

Process description and measured properties

The measured processes are summarised in Table 6.1. As can be seen from the table, some of the processes are taken from existing projects (scalabench and stress-ng) and some are originally created (processes with high disk usage and various in-memory algorithms). The measured properties are summarised in Table 6.2.

ID	Source	Description	CPU	RAM	Disk
A	scalabench	Renders set of images using ray tracing	++++	++	
F	scalabench	In-memory transactions of banking application	++	++++	
H	scalabench	Optimization of ABC, SWC and SWF files	++	+++	
K	scalabench	Stanford topic modeling	+++	++	
O	scalabench	Emulates AVR microcontrolers programs	++	++	
SMATRIX	stress-ng	Transposition on a 4096x4096 matrix	+	++	
AVL	original	Inserts and deletes nodes on an AVL tree	+	++	
CYPHERD	original	Cyphers a random string and writes result	+	++	++
EGG	original	Egg dropping problem solver	+	+	
FACE	original	Human faces detection on images from disk	+	++	+
FLOYD	original	Floyd-Warshall's shortest path search	+	+	
JSOND	original	Randomly generates and writes json files	++	++++	+
PDFD	original	Randomly generates and writes pdf files	+	++++	+++
RB	original	Inserts and deletes nodes on a RB tree	+	++	
ROD	original	Rod cutting problem solver	++	+	
SORTD	original	Random number sequence sorted and written	+	+	+++
ZB	original	Extraction of many small files from Zip archive	+	++++	++++

Table 6.1: List of selected processes.

Process ID	Description
elapsed	Elapsed time in miliseconds
ref-cycles	Total number of reference cycles
instructions	Total number of instructions
cache-references	Total number of cache references
cache-misses	Total number of cache misses
branch-instructions	Branch instructions
branch-misses	Branch target address cache misses
PAPI_L1_DCM	L1 data cache misses
rw_completed	Total number of reads and writes completed
rw_merged	How many times are reads and writes merged for efficiency
rw_sectors	Total number of sectors read and written
io_in_progress	Number of I/O in progress
io_time	Milliseconds spent doing I/Os
weighted_io_time	Measure of I/O completion time

Table 6.2: List of measured properties.

Node description

The servers used for measurement consisted of 64-bit quad-core Intel Xeon E3-1230v6 @ 3.50GHz CPU, the disk was Seagate Constellation.2 ST9500620NS 500GB 7200 RPM 64MB Cache, SATA 6.0Gb/s 2.5 and we had 32GB of RAM available.

The computer ran Fedora Linux 28, Docker and OpenJDK with hyper-threading, turbo-boost and other power management features were disabled.

6.3 Universal Load Simulator

In order to generate ground truth processes for weights training and benchmark processes for boundary detection, we created the Universal Load Simulator (ULS). The main idea behind ULS is the ability to quickly generate single-threaded processes with various resource usage and behavior.

ULS achieves this by executing a sequence of Blocks, namely CpuBlock, MemBlock, DiskBlock or IdleBlock. Each of these Blocks is focused on the usage of a different computer resource. CpuBlock calculates various mathematical functions. MemoryBlock allocates and reallocates matrices, switches numbers in the matrix and uses recursion to change data on the stack. DiskBlock writes random data to a file on disk. IdleBlock executes sleep for a specified number of milliseconds. ULS is configured by a config file, where the total number of blocks, the ratio between various blocks, the way they are mixed, and the properties of Blocks are specified.

Imagine that we want to simulate a process, which half of the time focuses on CPU and half of the time focuses on disk, while running 6 seconds. We simply create a config file, where we specify that we want 1000 randomly mixed Blocks, half of them CpuBlock, half DiskBlock. The number of calculations in CpuBlock and the number of written bytes in DiskBlock can also be specified in order to achieve the desired execution time on a given computer.

We can see that the creation of a process with desired properties is greatly simplified. ULS is also ideal for the creation of ground truth, as we know exactly the properties of generated processes. The main challenge of ULS is the design of Blocks, as each block has to use only a single computer resource in a realistic manner.

6.3.1 Ground truth and bechmarks

The creation of suitable ground truth is very important for the training of weights. We designed the ground truth to cover the most important aspects of resource usage (CPU, disk, memory and their combination) with various intensities. For example, a process with 60% intensity does nothing for 40% of the execution time (executes IdleBlocks) and 60% of the time uses computer resources.

Using the ULS, we created seven categories of processes, each category ranging from 40% to 100% intensity with 5% steps:

- C45, C50 ... C100 – CPU load processes

- M45, M50 ... M100 – Memory load processes
- D45, D50 ... D100 – Disk load processes
- CD45, CD50 ... CD100 – Equally mixed CPU and disk load processes
- CM45, CM50 ... CM100 – Equally mixed CPU and memory load processes
- DM45, DM50 ... DM100 – Equally mixed disk and memory load processes
- CDM45, CDM50 ... CDM100 – Equally mixed CPU, disk and memory load processes

In each category, processes with intensity from 45% to 70% are labeled as a part of low-intensity cluster and processes from 75% to 100% as a part of the high-intensity cluster. Thus the ground truth data consists of 84 processes and is labeled into 14 clusters.

We hypothesize that if we are able to find weights, which improve the cluster analysis of the ground truth, the same weights will also improve the cluster analysis of real processes. The improvement of cluster analysis should then manifest as an improvement in prediction accuracy.

6.4 Benchmarks and boundary detection

The benchmarks used in boundary detection consists of all the ground truth processes and all of the real processes. Because the supplied processes are single-threaded, we approximated the final maximal values of CPU related properties as the measured maximal values multiplied by four, due to our computer having four cores.

For the evaluation, we decided to set the operational boundary as 140% of the used computer resources. The reason being that with our benchmarks, we probably did not reach exactly 100% of the computer capacity and the real maximal values are probably a few percentage points higher. In addition, the boundary set to 100% of computer resources excludes over 90% of measured combinations, which leaves us with not enough combinations for the experiments. Boundary set to 140% excludes around 80% of measured combinations, which is more reasonable.

6.5 Evaluation of the prediction methods

In this section, we compare the ability of different combinations of algorithms to perform cluster analysis and their effects on prediction accuracy. For every combination of the clustering algorithm, distance measure and normalization, we calculate *prediction score*. Based on this score, we then examine one of the combinations in more depth. In this set of experiments, we disregard the influence of weights, which are examined in Section 6.6.4.

6.5.1 Prediction score

The prediction score is a simplified measure of the accuracy of our prediction methods.

1. For a specified number of iterations:

(a) Take randomly 50% of all data matrices and use them to predict the 90th percentile of the execution time of the remaining 50%. Only closest friend, primary-cluster and cluster-cluster predictions are calculated, as the regression-based predictions are not affected by the method combinations.
(b) Calculate prediction improvement, which is the difference between the prediction error and baseline error. The baseline is the 90th percentile of the execution time of the primary process running alone.

2. Calculate the average improvement of prediction over baseline.

$$prediction\_score = \frac{\sum_{i \in I} \sum_{p \in P} improvement(i,p)}{I \times |P|} \tag{6.1}$$

Where I are iterations, P is the set of prediction methods (closest friend, primary-cluster and cluster-cluster). The prediction improvement is calculated as:

$$improvement(i,p) = \frac{\sum_{c \in C_{i,p}} |1 - \frac{baseline(c)}{real(c)}| - |1 - \frac{p(c)}{real(c)}|}{|C_{i,p}|} \tag{6.2}$$

Where $C_{i,p}$ is the set of combinations able to be calculated in given iteration by a given prediction method p and $real$ is the real percentile of the execution time of the primary processes in given combination.

Table 6.3 shows the prediction score for each combination of algorithms. We can notice that the top 6 combinations perform comparatively well. ZValue normalization performs worse than MinMax, especially when combined with APCorrelation distance, which results in a large drop of performance. We will examine mean-shift, APCorrelation and MinMax combination in-depth, as it results in the best performance.

Prediction Score	**Clustering**	**Distance**	**Normalization**
14,58	MeanShift	APCorrelation	MinMax
13,97	Affinity	Frobenius	ZValue
13,57	Affinity	Frobenius	MinMax
12,99	Affinity	APCorrelation	MinMax
12,7	MeanShift	Frobenius	MinMax
12,69	MeanShift	Frobenius	ZValue
2,71	Affinity	APCorrelation	ZValue
0,16	MeanShift	APCorrelation	ZValue
Average = 10,42			

Table 6.3: Prediction scores of combinations of algorithms using uncorrupted processes.

