FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

Bachelor's thesis

Design and implementation of a scalable server for parallelization of optimization algorithms execution

Lukáš Forst

Department of Computer Science
Supervisor: Ing. Ondřej Vaněk, Ph.D.

May 13, 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Forst** Jméno: **Lukáš** Osobní číslo: **465806**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Návrh a vývoj škálovatelného serveru pro paralelizaci běhů optimalizačních algoritmů

Název bakalářské práce anglicky:

Design and implementation of a scalable server for parallelization of optimization algorithms execution

Pokyny pro vypracování:

Large-scale optimization problems are non-trivial to solve and require a significant amount of computational resources as well as computational time to find a solution. The challenge is to solve not only a single task but a multitude of them in a parallel manner. Additionally, the tasks are non-homogeneous, often describing a different problem. This problem can be solved by designing and intelligent scheduler able to schedule such tasks on a distributed computational platform.
The goal of the thesis is:
Study the state-of-the-art approach to computational tasks scheduling. Study various types of optimization problems and approaches and understand their computational needs. Study the distributed scalable architecture and approaches to schedule tasks on such architecture. Design a scheduling and load-balancing module able to ingest various optimization tasks and schedule them with respect to several criteria. Implement the scheduler. Evaluate the scheduler on a number of scenarios.

Seznam doporučené literatury:

[1] Boyd, S., & Vandenberghe, L. (2004). Convex optimization. Cambridge university press.
[2] AWS. Load Balancing, User Guide. Amazon. Online.
https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/elb-ug.pdf.
[3] Iqbal, M. A., Saltz, J. H., & Bokhart, S. H. (1986). Performance tradeoffs in static and dynamic load balancing strategies.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ondřej Vaněk, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **05.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Ondřej Vaněk, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Acknowledgements

I would first like to thank my thesis advisor Ondřej Vaněk, whose ofice's door was always open whenever I ran into a trouble spot or had a question about my research or writing.
I would like to also thank to my colleagues in Blindspot Solutions and especially to Petr Eichler, whose valuable comment suggestions on my paper gave me an inspiration to improve the quality of the assignment and helped me to overcome various challenges I faced during the development.
Finally, I must express my very profound gratitude to my parents and to my better half for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.
This accomplishment would not have been possible without them.

Thank you.

Lukáš Forst

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 13, 2019 .....

Czech Technical University in Prague
Faculty of Electrical Engineering

Abstrakt

abstract cs

Klíčová slova vyvažování zátěže, optimalizační algoritmy, Kotlin, Ktor, Docker

Abstract

abstract en

Keywords load balancing, optimization algorithms, Kotlin, Ktor, Docker

Contents

Chapter 1

Introduction

Optimization algorithms and solutions build on them are widely used in current manufacturing industry to reduce production costs. With more and more production automatization, optimization algorithms can manage and schedule whole factories with maximum available efficiency.

Complexity of optimization problems could be huge and therefore performance requirements are sometimes not easily satisfiable. Using one powerful instance of optimization algorithm in cloud seems like a solution for problems with smaller complexity, but what if we have multiple huge problems where each is performance demanding? Of course, we can create multiple instances, but that would be expensive and not well manageable and scalable since adding another instances manually requires some time and it is not much flexible. Another disadvantage of this approach is the fact, that optimization algorithm is not running 100% of time and thus resources allocated by this algorithm are unused while other algorithm instances could be potentially overwhelmed. Also paying for unused hardware is wasting money and optimization algorithms are supposed to save money.

Now imagine having two completely different problems that each requires its own application which visualises data and optimization algorithm to compute some kind of plan, this algorithm can be generic enough to operate on both domains with same code base, but it requires a lot of performance resources. If we use monolithic architecture of both applications, we would have same code in two applications, but what is even worse, we would need two powerful machines to run our applications. As previously mentioned, these two machines would not be using their power whole time and would be mainly idle.

What if one application runs only few minutes a day, but needs that power to complete tasks in time? A lot of resources would be wasted if it has its own server, but using not powerful server would lead to increasing duration of ongoing tasks which is something we do not want.

In this paper I would like to introduce **load balancer** specifically devel-

oped for optimization algorithms which could potentially minimize resources wasting and increase performance using correct utilization distribution across multiple instances of optimization algorithms.

1.1 Thesis goals

This thesis extensively studies and describes state of the art algorithms used for computational task scheduling and load balancing in section 2.1. It also brings overview of various optimization problems, techniques and algorithms that are being used in real-life situations to solve diverse industries problems in section 2.2. The complex problem of computational task scheduling is formalized in chapter 3.

Although the main goal is to develop fully functional scheduling system, mainly because of estimated complexity, this thesis focuses only on system's core - scheduling and load balancing for optimization algorithms in heterogenous distributed environment.

Solution design including final algorithm is proposed in chapter 4. In the following chapter (5), thesis describes resulting implementation of the load balancing system and also gives overview of its architecture in the section 5.1. Scheduler is then evaluated using multiple simulations in section 6.2.

To achieve ultimate goal, having fully functional scheduling and load balancing system, future necessary steps are outlined in section 7.1.

Chapter 2

State of the art

2.1 Load Balancing

Load balancing is technique for a division of processing work in the distributed environment of execution units [1] with aim to deliver faster service with higher efficiency. It improves the distribution of workloads across the whole environment and thus balances resources usage while maximizing throughput and minimizing response time. Load balancer is typically either dedicated *hardware device* or *software program*.

A **hardware** load balancer is a dedicated hardware device which distributes network traffic across a cluster of servers[25]. These devices are used mainly in the data centers to ensure equal distribution of traffic between the application servers. Main benefit of using hardware load balancer is zero balancing overhead on the host machines, because all decisions are made on dedicated hardware specially developed for such tasks.

A **software** load balancer is a program operating on the application server with the same aim as hardware load balancer. Main advantage of the software load balancing is that it can be heavily customized and deployed to its own server. This paper will discuss only software load balancing approach.

In general, software load balancing algorithms can be classified as either *static* or *dynamic*.

2.1.1 Static Load Balancing

Static load balancing is an approach where system information are provided a priori and load balancer does not use performance information about

[1] In general, execution unit can be CPU, network links, storage devices or other devices, in this paper *execution unit* or also referred as *execution node* or as *host* is a computer executing assigned job

execution node [2], to make distribution decisions. The performance possibilities and the load of the execution point (or node) are not taken in account when decision - where to execute current task - is being made, because load-balancing decisions are made at compile time. When a decision is made, no other interaction with executing node, regarding the current task, is being made. In other words, once the load is allocated to the execution node, it cannot be transferred to another node. Static load balancing method is to reduce the overall execution time of a concurrent program while minimizing the communication delays[28]. The main advantage of static load balancing methods is mainly the fact, that there is minimal communication delay between system nodes and therefore execution overhead is minimized to almost zero. For that reason is static load balancing mainly used in the fields, where server response is crucial such as serving a web page.Also the implementation of some static load balancing algorithm is straightforwards, since the used methods are very simple.