Cluster visualization

Figure 6.1 shows the clustering of processes calculated using mean-shift clustering, APCorrelation distance and MinMax normalization. In the figure, each dot

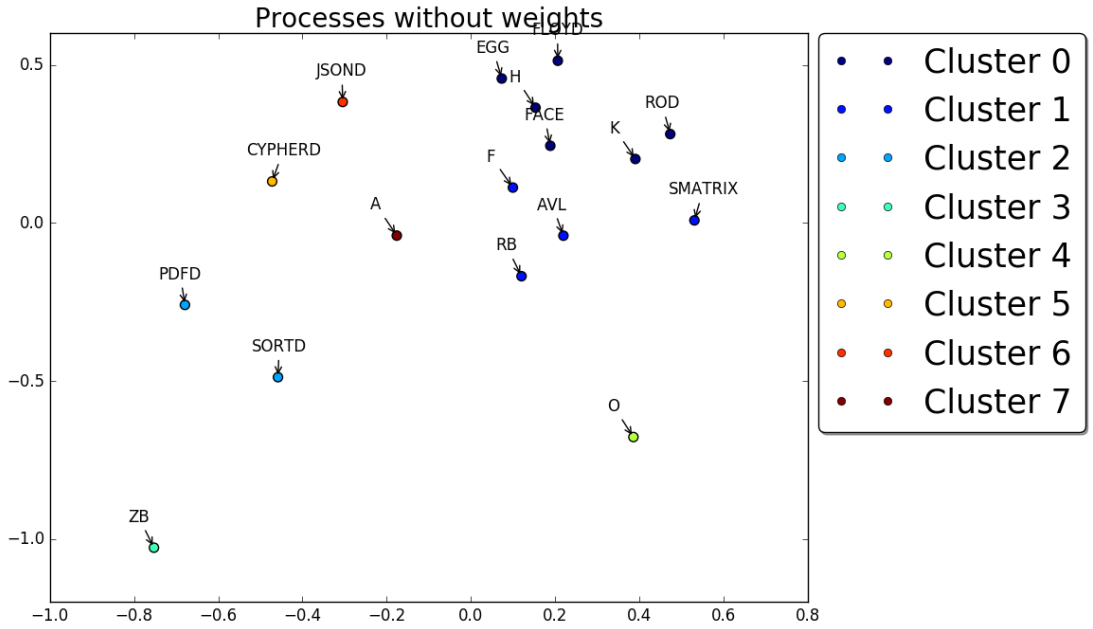

Figure 6.1: The clustering of uncorrupted processes using mean-shift clustering, APCorrelation distance and MinMax normalization.

represents a process and is labeled by its name using an arrow. The color of the dot represents the cluster in which the process belongs.

The visualization is created using multidimensional scaling (MDS implemented by sklearn[21]) of the process distance matrix (a matrix of distances between each pair of processes). MDS translates the information about the distance between pairs of processes into Cartesian coordinates in arbitrary dimensions (in our case, two dimensions).

6.5.2 Integrity test

The core of the in-depth analysis of prediction methods using mean-shift clustering, APCorrelation distance and MinMax normalization consists of integrity test:

1. Calculate clusters and process similarity
2. For integrity from 95% to 5% with 5% steps:
   (a) For a specified number of iterations:
      i. Take a percentage of data matrices equal to integrity, which will serve as the information used to form predictions. Single data matrices are always taken, the rest of the data matrices are chosen randomly.
      ii. Take the rest of the data matrices and use them as a target of the predictions.
      iii. For each target data matrix, form a prediction of the 90th percentile and compare it to the real value.
      iv. Calculate the average error, percentage of unsuccessful predictions and the average improvement over baseline.

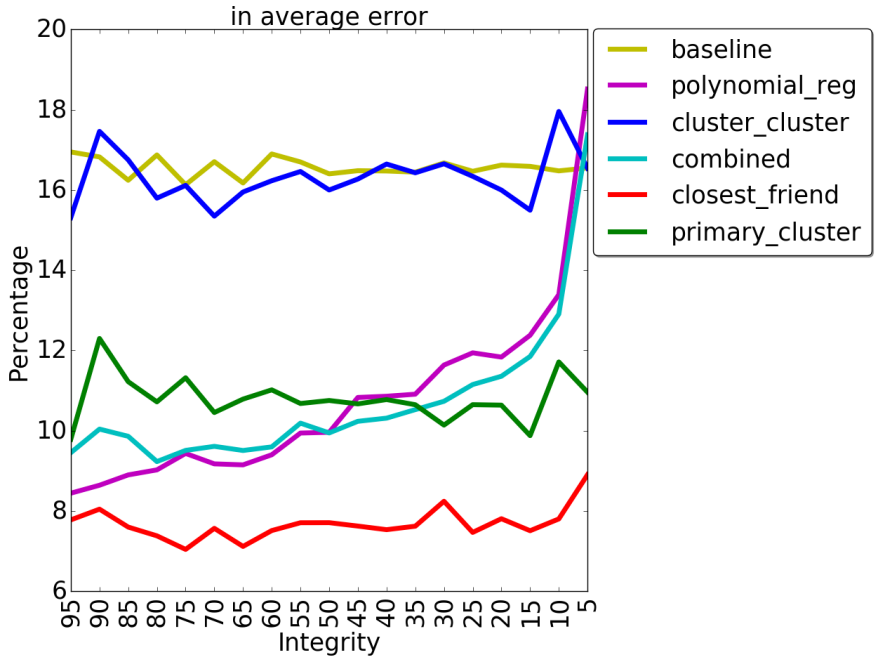

Figure 6.2: Integrity test average prediction error inside the boundary.

In our tests, we used ten iterations. The reason behind this integrity test is that it is important to evaluate the prediction methods with various amounts of information available to form predictions.

We have divided the visualization of results to outside and inside of the boundary. As already mentioned in Section 6.4, the boundary is set to 140% of the computer capacity.

To form the polynomial regression prediction, we use third-degree polynomial. The reason being, that first and second degrees resulted in very low accuracy, and the fourth degree is too computationally demanding.

6.5.3 Average Error

First let us examine the average error of each prediction method, which is calculated for each integrity level and each prediction method p as:

$$average\_error = \frac{\sum_{i \in I} \sum_{c \in C_i} |1 - \frac{p(c)}{real(c)}|}{\sum_{i \in I} (|C_i|)} \tag{6.3}$$

Where I are iterations and C_i is the set of combinations successfully predicted in a given iteration. The results are visualized in Figures 6.2 and 6.3. The baseline is again the 90th percentile of the primary process in a predicted combination when running alone.

Inside the boundary:

We can see that the closest-friend prediction has the best average error of around 8%. The polynomial regression is initially the second-best performing method, however, in the lower integrity levels, it is surpassed by primary-cluster prediction. We can also notice that the cluster-cluster prediction is the worst

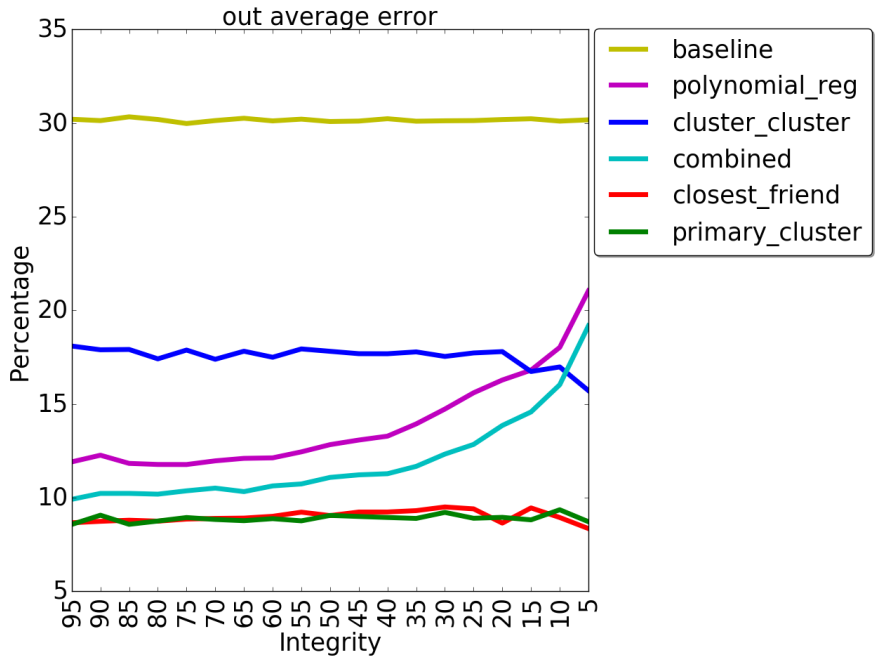

Figure 6.3: Integrity test average prediction error outside the boundary.

performing with almost the same average error as the baseline of around 17%. The combined prediction method forms prediction by first trying the closest-friend method, if unsuccessful due to insufficient data turns to primary-cluster, and, if still unsuccessful, uses the polynomial regression method. Unfortunately, at the 5% integrity level, the combined prediction method exceeds the baseline together with the polynomial regression prediction.

Outside the boundary:

We can notice that the closest-friend and the primary-cluster prediction have almost the same average error of around 8%. The regression method is again better than the cluster-cluster method. However, now the cluster-cluster method is significantly better than the baseline. The combined prediction first tries the closest-friend method, then primary-cluster followed by polynomial regression method and as a last resort uses the cluster-cluster prediction. We can again see a significant increase in the average error in the low integrity levels.