The main disadvantage of static load balancing is that it does not take in account current state of the system, when making decision. This could potentially lead to performance issues in the whole system because some nodes can be overloaded although others are not working at all.

Another drawback of this approach is that hardware resources are allocated only once in the execution time. Since optimization jobs are very heterogeneous, they sometimes have different power requirements during the execution. For example *TASP*[3] uses only one thread when creating feasible plan in the first algorithm iteration - this task relays only on single core performance. However, when first iteration is completed, all following can be done by multiple threads, therefore it could be useful to execute first iteration on a machine with better single core performance and then transfer algorithm into machine focused on multiple threads execution. This is something that can not be done while using static load balancing.
Following static load balancing algorithms are commonly used.

First Alive

First alive or also called *Central Manager* algorithm uses the concept of a primary server and backup servers[19]. All tasks are scheduled to be executed on primary server unless the primary server is down. Then the load will be forwarded to first backup server. This algorithm has almost zero level of inner process communication, which leads to better performance when there are lots of smaller tasks.

[2]Execution node - Server executing task which is being scheduled by load balancer. In our case, this task is solving optimization problem by solver.

[3]**Task and Asset Scheduling Platform** - proprietary optimization software developed by Blindspot Solutions, described in section 2.2.2

Round Robin

Round Robin algorithm which distributes work load evenly to all nodes. It is being done in round robin order, where load is distributed to each node in circular order without any priority. Round Robin is esy to implement and as well as *First alive* algorithm has almost none inner communication overhead. This algorithm performs best when tasks have equal, or at least similar, processing time.

Weighted Round Robin

Weighted round robin algorithm maintains a weighted list of servers and forwards new connections in proportion to the weight, or preference, of each server. This algorithm uses more computation times than the round robin algorithm. However, the additional computation results in distributing the traffic more efficiently to the server that is most capable of handling the request[19].

Threshold Algorithm

Threshold algorithm - execution nodes keep private copy of the system's load, when the load state of a node exceeds a load level limit, node sends message to all remote nodes, that it is overloaded. If the local state is not overloaded then the load is allocated locally. Otherwise a remote node, that is not overloaded, is selected and if no such node exists it is also allocated locally. This algorithm has low inter process communication and large number of local process allocations. The later reduces the overhead of remote process allocation and the overhead of remote memory access, which leads to performance improvements[27].

Least Connections

Least connections algorithm maintains a record of active server connections and forward a new connection to the server with the least number of active connections[19]. This can be generally useful while having many concurrent requests, that can be dispatched quickly.

Randomized Algorithm

Randomized algorithm uses random selection of the execution node without having any information about it.

2.1.2 Dynamic Load Balancing

Unlike static load balancing algorithms, dynamic algorithms use runtime state information to more informative decisions while distributing the jobs.

They monitor changes on the system work load and take it in account when decision, where to execute job, is being made. The process of monitoring the system is not stopped after execution job started and if circumstances change, job execution can be transferred to another system node which then proceeds with execution.

While many different load balancing algorithms have been proposed, there are four basic steps that nearly all algorithms have in common[22].

1. Monitoring workstation performance (load monitoring)

2. Exchanging this information between workstations (synchronization)

3. Calculating new distributions and making the work movement decision (rebalancing criteria)

4. Actual data movement (job migration)

Dynamic load balancing algorithms can be divided into two groups based on their control form, or in other words, where load balancing decisions are made[22].

- Centralized - a single node in the network is responsible for all load distribution

- Distributed - all nodes ale equal

While in centralized scheme are decisions made in one master workstation, in distributed scheme, the load balancing algorithm runs on all nodes and each node balances itself. Each of this approach has its own ups and downs, centralized scheme can be potential performance bottleneck since it relies on one system node, on the other hand distributed scheme has communication overhead, because it requires broadcast communication between all algorithm instances.

The main advantage of dynamic load balancing is that it allows changing execution node in runtime. For that reason it is possible to change hardware characteristics according to the job execution phase. For example execute initial phase of optimization algorithm on machine with powerful single core performance and then move the job to the machine with multiple, less powerful, cores to let it run in parallel.Also as a result of runtime scheduling, dynamic load balancing algorithms tend to provide a significant improvements in performance over static algorithms. However, this comes at the additional cost of collecting and maintaining load information[22]. For that reason dynamic load balancing suites better for long running tasks, which can be managed and distributed better, than for fast queries.

Dynamic load balancing strategies

There are three major parameters which usually define the strategy a specific load balancing algorithm will employ. These three parameters answer three important questions[22]:

1. Who makes the load balancing decision?
2. What information is used to make the load balancing decision?
3. Where the load balancing decision is made?

Question number 1 is answered based on whether a **sender-initiated** or **receiver-initiated** policy is employed. In *sender-initiated* policies, congested nodes attempt to move work to lightly-loaded nodes. In *receiver-initiated* policies, lightly-loaded nodes look for heavily-loaded nodes from which work may be received[22].

Question 'What information is used to make the load balancing decision¿ is answered by following policies - **global** and **local**. When algorithm uses *global* policy, the load balancer uses the performance profiles of all execution nodes connected to the network. When using *local* policy, only local [4] nodes are taken in account while creating performance profile of the system.

The last parameter - 'where the load balancing decision is made' - is answered by used control form, as mentioned previously, dynamic load balancing algorithms are divided into two groups based on their control form - **centralized** and **distributed**.

I would like to present two general dynamic load balancing algorithms - *Central Queue Algorithm* and *Local Queue Algorithm*.

Central Queue Algorithm

Central queue algorithm is based on centralized receiver-initiated load balancing strategy. It uses a cyclic FIFO queue on the main host to store new activities[5] and unfulfilled requests. New activity request is inserted into queue and here it is stored until some execution node picks it up.

Whenever a request for an activity (which is send by executing node in the case when its load has fallen bellow specified threshold) is received by the queue manager[6], it removes the first activity from the queue and sends it to the requester. If the queue is empty, the request is buffered, until a new activity is available. If a new activity arrives at the queue manager while there

[4] Workstations are usually divided into groups, in this context *local* means in the same group of workstations

[5] Activities - jobs to be executed, in our case optimization job

[6] Queue manager - central server which manages queue

are unanswered requests in the queue, the first such request is removed from the queue and the new activity is assigned to it.

When a execution node load falls under the threshold, the local load manager sends a request for a new activity to the central load manager (which manages the central system queue). The central load manager answers the request immediately if a ready activity is found in the queue, or queues the request until a new activity arrives[29].