Both inside and outside:

It is important to note that in both cases, the closest-friend, primary-cluster and cluster-cluster predictions maintain very stable levels of average error throughout all integrity levels. Unfortunately, the polynomial regression prediction suffers from increased average error in the lower integrity levels, which also causes the combined prediction to increase in average error. Interestingly the closest-friend and cluster-cluster predictions maintain almost the same average error both inside and outside the boundary. The primary-cluster even performs with lower average error outside the boundary. We can also see that the boundary detection successfully eliminates combinations with a higher slowdown, as the baseline error is almost double outside the boundary when compared to inside the boundary.

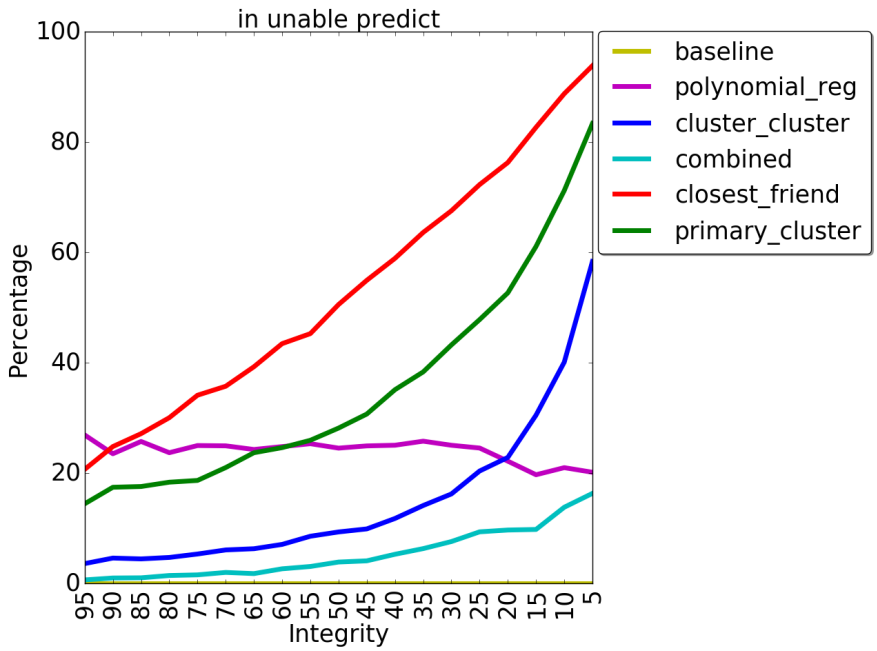

Figure 6.4: Percentage of predictions not calculated due to insufficient information inside the boundary.

6.5.4 Unable to calculate

Let us now focus on the percentages of successful predictions, which are visualized in Figures 6.4 and 6.5. A prediction can be unsuccessful due to an insufficient amount of information provided to form the prediction. Each prediction method calculated using a different set of data matrices, and as a result, some prediction methods can perform the prediction more often than others.

Inside the boundary:

Initially, all of the prediction methods except for the cluster-cluster method start around the 20% of unsuccessful prediction with regression prediction having the most unsuccessful predictions. The closest-friend prediction performs much better inside the boundary than outside.

Outside the boundary:

We can see that the closest-friend prediction has the highest amount of unsuccessful predictions by far. The reason being, that outside the boundary lie combinations with a higher number of processes (higher arity), which is harder to predict for closest-friend prediction. The polynomial prediction becomes the second-best performing in lower integrity levels.

Both inside and outside:

As explained in Section 4.6, we expect the closest-friend, primary-cluster and cluster-cluster to have an increasingly higher percentage of successful predictions, which is confirmed in both cases. Unfortunately, closest-friend, which is the method with best average error, also has by far the highest amount of unsuccessful prediction. Interestingly, the polynomial prediction performs better outside the boundary, with only around 15% of predictions unsuccessful due to predicting extreme values. The polynomial prediction also has the most stable percentage of unsuccessful prediction across all integrity levels. Finally, we can see that the

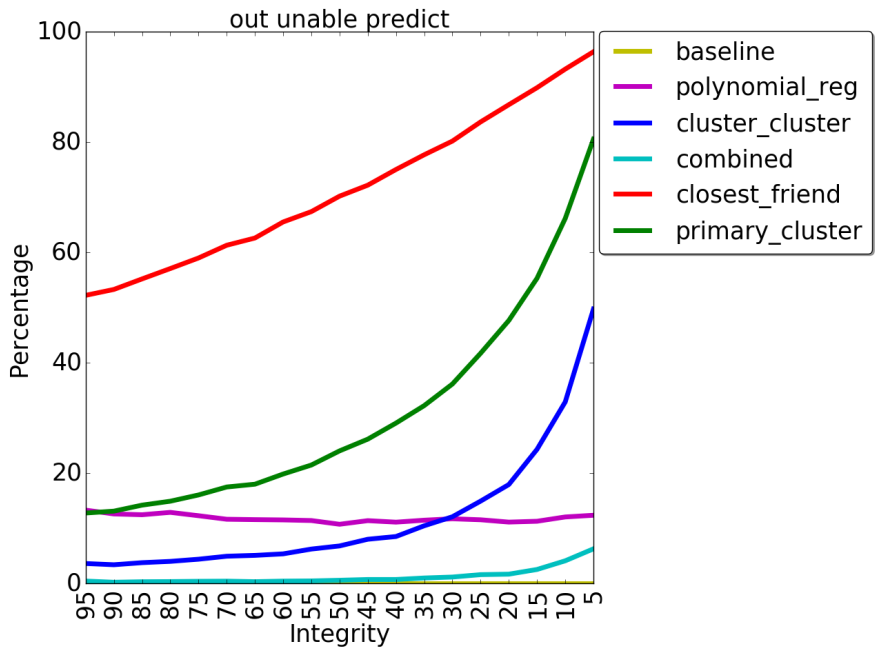

Figure 6.5: Percentage of predictions not calculated due to insufficient information outside the boundary.

combined prediction method can predict almost all combinations.

6.5.5 Average improvement over baseline

Because each prediction method is able to calculate a different set of combinations of processes, it is necessary to provide a more detailed comparison to the baseline. For example, the closest-friend prediction having a low average error is meaningless, if the baseline has an even lower average error for the same combinations, which the closest-friend prediction is capable of predicting. This could happen if the closest-friend prediction method could calculate only low arity combinations without much change in execution time.

As a result, we need to compare the average error of each prediction method to the average baseline error on the same process combinations, which is the same as the prediction score in Section 6.5.1. Figures 6.6 and 6.7 show an individual prediction score for each prediction method.

Inside the boundary:

We can see that even though the closet-friend prediction had the lowest average error, it does not have the highest average improvement over baseline, which belongs to the polynomial regression. We can also notice that the cluster-cluster method, which had almost the same average error as the baseline, even has negative average improvement. Importantly in the 5% integrity level, the polynomial and the combined prediction method decreases below the baseline.

Outside the boundary:

Outside the boundary, all of the prediction methods have a positive improvement over the baseline in all integrity levels. We can again see the decrease in performance of the polynomial and combined prediction method at the 5%

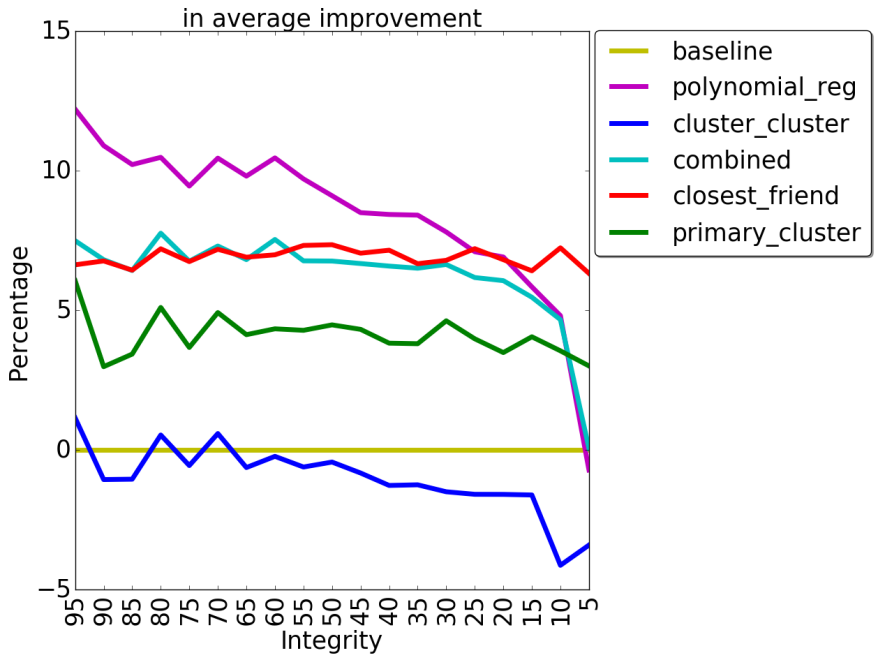

Figure 6.6: Integrity test average improvement over baseline inside the boundary.

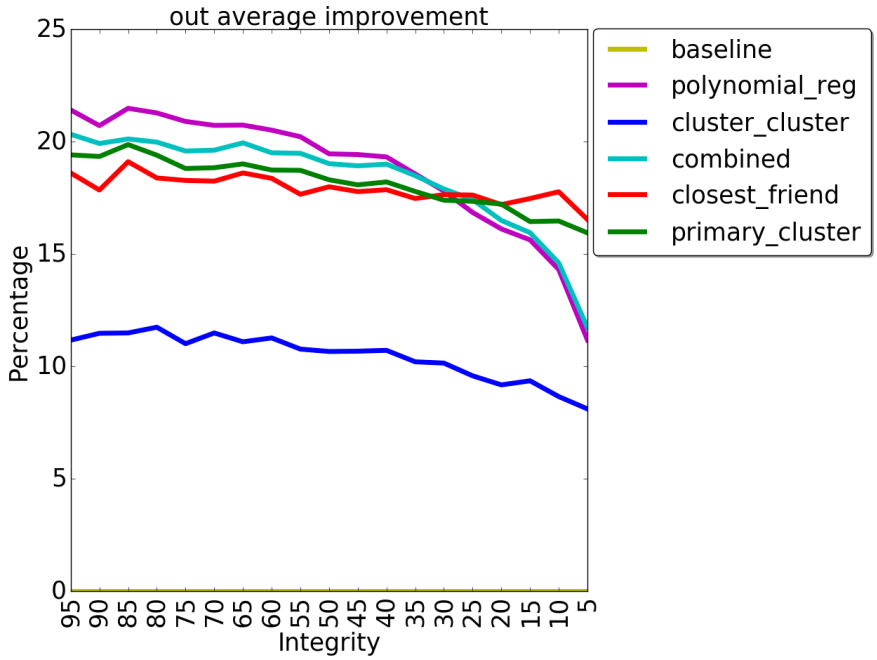

Figure 6.7: Integrity test average improvement over baseline outside the boundary.

integrity level, however, the methods still stay well above the baseline. The primary-cluster prediction performs better than the closest friend outside the boundary, which is the opposite inside the boundary.

Both inside and outside: We can see that in both cases, the polynomial regression prediction has the best performance, and the cluster-cluster prediction has the worst performance. In both cases, the combined prediction method also follows the polynomial regression. This is due to polynomial prediction having many more successful predictions in the lover integrity levels.

6.5.6 Confusion matrix

As explained in section 3.1, we propose a layer where a solver uses the predictor to assigns combinations of processes on edge-cloud servers. This is done in order to satisfy the soft real-time requirements of developers. Let us consider the following requirement: In 90% of calculations, a process is required to complete with maximally 10% slowdown when compared to the execution time of the process running alone.

It is important to examine, how succesful the predictor is in detecting, which combination of processes satisfy such a requirement and which does not. Figures 6.8 and 6.9 visualize the average confusion matrix (10 iterations of itegrity test) of the combined prediction method. The chart shows the average percentage of combinations, for which applies:

- TP – true positive – prediction:satisfied, reality:satisfied
- TN – true negative – prediction:not satisfied, reality:not satisfied
- FP – false positive – prediction:satisfied, reality:not satisfied
- FN – false negative – prediction:not satisfied, reality:satisfied

Inside the boundary:

We can see that the majority (around 85%) belong to true positive and true negative cases, with a true positive majority (around 55%).

Outside the boundary:

We can notice that around 90% are either true positive or true negative. The majority belongs to true negative (around 70%), which is in contrast to the cases inside the boundary.

Both inside and outside:

In our situation, the most problematic category is false positive, in which case we would assign a combination on a server, which would not satisfy the requirements set by the developers. We can see that this category is the least represented one, both inside and outside the boundary, with only around 5% of predicted combinations.

6.5.7 Results analyzed

The primary-cluster prediction is the most accurate, however, it comes with a high percentage of unsuccessful predictions. In our opinion, the combined prediction method is the most suitable one, as it still maintains high accuracy and is able

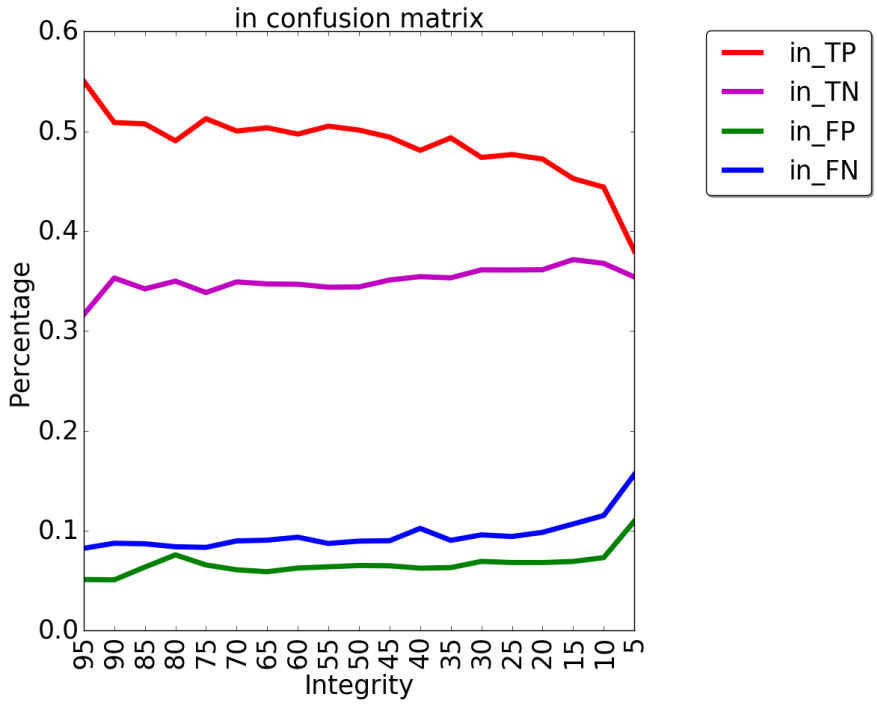

Figure 6.8: Visualization of confusion matrix for each integrity level inside the boundary for the combined prediction method.

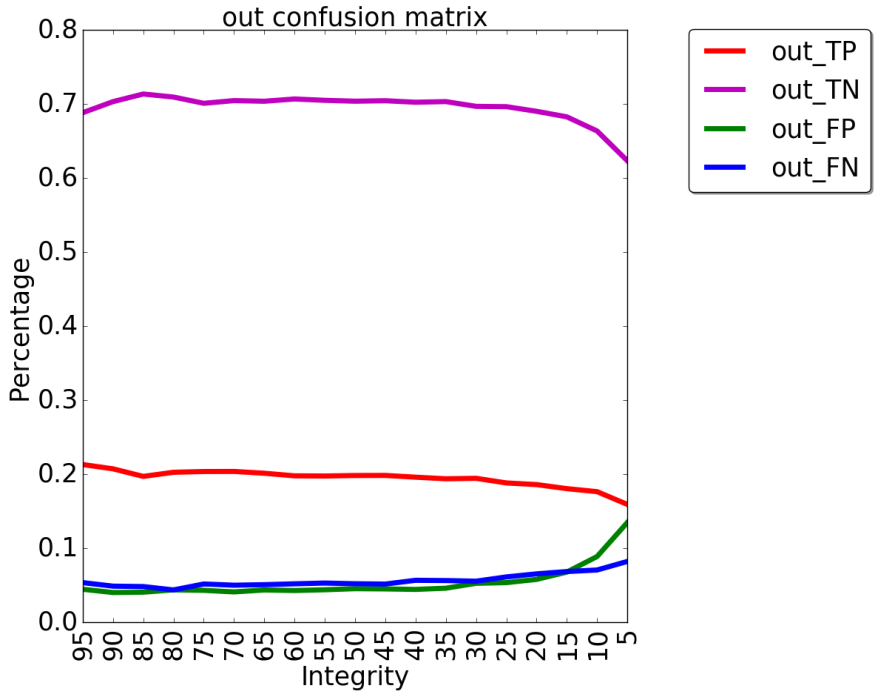

Figure 6.9: Visualization of confusion matrix for each integrity level outside the boundary for the combined prediction method.

to often successfully complete a prediction. The calculation of one combined prediction takes about 0.01 seconds on average.

We can see that the performance of prediction methods gets significantly worse at the 5% integrity level. As a result, the predictor should be supplied with at least 10% of the combinations measured in order to perform well.

The boundary detection system proved to be useful by selecting combinations with lower slowdown inside the boundary. This results in the majority of prediction cases being true positive inside the boundary (around 55%). Outside the boundary, only about 20% of the predictions are true positive.

With all of the tests analyzed, we think that the closest-friend, primary-cluster, polynomial regression, cluster-cluster prediction methods and the boundary detection system to be a success.

6.6 Evaluation of weights

In this section, we compare the ability of different combinations of algorithms to train weights, perform cluster analysis and their effects on prediction accuracy. The comparison consists of several steps:

1. For a specified number of iterations:
   (a) Randomly introduce noise into the data matrices
   (b) Cross-validate weights trained by each combination of algorithms using ground truth data
   (c) For each combination use all ground truth data to calculate weights and calculate prediction score for the 50% integrity level, with and without the weights
2. Average the results and calculate the Pearson Correlation Coefficient between improvement in cross-validation and improvement in prediction score

We hypothesize that if we find weights, which improve the cluster analysis of ground truth data, the same weights also improve cluster analysis of real processes. The improvement of cluster analysis should result in the improvement of prediction accuracy. If this is the case, we should find a high correlation between the improvement in cross-validation and improvement in prediction score.