Local Queue Algorithm

Local queue algorithms uses distributed receiver-initiated strategy.

Its main feature is, that it supports dynamic process migration.This algorithm in the first step uses static allocation of all new processes - all processes are allocated to under loaded hosts. In the second step the process migration is initiated by a host when its load falls under predefined threshold[7]. In such case, the execution node attempts to get several processes from remote hosts. It randomly sends requests with the number of local ready processes to remote load managers. When a load manager receives such a request, it compares the local number of ready processes with the received number. If the former is greater than the latter, then some of the running processes are transferred to the requester and an affirmative confirmation with the number of processes transferred is returned.[29]

Local queue algorithm is distributed load balancing algorithm where each execution node requests a new activity when it is under loaded. The main advantage of using such algorithm is the fact, that there is no central point, where all requests are managed and distributed to another segments of system. For that reason is this particular algorithm copes and performs well under an increased or expanding workload.

2.1.3 Load Balancing for Optimization Algorithms

In general, load balancing algorithms don't use information about what exactly is being executed on the execution nodes. This is because they are working mainly on the network layer and thus don't need that information. Also, they are mainly designed to be generic - to be used with any system and to be suitable for every environment. From the load balancer point of view, everything behind load balancing layer of the system is a black box.

Because there is no knowledge about the algorithms operating on the execution nodes, load balancing algorithm can not make fully informed decision about the job execution. However, this paper focus on the load balancing and execution scheduling of optimization algorithms, therefore, unlike generic load balancing solutions, proposed load balancer **have** the information about execution algorithms on the host machine and thus load balancing decision

[7] This threshold can be defined by the user and it is an input for the algorithm

are more informed. More informed load balancing decisions could potentially lead to better performance and costs reduction as well as greater capacity of whole system.

Since load balancer is aware of algorithms running on the hosts, it can take in account an execution criteria which can be specified (such as execution time) or at least estimated (how much memory will be needed according to the domain size) in advance to make even more informed balancing decision when scheduling the job execution. This is also the main difference between the generally used and existing load balancing software and a solution proposed in this paper.

2.2 Optimization Algorithms

In this section thesis presents various optimization techniques and existing solvers implementations, that can be used to solve such optimization problems.

2.2.1 Linear Optimization

Linear optimization (or linear programming) is a method to achieve the best outcome in a mathematical modelwhose requirements are represented by linear relationships. The algorithms are widely utilized in company management, such as planning, production, transportation, technology and other issues.

The main benefit of linear optimization is that it provides the best possible solution, because optimization algorithms are guaranteed to provide optimal solution. Although almost everything can be represented as linear problem, linear programming solvers could be unable to provide solution since, in the most cases, computation time grows exponentially. Even though there are solvers that are able to provide ϵ (partial) solution,this solution can be (and in most cases is) unusable, because is is not optimal at all.

There are plenty of linear programming solvers available. I would like to highlight following two optimization kits.

GLPK

GNU - *GNU Linear Programming Kit* is a software package intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI *C* and organized in the form of a callable library[21]. Although originally is GLPK written in *C* programming language, there is an independent project, which provides Java-based interface for execution of GLPK via Java Native Interface. [8]

Google OR-Tools

Google OR-Tools - OR-Tools is an open source software suite for optimization, tuned for tackling the world's toughest problems in vehicle routing, flows, integer and linear programming, and constraint programming[17]. Tools contain *Glop* which is Google's custom linear solver. One of the greatest advantages of Google OR-Tools is great API supporting multiple programming languages - *C++, Python, C#* and *Java*.

[8] Java Native Interface - Interface provided by Java platform to run and integrate non-Java language libraries

2.2.2 Heuristic algorithms

Heuristics algorithms (or HA) are designed to solve optimization problems fasterand more efficient fashion than Linear Optimization methods by using different kinds of heuristics and metaheuristics. In exchange for that, algorithms sacrifice optimality, accuracy, precision, and completeness. Thus solution provided by HA is not guaranteed to be optimal. HA are often used to solve various types of NP-complete problems such as Vehicle Routing, Task Assignment, Job Scheduling or Traveling Salesmen Problem. Heuristic algorithms are most often employed when approximate solutions are sufficient and exact solutions are necessarily computationally expensive[26].

The main advantage of heuristic algorithms is that they provide quick feasible solution. Because the implementation of HA is easier than LP and they provide at least feasible solution for optimization problems, they are solving, they are widely used in organizations that face such optimization problems. The main downside of HA is the fact, that they can't guarantee that the found solution is the optimal one.

I would like to mention two implementations of heuristics algorithms - OptaPlanner and TASP.

OptaPlanner

OptaPlanner is an open source generic heuristics based constraint solver. It is designed to solve optimization problems such as Vehicle Routing, Agenda Scheduling etc. While solving optimization task, it combines and uses various optimization heuristics and metaheuristics such as Tabu Search or Simulated Annealing.

OptaPlanner is written in pure Java and runs on JVM, therefore it can be used as Java library.

TASP

Task and Asset Scheduling Platform is a lightweight framework developed by Blindspot Solutions[12] designed to solve a large variety of optimization and scheduling problems from the area of logistics, workforce management, manufacturing, planning and others. It contains a modular, efficient planning engine utilizing latest optimization algorithms. TASP is delivered as a software library to be used through its API in applications which require powerful scheduling capabilities.

It is written in Kotlin which runs on JVM, therefore it can be easily used as library to any JVM based project.

2.2.3 Selected algorithms

I decided to use one linear solver and one heuristic algorithm to test load balancing server. This will provide us heterogeneous environment for distinguish optimization tasks as well as different demands on performance. While choosing suitable solvers I was looking mainly at possibility running on JVM and their API as well as at their suitability for my paper. For final testing I selected **GLPK** as linear solver, mainly because it is widely used linear optimization kit and because of it's convenient Java interface.

As a representative of heuristics algorithms I selected **TASP** because of it's great scalability, Kotlin interface and because I have already worked with it and I'm familiar with multiple TASP implementations.

Chapter 3

Problem formalization

The problem with implementation of optimization algorithms in applications is that their performance requirements are quite high and are fully utilized only while working. Optimization algorithm is not running all the time and for that reason hardware resources are mainly unused. These unused resources could be potentially used by another instance of algorithm or can be shutdown completely to reduce hosting costs.

Also adding more time to job execution does not always bring better solution but it certainly costs more. Therefore proposed load balancer must be able to stop execution when solution value is not getting better in comparison with scheduling costs.

3.1 Formal definition

Whole definition implies that it can be represented as Integer Linear Programming (ILP). However, it is not entirely formalized, it is only mechanical work to transcript all implications into equations. For that reason, and better readability, I left the implications.

3.1.1 Variables Definition

Following indexes, inputs and variables are used in the optimization criteria.

Indexes

- j - index used to identify something related to the execution job, in real world this is most likely job id, located in the right upper corner - x^j, set of all jobs in the system is represented by J
- r - index used to identify resources, written in the left upper corner - ${}^r x$, set of resources is represented by R

- t - right bottom index represents time - x_t

Input

Input which is specified before executing optimization job by the user outside of the system. When new execution job is requested - this is done by the user, or client application, following data should be provided.

- D^j - maximal duration of the job execution which cannot be exceeded
- P^j - maximal used resources cost per job, or in other words highest possible price paid for the job execution which cannot be exceeded

There are also constants defined before load balancer start up.

- ${}^r c$ - cost of the using particular resources per one time unit

Each of the previously mentioned variable must be non-negative.

Program Output

Apart from result of the underlying optimization algorithm, following data are returned to user after successful job execution.

- T^j - time taken, duration of the actual job execution
- C^j - resource costs, how much money was actually paid for the job execution

Variables

- v_t^j - value(i.e. cost of the scheduled plan) of the job j at the time t Value is greater than zero and it is non-increasing during the time.

$$\forall t, j : v_t^j \geq v_{t+1}^j > 0 \tag{3.1}$$

  It is non-increasing because optimization algorithms return always best found solution, so when worse solution, than currently best one, is found, returned is still the best solution found.

- ${}^r x_t^j$ - represents assignment of the resources r at time t to job j

$$ {}^r x_t^j = \{0, 1\} \tag{3.2}$$

  Each x is either 1 = indexed resources are assigned to the job at given time or 0 = given combination does not have assignment. We assume that each job has only one such assignment at one time, which effectively means that this job is executed on the single computation node. This is defined by following constraint:

$$\forall j, t : \sum_{r \in R} {}^r x_t^j \leq 1 \tag{3.3}$$

- ${}^{r}\Delta_t^j$ - enhancement of the value v with resources r on the job j per time t.

$$ {}^{r}\Delta_t^j = {}^{r}|v_t^j - v_{t-1}^j| \cdot {}^{r}x_t^j \tag{3.4} $$

  It is improvement of the solution value v which can be achieved by using resources r at time t. This value is always non-negative since optimization algorithm always stores best found solution, and therefore $\forall j, t, r : \ {}^{r}\Delta_t^j \geq 0$. ${}^{r}x_t^j$ ensures that only resources, that are actually used, are taken in account.

- S_t^j - reward for improving solution value until time t per job j. Accumulation of enhancements ${}^{r}\Delta_t^j$ through all resources r and time units t.

$$ S_t^j = \sum_{t} \sum_{r \in R} {}^{r}\Delta_t^j \tag{3.5} $$

- C_t^j - defines how much execution of job j cost from the beginning of the execution until time t. Sum of all allocated resources for their time for the particular job.

$$ C_t^j = \sum_{t} \sum_{r \in R} {}^{r}c \cdot {}^{r}x_t^j \tag{3.6} $$

  Because C_t^j is defined as sum and ${}^{r}c$ is non-negative, it is true that $\forall j, t : \ C_{t+1}^j \geq C_t^j$. Input of the program specifies maximal cost paid for job execution as a P^j, therefore it must be enforced by the system that this cost will not be exceeded. This constraint can be defined as follows.

$$ \forall t, j : \ P^j \leq C_t^j \implies \sum_{t+1}^{\infty} \sum_{r \in R} {}^{r}x_t^j = 0 \tag{3.7} $$

  Which effectively means that when cost of job execution C_t^j has reached maximal defined cost P^j, no resources can be assigned to this job.

- t - time, it is not only index but also variable, there are also constraints regarding time - since client application can specify deadline to job D^j, there must be additional constraint for job execution in a matter of resources assignment.

$$ \forall t, j : \ D^j \leq t \implies \sum_{t+1}^{\infty} \sum_{r \in R} {}^{r}x_t^j = 0 \tag{3.8} $$

  When maximal time is exceeded, no additional resources can be assigned to the job execution, which could be defined by following constraint.

3.1.2 Resources reconfiguration

System should be capable of changing resources assignment per job in the runtime. This will help to distribute performance according to the current nodes load across whole network. Unfortunately, it is not always possible to reconfigure resources assignment while scheduling is being performed. Therefore there must be at least one time unit, between different resources assignment, where no resources are assigned to the job. In other words, if resources reconfiguration is triggered in time t then $\sum_{r \in R} {}^{r}x_t^j = 0$. This can be formalized as:

$$\sum_{r \in R} {}^{r}x_t^j = 1 \implies \forall r \in R : {}^{r}x_t^j - {}^{r}x_{t+1}^j \geq 0 \tag{3.9}$$

3.1.3 Optimization criteria

The main goal of the system is to minimize outcome value of the underlining optimization algorithm and at the same time to minimize cost of used resources. We can optimize single job or sum of outcomes from all jobs in the system at once. First approach provides possibility to control and optimize outcome of a particular job, which is an advantage for single client (job owner), but it does not necessarily means that it is optimal for the whole system and vice versa. Optimization criteria for the single job at particular time t is then maximization of weighted difference between the value enhancement reward and cost paid for the enhancement, which can be described by following equation.

$$\max crit_t^j = \alpha S_t^j - (1 - \alpha) C_t^j \qquad 0 \leq \alpha \leq 1 \tag{3.10}$$

For optimization of system-wide resources and costs, all jobs execution optimization is then defined like a weighted sum of all rewards per jobs lowered by sum of all resources costs across the set of all jobs.

$$\max crit_t = \alpha \sum_{j \in J} S_t^j - (1 - \alpha) \sum_{j \in J} C_t^j \qquad 0 \leq \alpha \leq 1 \tag{3.11}$$

Based on the previous equation, it is possible to define time independent optimization criterion.

$$\max crit = \sum_{t}^{\infty} crit_t \tag{3.12}$$

3.2 Resulting problem

Taken together, optimization algorithms load balancing problem results in computational task scheduling optimization problem. Proposed formal definition (3.1) implies that problem is in fact, integer linear problem with job execution cost as its main optimization criteria. These kind of problems are solvable by various types of linear programming solvers. Linear optimization and its characteristics are outlined in previous section 2.2.1.

However, apart from load balancing decisions, the problem brings another challenges in time series prediction during the job executions. In the following chapter 4, I would like to describe how I managed to solve the main optimization problem and related time series prediction.

Chapter 4

Solution design

Presented problem can be solved using many possible approaches, I have decided to use mathematic optimization for running algorithms values predictions and heuristic algorithm for load balancing decisions.

4.1 Algorithm value prediction

To have the most informed decisions while creating load balancing decisions, algorithm creating these decisions should be able to estimate impact of assigned resources to the job-value development. Unfortunately, it is not possible to exactly predict the value function, because for instance optimization algorithms based on heuristics and metaheuristics are stochastic and therefore there is no guarantee that they will eventually find a better solution or how fast they will be able to do that. That said, the only thing that is guaranteed is that the job value function over time period is monotonically non-increasing.