6.6.1 Introducing noise

Because we do not know, which measured properties are useful and which are useless, we introduce noise into the data. This way, we can be sure which properties should have low importance in the weights and evaluate how useful are weights in finding these erroneous properties.

To introduce noise into the data matrix, we added 9 non-scaling properties to each process. These properties can have one of the following values:

- $[1, 1, 1, 0, 0, 0, 0, 0, 0]$
- $[0, 0, 0, 1, 1, 1, 0, 0, 0]$

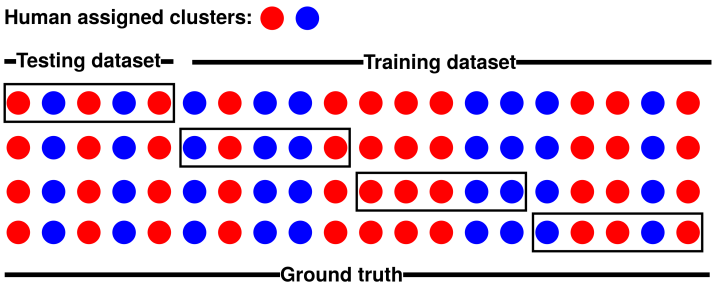

Figure 6.10: Illustration of 4-fold cross-validation dataset selection.

- $[0, 0, 0, 0, 0, 0, 1, 1, 1]$

Thus random processes will either share a complete similarity in three properties or complete dissimilarity in 6 properties. The higher number of properties is needed in order to sufficiently confuse the similarity calculation and force otherwise distant processes into the same clusters. As a result, we can evaluate how the prediction methods work with confused similarity and cluster analysis and how much do the weights improve the prediction.

6.6.2 Weights cross-validation

After calculating the weights on a dataset, it is not guaranteed that they will also improve other datasets. For example, even if the weights improved cluster analysis of the training dataset substantially, they could actually be detrimental on other datasets due to factors such as overfitting or selection bias. Overfitting is a case of over-specialized weights and selection bias represents wrongly selected training data set.

As a result, we need to confirm that the weights trained on the training dataset improve the cluster analysis on other datasets as well. We do this using a technique called cross-validation. The process of cross-validation is following:

1. For a specified number of splits:
   (a) Split the ground truth into training and testing dataset
   (b) Train the weights on the training dataset
   (c) Run cluster analysis on the testing dataset with and without the trained weights.
   (d) Calculate the clustering score with and without the weights.
2. Calculate the average improvement of clustering score caused by weights

We use a specialized version of cross-validation called k-fold, in which the ground truth dataset is randomly split into k equal-sized partitions. One partition is then used as a testing dataset and the rest as the training dataset. Each partition is used exactly once as a testing dataset.

For example, 4-fold cross-validation creates four partitions, thus 25% of the ground truth is used as a testing dataset and the remaining 75% as the training dataset. As a result, four iterations are performed as illustrated in Figure 6.10.

The main advantage of k-fold cross-validation is that we can generate multiple train/test scenarios using the same ground truth dataset, and analyze the average performance while maintaining the same ratio of train and test data. The averaging of the test scores is done in order to reduce the effects of overfitting and selection bias.

6.6.3 Pearson Correlation Coefficient

It is necessary to confirm the hypothesis that weights trained on the ground truth processes also improve cluster analysis of real processes. If the hypothesis was correct, the improvement in the clustering of the ground truth data should be correlated to the improvement of prediction accuracy. To confirm this, we calculate the Pearson correlation coefficient between the improvement of the score during the cross-validation of weights and the improvement of prediction score, when using weights produces by the same combination of algorithms.

The Pearson correlation coefficient ranges between -1 and 1, where 1 means that a linear equation perfectly describes the relationship, where the prediction score increases as the improvement in cross-validation increases. The value of -1 would mean the opposite, where the prediction score decreases linearly as the cross-validation score increases. Finally, no linear correlation would result in 0 value.

For a set of tuples $\{(x_1, y_1), ..., (x_n, y_n)\}$, where x is the improvement in cross-validation, y the improvement in prediction score and $\bar{x}$ is the sample mean (mean of all x values), the coefficient is calculated as:

$$P = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{6.4}$$

If the correlation is significant, we can conclude that a combination of algorithms, which produces weights with the ability to improve the clustering of ground truth data, also produces weights, which improve the clustering of real data in such a way, that it benefits our prediction methods.

6.6.4 Comparison of method combinations using corrupted data

We set the number of iterations in the test introduced in at the start of Section 6.6 to five, and we used 3-fold cross-validation. The averaged results are split by the scoring method and by the used optimization method, as can be seen in Tables 6.4, 6.5. To compare the weights found using simulated annealing to a baseline, we also measured the same tests using random weights, as shown in Table 6.6.

Contents of the tables are sorted by the weights effect column, which indicates the gain/loss of prediction score caused by weights. The CV-improvement column indicates the improvement in the cross-validation score caused by weights. PS stands for the final prediction score. Yellow row indicates the combination of algorithms with the highest prediction score.

Fowlkes	**Annealing**				
PS weighted	**Weights effect**	**CV-improvement**	**Clustering**	**Distance**	**Normalization**
9,36	8,34	0,34	Affinity	APCorrelation	MinMax
7,95	6,17	0,36	Affinity	APCorrelation	ZValue
10,31	3,32	0,31	Affinity	Frobenius	MinMax
10,93	2,46	0,23	Affinity	Frobenius	ZValue
12,38	1,80	0,04	MeanShift	Frobenius	MinMax
12,14	1,52	0,12	MeanShift	Frobenius	ZValue
11,88	0,17	0,01	MeanShift	APCorrelation	MinMax
7,63	-2,16	0,14	MeanShift	APCorrelation	ZValue
Average = 10,32	Average = 2,70	Average = 0,20			

Table 6.4: Comparison of algorithms using corrupted data, Fowlkes-Mallows scoring and Simulated Annealing optimization.

VMeasure	**Annealing**				
PS weighted	**Weights effect**	**CV-improvement**	**Clustering**	**Distance**	**Normalization**
9,70	8,45	0,33	Affinity	APCorrelation	MinMax
5,99	4,09	0,32	Affinity	APCorrelation	ZValue
9,72	2,57	0,28	Affinity	Frobenius	MinMax
12,57	2,17	0,03	MeanShift	Frobenius	MinMax
9,80	1,33	0,15	Affinity	Frobenius	ZValue
9,95	-0,52	0,04	MeanShift	Frobenius	ZValue
9,92	-1,78	0,00	MeanShift	APCorrelation	MinMax
7,76	-2,00	0,03	MeanShift	APCorrelation	ZValue
Average = 9,43	Average = 1,79	Average = 0,15			

Table 6.5: Comparison of algorithms using corrupted data, VMeasure scoring and Simulated Annealing optimization.

Random						
PS weighted	**Weights effect**	**Fowlkess CVI**	**VMeasure CVI**	**Clustering**	**Distance**	**Normalization**
3,77	2,79	0,01	0,02	Affinity	APCorrelation	MinMax
9,78	0,10	-0,01	-0,01	MeanShift	APCorrelation	ZValue
2,16	0,10	0,04	0,02	Affinity	APCorrelation	ZValue
8,78	0,09	0,07	0,04	Affinity	Frobenius	ZValue
10,39	-0,06	-0,03	-0,01	MeanShift	Frobenius	MinMax
10,77	-1,12	-0,04	-0,02	MeanShift	APCorrelation	MinMax
9,42	-1,20	-0,06	-0,02	MeanShift	Frobenius	ZValue
2,04	-5,13	0,01	0,00	Affinity	Frobenius	MinMax
Average = 7,14	Average = -0,55	Average = 0.00	Average = 0.00			

Table 6.6: Comparison of algorithms when using random weights, corrupted data and both Fowlkes-Mallows and VMeasure scoring. CVI stands for cross-validation improvement.

Analysis of simulated annealing

Let us first analyze the results of simulated annealing in Tables 6.4 and 6.5. In almost all cases, the weights were able to improve the prediction accuracy.

Effects of clustering algorithm:

Weights calculated using affinity propagation clustering have a stronger effect than mean-shift clustering. Even though weights found using mean-shift have a weaker effect, the final prediction score is higher when using this clustering method.

Effects of distance measure:

The effects of distance measures are mixed. In general, the APCorrelation seems to be more volatile as it yields both the highest and the lowest weights effect. On average, weights calculated using Frobenius distance have a better final prediction score.

Effects of normalization algorithm:

A combination using Zvalue almost always produces weaker weights and worse prediction score than when using MinMax normalization.

Effects of clustering score:

The Fowlkes-Mallows based combinations of algorithms perform better on average than those using VMeasure. On the other hand, VMeasure was able to achieve both the highest weights effect and the highest final prediction score.