However, since it is not necessary to have exact prediction values, we can roughly approximate the job value function as the hyperbola function. Using this approximation means, that value prediction algorithm is trying to fit time series data into hyperbola.

Moreover, it is not possible to use the time as x axis, because it would not be possible to extract information about how adding more available performance will modify the predicted job value function. For that reason, it is better to use the number of iterations, that optimization algorithm performed, instead of time unit.

4.1.1 Hyperbola time series fitting

The generic equation for expressing hyperbola function on two dimensional graph is following.

$$a + \frac{b}{x + c} = y \tag{4.1}$$

Where a, b, c are parameters and x, y are axis values. For usage in computer algorithm, it is better to transform the equation into the form, where there is no division. Apart from the better performance in favor of multiplication[20], it is possible that the state when $x = -c$ occurs. If there is the division, the resulting value will be infinity, which breaks next algorithm iterations.

Instead, it is better to use following form, where there is no division and 0 as a result is not a problem for the following algorithm iterations.

$$ax + ac + b - yc = yx \tag{4.2}$$

Since the algorithm value prediction should be computed as fast as possible and it is not necessary to have the precise value, I decided to use mathematics optimization approach transforming the problem into non-linear least squares problem.

Non-linear least squares problems are often solved by the iteration algorithms such as Gauss-Newton algorithm[18] and Levenberg–Marquardt algorithm[23]. The second mentioned non-linear algorithm is more robust than the Gauss-Newton, because of the Marquardt parameter[23], which means that in many cases it finds a solution even if it starts very far off the final minimum.

Using Levenberg–Marquardt algorithm means, that it is necessary to know the derivation of the function, algorithm is trying to fit in. Derivation of the used function 4.2 is then following.

$$f'(x) = (c + x, 1, a - y) \tag{4.3}$$

As the target values, are used $x \cdot y$. As the parameters, that Levenberg–Marquardt algorithm is trying to fit in are used a, b, c.

4.2 Load balancing decisions

To make most informed load balancing decisions while scheduling multiple optimization algorithms, the application uses dynamic scheduling with centralized node running load balancing algorithm.

The application also should take advantage of knowing the exact maximal time of execution thanks to the input parameter D^j defined in section 3.1.1. Thanks to this parameter, it is possible to create scheduling decisions for much larger time horizon, because algorithm can know, what workload expect in the future.

The definition formalized in section 3.1 implies that it is possible to use integer linear programming solver, because the problem itself is defined as integer linear programming problem. That is indeed possible, but after careful consideration I rather decided to use heuristic approach. The main reason for selecting this type of optimization algorithm is, that it provides suitable results during the whole runtime. This could be very handy when the time for load

balancing decisions is tight and in such case, the integer linear solver would not have enough time to provide suitable solution, because it would not be optimal at all. Also, it is easier to maintain and modify heuristic algorithm than the linear one and constraint modification in the future would be easier and more flexible.

The optimization solver, I chose for the application, was previously mentioned (2.2.2) optimization engine OptaPlanner. Unlike TASP (2.2.2), the OptaPlanner is open sourced and can be freely used for development. The implementation based on OptaPlanner is described in section 5.4.

4.3 Complete application algorithm

There are two possible ways, how to create execution plan with load balancing decisions. The first should be used when no scheduled optimization algorithm runtime data are available. Typically, when system is starting up and no job is being currently scheduled. The second aims to provide more informed decisions by using jobs runtime data with job-value function history. Thanks to this provided history, load balancing algorithm is able to take in account the prediction of future job-value function development.

Common thing for the two scheduling ways is, that they need to receive data set, convert it into inner core representation and after plan creation send the scheduled data to the client.

The final algorithm is then following, the two phases are described below.

```
1 jobsDomain ← gather jobs from waiting queue;
2 plan ← execute initial initial plan creation on jobsDomain;
3 while job to schedule exists do
4 |   waitingJobs ← gather jobs from waiting queue;
5 |   jobsExecutionData ← gather runtime information about jobs
  |     runtime;
6 |   plan ← enhance plan with jobsExecutionData and
  |     waitingJobs ∩ jobsDomain;
7 end
```

4.3.1 Initial plan creation

In the initial plan creation phase, there are no data to begin with. Therefore the prediction functionality is not used. Also, the predictions dependents constraints in core OptaPlanner wrapper implementation are skipped.

The following algorithm steps are used when the initial plan creation is

executed.

Input: jobs, scheduling properties
1 **jobsToSchedule** ← convert given jobs to inner core representation;
2 **plan** ← start OptaPlanner scheduling with **jobsToSchedule**;
3 convert created **plan** to output data;
Output: Time schedule for each job

4.3.2 Plan enhancement

During the plan enhancement scheduling stage are used all available information for more informed decisions.

Following pseudocode describes behavior of the program when the plan enhancement phase is triggered. The obsolete data are jobs, that were already scheduled and executed, or are due their scheduling history unschedulable (their D^j or P^j exceeded).

Input: jobs, scheduling properties, job history
1 **jobsDomain** ← filter obsolete data from jobs;
2 **jobsToSchedule** ← convert **jobsDomain** to inner core representation;
3 **predictions** ← create predictions of job-value function based on job history data;
4 **plan** ← start OptaPlanner scheduling with **jobsToSchedule** and **predictions**;
5 convert created **plan** to output data;
Output: Time schedule for each job

Chapter 5

Implementation

In this chapter I would like to present the technologies that were used while implementing the previously described system. During the development, base package of the application was named `OLB`, which is an acronym for **O**ptimization **L**oad **B**alancer. In the following pages, the developed application is called this way.

The implementation itself was focused mainly on load balancing algorithm and on system's core. What is not part of this paper, and belongs to future work (described in section 7.1), is an execution module, which would be able to perform the decisions made by the core algorithm presented and implemented in this thesis.

5.1 Architecture

Although this paper focuses only on scheduling system's core, the architecture keeps in mind future work on whole infrastructure. Therefore us the system's architecture based on the idea of cooperating microservices, where each service has control over specific part of the infrastructure.

Microservice architecture is an software design architectural style that structures an application as a collection of loosely coupled services that are organized around system's business capabilities[24] and are independently deployable with enabled continuous delivery[14]. Microservice architectural design also helps to system's better horizontal scalability by using multiple instances of one microservice and orchestration module.

5.1.1 Architecture scheme

Scheme 5.1 visualizes only system's core architecture. However, whole design keeps in mind future infrastructure development proposed in section 7.1. Implementation itself was developed accordingly and used technologies and techniques are described in following sections.