Best performing combinations:

The combination of affinity propagation clustering, APCorrelation distance and MinMax normalization has the strongest weights effect when using both the Fowlkes-Mallows and VMeasure scoring. The combination of mean-shift clustering, Frobenius distance and MinMax normalization results in the best prediction score when using both the Fowlkes-Mallows and VMeasure scoring.

Analysis of random weights

Now let us focus on Table 6.6 containing results of randomly generated weights. We can see that both the prediction score and weights effect is much worse, and on average, the weights have a negative effect. We can also notice that the cross-validation improvement is almost zero in all cases.

Same as with the weights calculated using simulated annealing, the combination with the highest weights effect is affinity propagation clustering, APCorrelation distance and MinMax normalization. Also, the combination of mean-shift clustering, Frobenius distance and MinMax normalization almost results in the best prediction score.

Pearson correlation coefficient

Table 6.7 contains the Pearson correlation coefficients between the improvement of the clustering score in cross-validation and the weights effect (improvement of prediction score due to the weights). The coefficient is calculated separately for weights trained using simulated annealing, randomly found weights, and finally, for both approaches combined.

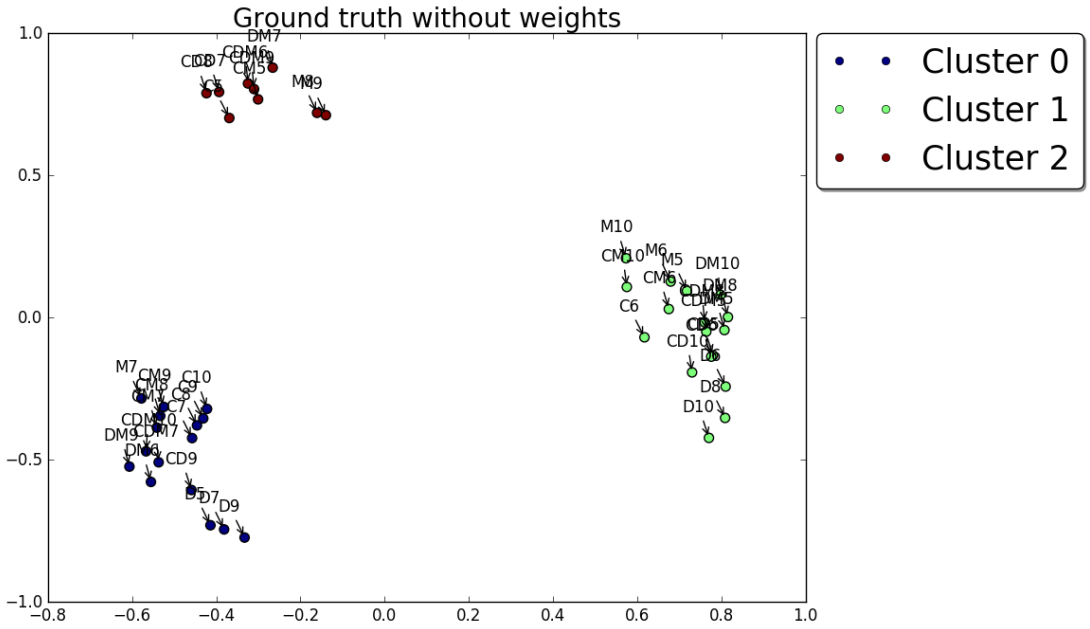

Figure 6.11: Clustering of ground truth without weights.

We can see that there is a strong correlation between the improvement of the clustering score of the training dataset and the improvement of prediction accuracy when using the simulated annealing. We think that the reason why the correlation is lower for the random weights is due to almost zero cross-validation improvement.

	Fowlkes	**VMeasure**
Annealing	0,77	0,84
Random	0,12	0,34
Combined	0,75	0,81

Table 6.7: Pearson correlation coefficient calculated between the improvement of clustering score in cross-validation and improvement of prediction score.

6.6.5 Best training combination examined

Let us now focus on the combination of Fowlkes-Mallows scoring, affinity propagation clustering, APCorrelation distance function and MinMax normalization, which provides weights with the largest improvement in prediction score.

Figures 6.11 and 6.12 show the clustering of ground truth processes before and after the use of weights. For visual clarity, these figures do not show processes with odd intensity (e.g., C45). We also subtracted one zero from the process names (e.g., C50 becomes C5 and C100 becomes C10), however, this does not affect the number of clusters shown, nor does it affect the overall information provided in the figures.

We can see that the clustering was essentially random without the weights. With weights, we can see 7 clusters, which correspond to the main differences in resource usage. As a result, through the process of training, we were able to improve the ability to detect meaningful clusters considerably.

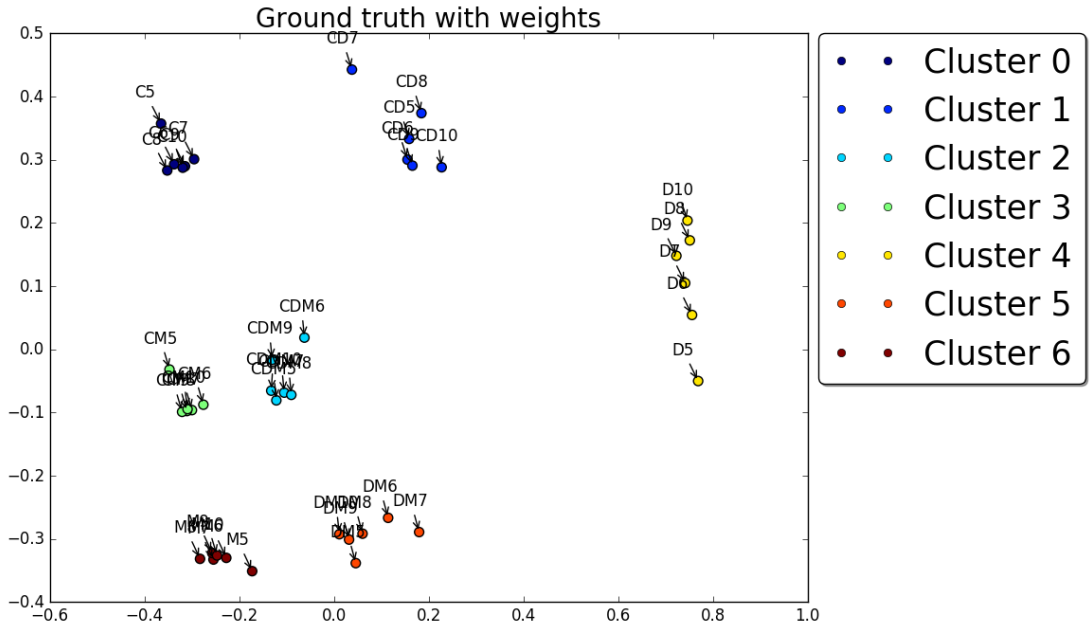

Figure 6.12: Clustering of ground truth with weights.

Figures 6.13 and 6.14 show the clustering of real processes before and after the use of weights. Again we can see that without weights, the clustering is almost random. However, with the use of weights, we can see that processes using disk separated nicely from the rest and even separated by the level of disk usage into high, mid and low disk-intensive processes. Overall the clustering strongly resembles the clustering of uncorrupted processes shown in Figure 6.1.

At last, Figures 6.15 and 6.16 show the progress of the clustering score during the training and the final trained weights. We can see that the initial clustering score was very low and increased more than three-fold during the process of training. The final weights have almost perfectly erased the influence of erroneous noise properties while maintaining the importance of the original ones.

6.6.6 Results analyzed

The weights trained using simulated annealing are able to detect erroneous properties and improve the accuracy of the prediction methods. We have demonstrated that this is not caused by luck, as random weights negatively impacted the prediction methods.

We have also demonstrated that the improvement of the clustering of ground truth data correlates strongly with the improvement in prediction accuracy. When examining the clustering of ground truth and real processes before and after the use of weights, we can see that the clustering is truly improved, which is the cause of improvement of the prediction methods accuracy.

With the experiments analyzed, we think that the weights training system is a success.