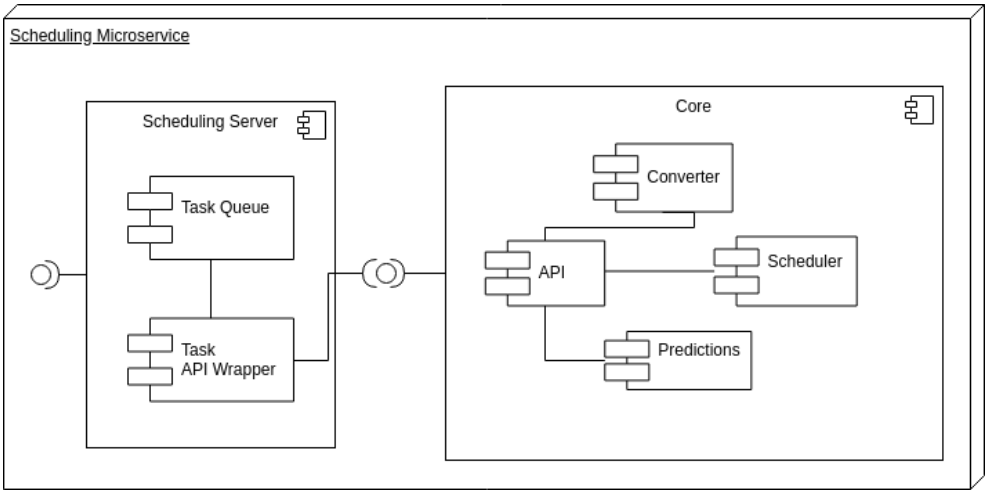

Figure 5.1: Microservice architecture with scheduling core

Core component, which is responsible for scheduling and load balancing, is composed of API, converter, predictions and scheduler module.

- *API module* implements common core interface `OlbCoreApi` and is responsible for scheduling requests handling. Actual implementation of API interface is class `OlbCoreApiImpl`.

- *Converter module* is used to convert received input data into inner scheduler data representation and then back to common data transfer objects. This converter is implemented as a class `InputToDomainConverter`.

- *Scheduler module* contains constraints, evaluator and scheduling system based on OptaPlanner solution (described in section 5.4). Scheduler module consist of multiple packages, `constraints` - containing all constrains for scheduling algorithm, `domain` with defined scheduling domain for OptaPlanner, `evaluation` which includes evaluator calculating planning score and `solver` package with factory initializing OptaPlanner scheduling core.

Scheduling server provides HTTP API access to the core and serves as microservice base. Server module is based on Ktor framework (described in section 5.2.4) that runs under the hood and provides HTTP functionality.

In the current implementation, server API accepts binary serialized data transfer objects instead of common JSON or XML. This is because of number of interfaces and loosely coupled data objects, that are being used in the application. Migration to JSON technology is addressed in future work in section 7.1.

Simulations architecture

Simulations module (project module `simulations`) is designed as another microservice to simulate future load balancing system's behavior. Following figure 5.2 shows architecture of the module.

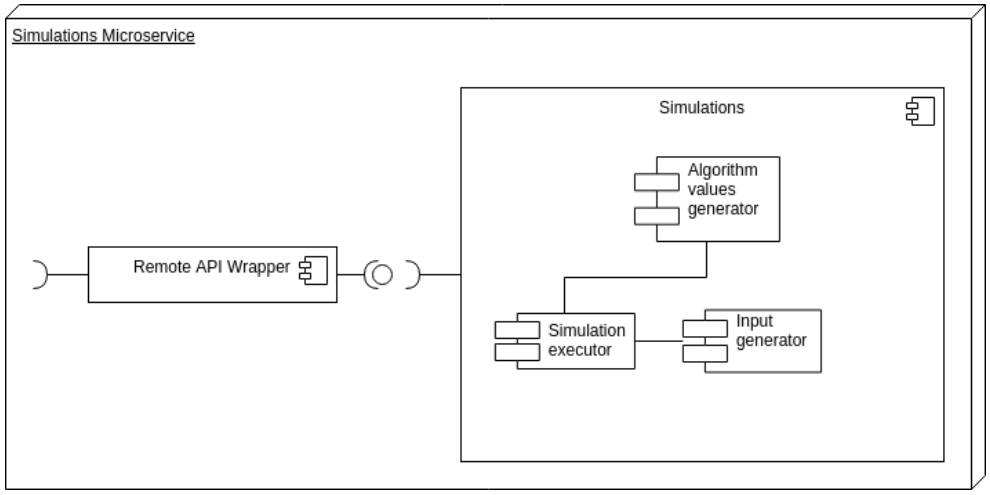

Figure 5.2: Simulations module scheme

Input data are generated by input generator module, which can be found for example in `DomainBuilder` class. Algorithm values are parsed from file in class `DataParser` and corresponding model is created on top of them in `JobWithHistoryFactory`.

Remote API wrapper is used to create connection between remote microservice, with scheduling API, and simulation. For the simulation module, the connection seems to be synchronous. This is because there is a blocking queue used to store and answer the calls between these two microservices. Thanks to this solution, simulations can be run in microservices mode as well as locally without any additional effort needed. Remote scheduling solution is implemented in project module `remote−scheduler`. Local simulations can be found for example in `OnePlanningRoundMain` or `ExecutionConfiguration` classes.

5.2 Development stack

The development stack is based on the Java platform, targeting primarily JVM 11[9] but including backwards compatibility with JVM 8. However, the traditional Java platform programming language Java was not used.

[9]Java Virtual Machine - runtime environment for Java byte code

5.2.1 Programming Language

The OLB[10]is not bound to the single technology, which could limit the development stack, and for that reason, I had a free choice while making the decision about programming language used for the OLB implementation.

OLB is programmed in the next generation programming language **Kotlin**. This cross-platform, statically typed, general-purpose programming language is developed by JetBrains[3]. Kotlin is 100% interoperable with Java because it uses JVM as its runtime and it is compiled to the Java Bytecode. Apart from Java Bytecode, it can be also compiled to JavaScript or native code.[3] The main advantage of Kotlin is its strong and aggressive type inference, meaning that for the most of time, it is not necessary to specify used data type since Kotlin compiler is able to infer it from the context.[3] It results to concise language syntax and therefore to the faster development in general.

Another great advantage is Kotlin's *null safety*. Kotlin compiler distinguishes between non-nullable types and nullable types and enforces *null checks*[11] when the object has nullable data type. This feature effectively leads to less problems in code (also called *bugs*) and drastically reduces *Null Pointer Exceptions*[12] during the runtime.

5.2.2 Build environment

The application uses Gradle as its build automation system. It was chosen mainly because its incrementally build system, that works by tracking input and output of tasks, including files changes tracking, and only running tasks, that are necessary and thus reducing the time necessary to build the project. Also, it processes only these files, that were changed between tasks execution. Another reason I choose Gradle was, that it is preferred build system for Kotlin.

To build the application using only Gradle, it is necessary to have installed at least JVM 8. There are pre-prepared Gradle wrapper (*gradlew*) scripts, that are able to build the application without having the Gradle installed on the local machine.

However, the preferred approach to build the application is to use Docker and build the application to the Docker image, which can be then run inside the Docker container.

[10]Optimization Load Balancer, name of the application
[11]Check whether the object being used has not null value
[12]Exception raised when code access reference that has null value

Docker build environment

To keep build clean and reusable on almost every operating system and machine setup I decided to use **multistage Docker build**[13] which uses different base docker images for the build and for the run phase. Since OLB targets JVM 11 environment and uses Gradle as its build system, *gradle:5.4.0-jdk11-slim* is used as base image for build stage. This image contains all necessary Gradle build tools while having smaller size than common Gradle Docker image. Even smaller (in terms of size) are *alpine* based docker images. Alpine is smallest possible Linux core, which is widely used in the wide range of Docker base images. Alpine is focused on the smallest possible size of the image, while having all necessary tools build in. Unfortunately, there were (at the time of development) no official JVM 11 alpine images since there is no official stable OpenJDK[14] 11 build for Alpine Linux.

5.2.3 Runtime environment

There are multiple ways, how to start and run the application on the local machine.

- As a container inside Docker system - **preferred**
- Locally on JVM 11
- Locally on the older JVM (but at least JVM 8)

The preferred runtime environment is Docker system, where application image runs inside the created docker container, this is described in the next subsection. Although this is preferred execution approach, there are few other approaches, how to start and run the application.

It is possible to start the application locally (without Docker environment) by having JVM installed directly on the machine. Published application setup targets JVM 11, therefore for the successful application execution there must be present latest version of JVM 11.

However, it is also possible to run the application on the previous versions of the JVM up to JVM 8. Using this way of application execution means, that it must be build directly under local JVM 8 using the Gradle build system.

Docker runtime environment

The build application files are copied from the Docker build stage to the Docker runtime stage. As the runtime base image in multistage build was used

[13] Docker is a technology that performs operating-system-level virtualization, meaning that it uses hosts operating system kernel

[14] Open-source implementation of the Java Platform, Standard Edition

openjdk:11-jre-slim image, because it is official OpenJDK 11 Docker image and therefore it is declared as stable.

Because there was used *gradle application plugin* while building the application, startup scripts were generated by the Gradle. These scripts are then used to start the application itself inside the Docker container.

When starting the whole application, multiple services must be started up. Therefore, because of the containerized environment, where containers can not access each other, multiple containers must be started and virtual network connecting them together must be created. This process can be automated using Docker Compose.

Docker Compose

Docker Compose[1] is an application for defining, running and managing multi-container Docker applications. It automatically creates Docker networks as well as Docker volumes. With writing down the definition of multiple Docker applications to the one Docker Compose configuration file, it is possible to create powerful micro services architecture, which can be build or started using single command.

Thanks to the created Docker networks, containers can communicate with each other using Docker Compose service names, therefore they do not need to know specific IP address they are running on.

Docker Compose is used in the implementation of OLB since it is designed with micro services architecture in mind. There are two services - Scheduling server and Scheduling client. Scheduling server provides ability to schedule process execution on the various computers and contains all core algorithms. Scheduling client is an example application which uses ability of scheduling server. There are implemented various simulations, which are being executed by scheduling client.

5.2.4 Framework

Because of the overall micro services architecture of the project, some kind of web framework was needed. There are many Java based web frameworks, that could be used. I would like to present two of them - *Spring Boot*, which is traditional and widely used web framework for all kind of Java web applications and *Ktor* - relatively new, lightweight Kotlin framework build upon the Kotlin Coroutines[15].

[15] Way of asynchronous or non-blocking programming that generalize subroutines for non-preemptive multitasking with using suspended/resumed task execution

Spring Boot

Spring Boot[10] is an open source Java Spring-powered web framework. It takes an opinionated view of the Spring platform, meaning that Spring Boot automatically configure Spring and 3rd party libraries whenever possible, and therefore enables usage of it to wider audience. It is highly dependent on the starter templates feature which provide pre-prepared templates for various types of web applications. This for example allows user to start with already working we server and thus simplify the start of the application development[9]. Spring Boot contains comprehensive infrastructure support for developing enterprise monolith web applications as well as micro services[9].

Ktor

Ktor[5] is an open source web framework for building asynchronous servers and clients in connected systems such as web applications and http services. It designed for quickly building web applications with minimal effort and it doesn't impose a lot of technology constraints such as logging, persistent, serializing, dependency injection etc.[4] It is developed by the same company as Kotlin is, JetBrains.

The final decision was to use **Ktor** as the web framework, mainly because of its very light implementation and native Kotlin support. Also, for such project, the features of Spring would not be fully utilized and therefore the complexity of Spring could potentially slowed down the entire application.

Because Ktor by default does not contain any dependency injection framework, I decided to use lightweight DI[16] framework *Koin*[2]. This framework is written in Kotlin and have its own DSL[17] for the dependency specification, which is very handy for the medium sized project.

Route discovery library

The default way, how to create a HTTP endpoint (Ktor calls them *routes*), which can handle HTTP requests is registering it within the `Application` context. The `Application` context is accessible by its instance that is given to user when the Ktor is being started. This means that no *route* can be registered without using instance of `Application`.

During the development of the application and using Ktor framework, I decided that this way of registering routes was not something I would like to be using, mainly because it did not allowed to have simple class serving only as a route without having to inject *Application* instance. Also, since routes must be registered during the application startup, using the new class for each route

[16] Dependency Injection
[17] Domain Specific Language

would mean to create instance of the class and calling method to register the routes manually. Another reason I did not like the Ktor default approach was that I personally prefer to inject class dependencies using construct injection instead of using setters injection.

- Constructor dependency injection - using constructor of the class to set all instances of the objects, that class uses. The main advantage is that the instance of class is always in a valid state, because it has all dependencies resolved during the instance creation.

- Setter dependency injection - the dependent objects are provided by the setter methods. This gives the freedom to manipulate the state of the dependency references at any time. However, it is possible to use the instance without setting the dependencies which could lead to the undefined behavior or to the *Null Pointer Exceptions*

To solve this `Application` instance dependency and to enable constructor dependency injection approach, I decided to implement simple library which would solve this issue for me. I came up with different way how to register various types of application's routes using the annotations, reflection and dependency injection strategy.