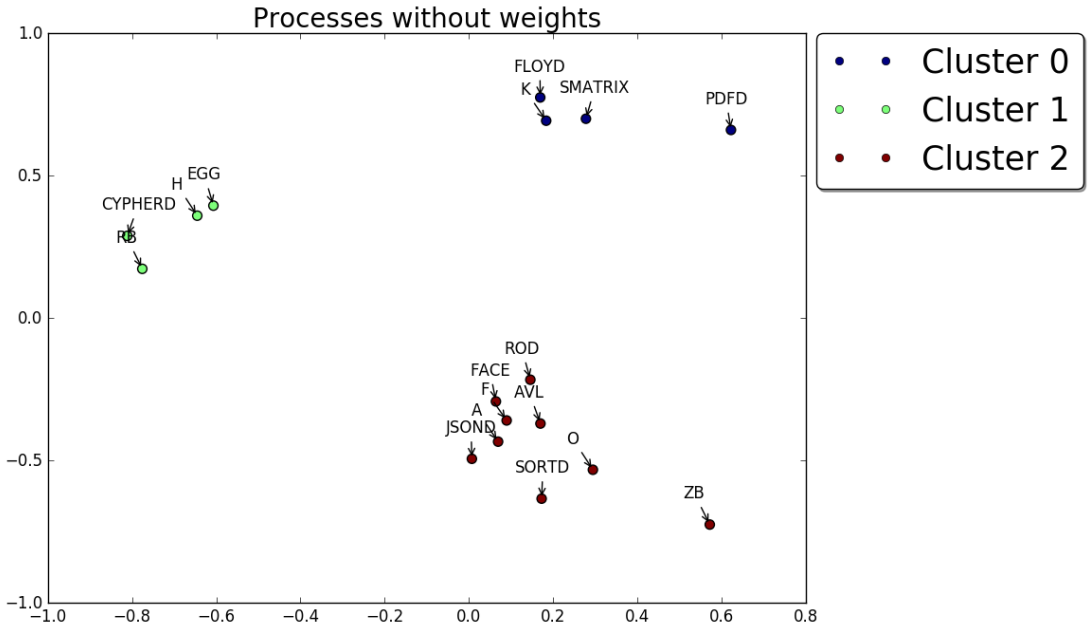

Figure 6.13: Clustering of real processes without weights.

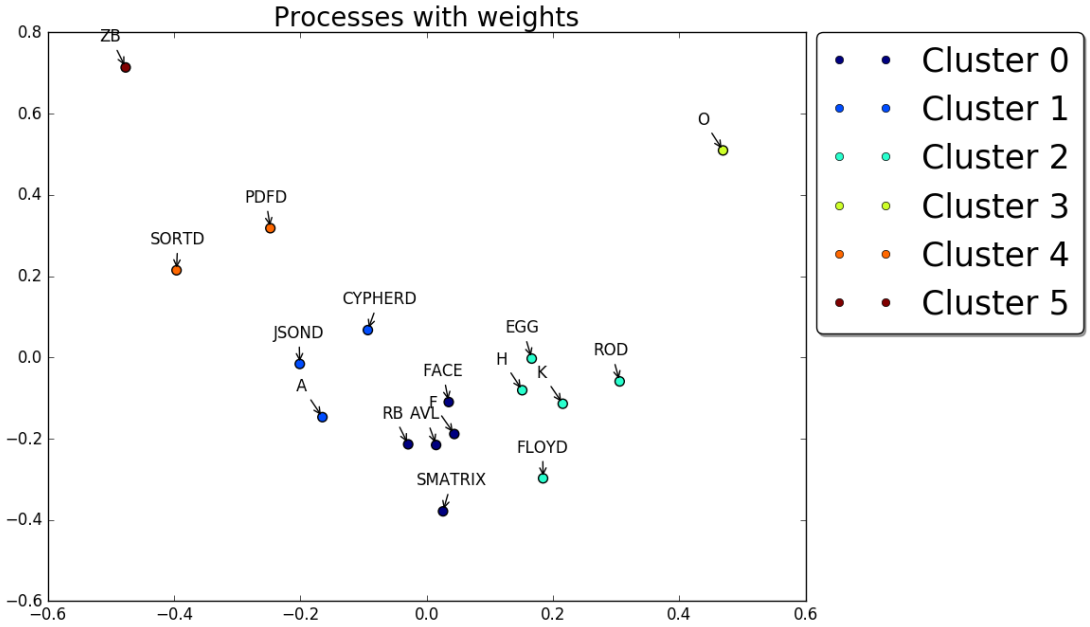

Figure 6.14: Clustering of real processes with weights.

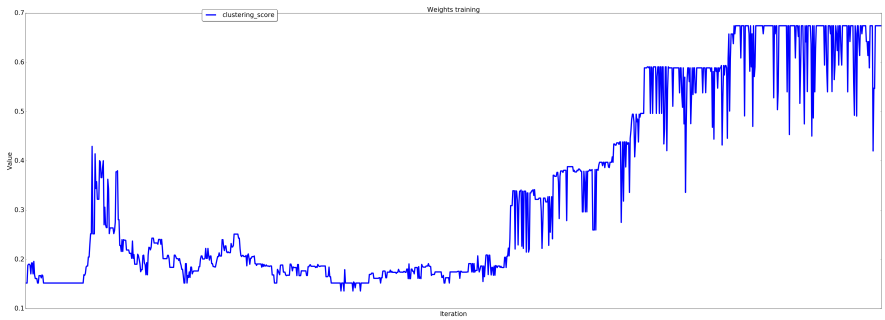

Figure 6.15: Clustering score progress during training.

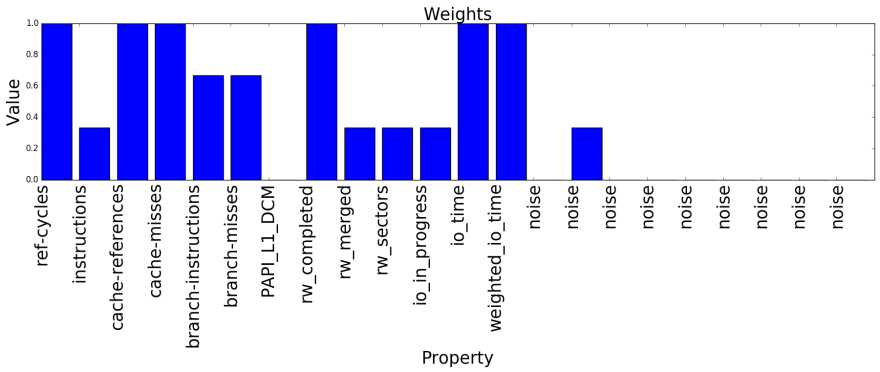

Figure 6.16: Trained weights used in Figures 6.12 and 6.14.

6.7 Related work

To position our work in the context of other research in the area, we review some of the works (ordered chronologically) that we consider most relevant to our approach.

Q-Clouds[22] is a QoS-aware control framework, which adjusts resource allocation to mitigate the effects of interference on shared resources. Q-Cloud first profiles the virtual machines (VM) submitted by clients on a staging server to assess the amount of resources needed to attain the desired QoS without interference and then manages the resources allocated to the deployed VMs in a closed control loop.

Contrary to Q-Clouds our approach focuses on container technologies instead of VMs. Also, instead of allocating resources to deployed processes, we instead re-deploy process to satisfy the requirements set by the developers. Similarly to Q-Clouds we also utilize a data acquisition phase to measure the resource usage.

Paragon[23] is an online interference-aware scheduler, which uses collaborative filtering to classify incoming applications based on limited profiling and similarity to previously scheduled applications. It does not differentiate between batch and latency-sensitive applications and schedules applications so as to minimize interference and maximize utilization. Applications are classified for interference tolerance using microbenchmarks stressing a specific shared resource with tunable intensity, which are run concurrently with an application to find out the interference level at which the application's performance falls below 95% of its performance in isolation.

Similarly to Paragon, our approach considers the similarity of applications. Contrary to our approach, Paragon uses proxy microbenchmark workloads to estimate the performance of processes, while we rely on the measurement of interference of real processes.

CloudScope[24] is a representative of model-based approaches to QoS-aware cloud resource management and uses a discrete-time Markov Chain model to predict performance interference of colocated VMs. CloudScope runs within each host and collects application and VM-related metrics at runtime. The metrics serve to maintain an application-specific model capturing the proportion of the time an application uses a particular resource. The model is then used to predict

slowdown due to collocation and ultimately to control the placement of guest VM instances as well as adjusting the resources available to a hypervisor.

Contrary to our approach, CloudScope focuses on VMs instead of containers. However, similarly to our approach, CloudScope focuses on interference caused by collocation and solves it using deployment control. CloudScope also uses predictions based on the knowledge of resource usage of the VMs. Contrary to our approach, CloudScope does not utilize the similarity of the VMs and relies instead on the Markov Chain model to form the predictions.

CtrlCloud[25] is a performance-aware cloud resource manager and controller, which optimizes the allocation of CPU resources to VMs to meet QoS targets. It maintains an online model of the relationship between allocated resource shares and the application performance and uses a control loop to adapt the resource allocation so as to progress towards a probabilistic performance target expressed as a percentile of requests that must observe a response time within certain bounds.

Again CtrCloud focuses on VMs instead of containers. Similarly to our approach, CtrCloud aims to guarantee the performance of an application expressed as a percentile of execution time, however, CtrCloud uses CPU resource allocation instead of re-deployment to satisfy the requirements.

Pythia[26] is a collocation manager, which uses a linear regression model to predict combined contention on shared resources (the conflict over access to the resource) when colocating multiple batch workloads with a latency-sensitive workload. Pythia performs contention characterization for each batch workload running together with a particular latency-sensitive workload and removes batch workloads that are too contentious to allow safe collocation. It then selects a small subset of batch workloads to colocate with a latency-sensitive workload and measures their combined contention to build a linear regression prediction model for contention due to multiple batch workloads.

Similarly to Pythia, our approach profiles processes in a measurement setup and we also utilize regression models. However, contrary to Pythia, we also calculate the similarity of processes in order to form predictions.

In general, the approaches presented above, including our own, aim to provide performance and an interference-aware system, which manages resource allocation in a cloud environment to achieve efficient utilization of available resources while allowing applications to meet their QoS target. Our selection illustrates the variety of approaches proposed over the years, each designed for a different context.

Another related area is represented by works targeting *service-level agreements* (SLAs) in (edge-)cloud environment. SLA is a commitment to provide a service with certain aspects, such as with certain throughput, jitter, mean time between failures, etc. In general, the main difference between our approach and classic SLAs for clouds is that our approach is focused on providing soft-realtime guarantees on execution times, while the SLAs can focus on many different aspects of the service.

In the following, we review some of the approaches that differ from classic SLAs and bear more similarity to our approach.

The work of Remesh et al.[27] deals with SLA-aware scheduling and load-balancing. While the general idea is similar to ours, their primary goal is to load-balance services (i.e., re-deploy them to another computer in the cloud) in

order to keep computers in the cloud evenly loaded and reduce the chance of SLA breaches. The approach does not measure service response time and instead relies on low-level information, which has to be provided by the application developer, such as the required amount of memory, CPU speed in terms of instructions-per-second, etc.

This is in direct contrast with our approach, where we discourage the use of low-level information provided by the developers and instead measure the execution (response) times of combinations of processes. The boundary system implemented in our approach can be viewed as a way to provide a certain amount of memory and CPU instructions-per-second, however, our system does not require the input of a developer.

Cerebro[28] is similar to our approach in that it first statically analyzes services to find out important calls, measures the performance of these calls at runtime, and predicts bounds on response time using time-series analysis. The main difference is that Cerebro focuses only on a specific set of calls while our approach measures the performance of a whole process. Our approach does not rely on a knowledge of the importance of different calls, however, we can estimate the importance of measured process properties using weights. In addition, our approach focuses on slowdown caused by the collocation of process, however, Cerebro focuses on predicting the performance of an application, which composes of multiple calls to different services.

Panda et al.[29] present an SLA-based scheduling algorithm for a cloud. Similarly to our approach, the authors consider clouds composed of multiple data centers and schedule service for deployment to data centers so as to honor the SLAs. Contrary to our approach, the authors expect SLAs to be provided with the services and consider only execution time, cost, and the penalty for SLA violation. The scheduling algorithm then attempts to minimize service execution time and cost.

We can notice that many of the approaches presented above focus on VMs, however, we focus on container technologies. The developers are required to provide the processes and to specify their soft real-time requirements explicitly. Consequently, we only admit applications for deployment if the predictor considers the requirements to be satisfiable. Other than that, we treat the processes as a black box.

In our approach, we form predictions of the percentile of the execution time by combining cluster analysis and polynomial regression analysis, and we can enforce operational boundaries on our prediction algorithm. On top of that, the weights system provides the capability to automatically detect errors in the provided data. We think that the novelty in our approach is this combination of many strategies, which together create a robust prediction system.

7. Conclusion

To summarize, we have proposed a new layer, in order to provide soft real-time guarantees on the execution times of processes running in an edge-cloud system. This layer is composed of a solver, which searches for suitable combinations of processes to deploy using a predictor of a percentile of execution time. We have focused on the design of the predictor, which is the most important component when guaranteeing the soft real-time requirements.

We have introduced several prediction methods based on the similarity between processes, cluster analysis and regression analysis. With all of the prediction methods combined, we have achieved high levels of accuracy.

The predictor can also detect if a combination of processes exceeds a certain percentage of the capacity of the computer using boundary detection. The boundary detection system proved to be useful in restricting the predictor and avoiding unreliable predictions.

In case the predictor is provided with erroneous data, we have also created a system of weights, which can detect the errors and correct them. The weights system works successfully and can almost completely erase the influence of an error.

With these features combined, we are confident that the predictor is capable enough, when combined with a simple deployment solver, to provide a soft real-time guarantee on the execution time of a process collocated with other processes. As a result, we think that the goals of this thesis have been completed.

7.1 Future work

We see great potential in the prediction methods based on cluster analysis. We think that exploring fuzzy and overlapping clustering could provide interesting results. Similarly, an exploration of alternative regression analysis methods could be worthwhile.

Boundary detection could be extended by a dynamic solution, where confidence in the prediction is estimated for each prediction is very interesting. Another opportunity is to extend the ground truth data set for weights training. New blocks for the universal load simulator can be designed, and new process categories can be added.

Perhaps the most interesting extension would be to measure a greater amount of real processes and more combinations of those processes to analyze.

Bibliography

[1] Schahram Dustdar Weisong Shi. The promise of edge computing. *IEEE Computer Journal*, Vol. 49(No. 5):78–81, 2016.

[2] M. Armbrust et al. A view of cloud computing. *Commun. ACM*, Vol. 53(No. 4):50–58, 2010.

[3] ZHANG J. VOAS, J. Cloud computing: New wine or just a new bottle? *IT Professional*, Vol. 11(No. 2):15–17, 2009.

[4] Timothy Grance Peter Mell. The nist definition of cloud computing. *NIST Special Publication 800-145*, 2011.

[5] R. Griffith A. Joseph R. Katz A. Konwinski G. Lee D. Patterson A. Rabkin I. Stoica M. Zaharia M. Armbrust, A. Fox. Above the clouds: A berkeley view of cloud computing. 2009.

[6] Ali Khajeh-Hosseini Ilango Sriram. Research agenda in cloud technologies - submitted to the 1st acm symposium on cloud computing. 2010.

[7] Santosh Kumar and R.H. Goudar. Cloud computing–research issues, challenges, architecture, platforms and applications: A survey. *International Journal of Future Computer and Communication*, Vol. 1(No. 4), 2012.

[8] Microsoft. Edge computing. `https://www.microsoft.com/en-us/research/project/edge-computing`. Accessed: 24.10.2019.

[9] Quan Zhang Youhuizi Li Lanyu Xu Weisong Shi, Jie Cao. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, Vol. 3(No. 5):637–646, 2016.

[10] P. Maniatis M. Naik A. Patti B.-G. Chun, S. Ihm. Clonecloud: Elastic execution between mobile device and cloud. *Proc. 6th Conf. Comput. Syst., Salzburg, Austria*, pages 301–314, 2011.

[11] Z. Qin Q. Li S. Yi, Z. Hao. Fog computing: Platform and applications. *Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb), Washington, DC, USA*, pages 73–78, 2015.

[12] K. Ha et al. Towards wearable cognitive assistance. *Proc. 12th Annu. Int. Conf. Mobile Syst. Appl. Services, Bretton Woods, NH, USA*, pages 68–81, 2014.

[13] 5G-PPP. 5g automotive vision. `https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Automotive-Vertical-Sectors.pdf`, 20.10.2015.

[14] scikit. scikit-learn. `https://scikit-learn.org/stable/index.html`. Accessed: 24.10.2019.

[15] Delbert Dueck Brendan J. Frey. Clustering by passing messages between data points. *Science*, Vol. 315:972–976, 16.2.2007.

[16] Peter Meer Dorin Comaniciu. Mean shift: A robust approach toward feature space analysis. In *In PAMI*, pages 603–619, 7.8.2002.

[17] Jan K. Lenstra Emile Aarts. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[18] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.

[19] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[20] scipy. scipy.optimize.nnls. `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.optimize.nnls.html`. Accessed: 24.10.2019.

[21] sklearn. scipy.manifold.mds. `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html`. Accessed: 24.10.2019.

[22] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: Managing Performance Interference Effects for QoS-aware Clouds. In *Proceedings of EuroSys 2010, Paris, France*, pages 237–250. ACM, 2010.

[23] Christina Delimitrou and Christos Kozyrakis. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *Proceedings of ASPLOS 2013, Houston, USA*, pages 77–88. ACM, 2013.

[24] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, F. Franciosi, and W. Knottenbelt. CloudScope: Diagnosing and Managing Performance Interference in Multi-tenant Clouds. In *Proceedings of MASCOTS 2015, Atlanta, USA*, pages 164–173, 2015.

[25] O. Adam, Y. C. Lee, and A. Y. Zomaya. CtrlCloud: Performance-Aware Adaptive Control for Shared Resources in Clouds. In *Proceedings of CCGrid 2017, Madrid, Spain*, pages 110–119, 2017.

[26] Ran Xu, Subrata Mitra, Jason Rahman, Peter Bai, Bowen Zhou, Greg Bronevetsky, and Saurabh Bagchi. Pythia: Improving Datacenter Utilization via Precise Contention Prediction for Multiple Co-located Workloads. In *Proceedings of Middleware 2018, Rennes, France*, pages 146–160. ACM, 2018.

[27] Kaippilly Raman Remesh Babu and Samuel Philip. Service-level agreement-aware scheduling and load balancing of tasks in cloud. *Software: Practice and Experience*, 49(6):995–1012, 2019.

[28] Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski. Response time service level agreements for cloud-hosted web applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15*. ACM Press, 2015.

[29] Sanjaya K. Panda and Prasanta K. Jana. SLA-based task scheduling algorithms for heterogeneous multi-cloud environment. *The Journal of Supercomputing*, 73(6):2730–2762, 2017.

List of Figures

List of Tables