Preconditions for successful usage of the library are following:

- *Koin*[2] - dependency injection framework which is used for resolving dependencies needed in the routes

- *Reflection library*[8] - library used for runtime lookup for classes with specific annotation

- Using the `@Route` annotation on the class that is meant to be route, the class has to also inherit from `RouteBase`

- Registering all necessary routes dependencies in *Koin* modules during the application startup

- Provide base package name where routes are placed.

Following algorithm is used to find and register all routes used in the project.

Input: Package name, where all routes are stored
routes ← find all classes annotated as `@Route` in provided package name;
for ***route*** *in* ***routes*** **do**
    dependencies ← obtain dependencies needed for creating instance of **route**;
    routeInstance ← create instance of **route** using **dependencies**;
    register **routeInstance** in instance of `Application` ;
end
Output: All routes are registered and ready to use

The implementation of the simple route is then following:

```
@Route
class HelloRoute(sr: Service) : RouteBase("hello") {
    init {
        route {
            get {
                call.respond(sr.sayHello())
            }
        }
    }
}
```

The route is automatically instantiated and registered by the Routes discovery library. The programmer does not need handle it by himself.

For the library startup I designed a builder class using fluent builder pattern `ApplicationDependencyBuilder`. Usage of this class can be found in the `ServerStarter.kt`.

5.3 Algorithms values prediction

The implementation of Levenberg–Marquardt algorithm is not the aim of this paper, for that reason the application uses Java compatible library, which contains implementation Levenberg–Marquardt algorithm and provides way, how to use it.

OLB uses two different libraries Apache Commons Mathematics Library[11] and Mathematical Finance Library[13]. The both libraries have custom wrapper which implements `HyperbolicRegression` abstract class and they can be exchanged in the application when decided. The reason, there are two different implementations and two different libraries, is that their performance can differ in distinguish scenarios. By default, the Mathematical Finance

Library is used, because it provides better results in runtime predictions[18]. The implementations can be found as `ApacheHyperbolicRegression` class for Apache Commons Mathematics Library and `FinMathRegression` class for Mathematical Finance Library.

5.4 Load balancing decisions with OptaPlanner

Scheduling system implementation uses OptaPlanner library core in version `7.19.0.Final` [6] and it is crucial part of the `core` module and application itself.

5.4.1 Scheduling Algorithm

The OptaPlanner scheduling itself has two main phases. *Construction heuristics*, that tries to build initial solution in a finite length of time. This partial solution is not always feasible, but it finds it fast and then leaves it to next phase. *Local search* with metaheuristics that can enhance the partial solution found in previous phase.

OptaPlanner contains various types of construction heuristics (i.e. *first fit*, *weakest fit* and *strongest fit*) as well as local search metaheuristics such as *hill climbing*, *tabu search* and *simulated annealing*. As the best combination of construction heuristics and local search proved to be *first fit* with *tabu search*.

The First Fit algorithm cycles through all the planning entities, initializing one planning entity at a time. It assigns the planning entity to the best available planning value, taking the already initialized planning entities into account. It terminates when all entities have been initialized[7].

The Tabu Search metaheuristics search is based on local search optimization method and enhances it by worse step strategy, when at each step worsening moves can be accepted if no improving move is available. In addition, prohibitions are introduced to discourage the search from coming back to previously-visited solutions[16].

5.4.2 Implementation

Whole scheduling implementation can be found in module `core`. Moreover, constraints are placed in package `pw.forst.olb.core.constraints`. OptaPlanner related implementation is placed inside `pw.forst.olb.core.domain` and plan solution evaluator in `pw.forst.olb.core.evaluation`.

All custom constraints should implement `CompletePlanEvaluation` or `PlanEvaluation` interface. That way, they can be used in custom score evaluator, which is then used to generate score of the given plan. The constrains

[18] There is a Python code, which can provide graphs from values predictions, please see `plot_predictions.py` and `pw.forst.olb.simulation.prediction` package.

are stored in collections of previously mentioned interfaces and evaluated at once, when plan is submitted. Thanks to this solution, additional constrains can be added or removed very easily.

As an input for the scheduling algorithm interface `SchedulingProperties` is used. This interface defines properties necessary for the scheduling infrastructure such as `maxTimePlanningSpend` which defines how long can the application run the scheduling or `cores`, describing how many cores in the computer can algorithm utilize by spawning scheduling threads.

The OptaPlanner scheduler instance is created by custom factory implementation `OptaPlannerSolverFactory` and uses base configuration defined in `solverConfiguration.xml` file.

Chapter 6

Experiments

To extensively test the application, there are two modules, that contain simulation codes used for testing - `simulation` and `remote−scheduler` modules. Simulation engine and all simulation use cases are located in simulation module. The second mentioned module contains server, which runs simulations on remote API. This is especially useful when testing whole scheduling environment for example in Docker Compose network and it is as closest as possible to real life environment, which should be running in micro services architecture.

6.1 Optimization algorithms data

For the proper testing environment, the real runtime data of some optimization algorithm were needed. I decided to use TASP (mentioned in subsection 2.2.2) as an example heuristic algorithm which execution could be potentially scheduled by the instance of optimization load balancer. I did not implement new TASP instance, instead I used instances from Stochastic Dynamic Vehicle Routing Problem master's thesis by Petr Eichler[15].

These instances solve real life vehicle routing problem and mainly for that reason are ideal for testing purposes. I slightly modified the code from thesis for observation purposes and added time measuring functionality, which tracked time between the algorithm's iterations and the current job value in each iteration.

In overall, I executed and measured 56 different instances of TASP. The measurement can be found in `jobs-data/input` folder in the project and they are being used in the simulations, where the simulation module randomly selects for each job one file with runtime data and assigns it to the job, that is being scheduled. The data are then effectively used as input data for the scheduling algorithm.

6.2 Simulations

Simulation module, particular simulation scenarios, scenarios execution

Chapter 7

Conclusion

This thesis focused mainly on the core of the proposed load balancing system and outlined the way, how to make more informative decisions while scheduling optimization algorithms execution in heterogenous system.

The solution with 6767 lines of Kotlin code[19] proved, that it is possible to use it as load balancing algorithm for optimization algorithms. Although the core part of the scheduling server for optimization algorithm was developed in this thesis, there is lot of work being left.

7.1 Future work

The main missing part is an execution module, which would transform the created plan into physical actions performed in the infrastructure like running, stopping and moving the jobs between the physical execution nodes.

Infrastructure development

As soon as the previously mentioned execution module is implemented, the application can be deployed into real life environment and properly tested. Proposition is to make optimization algorithms running as Docker containers and the execution module would operate with Docker machines, which would be running on the execution nodes. In this way, the assigned resources would be very easily changed on one execution node. Migration to the next execution node would not be problem as well, since containers could be wrapped and transported through the network.

Another infrastructure related missing feature in current solution is full REST API. Currently only binary serialization is supported. In the future, full REST with JSON as its transport format should be supported.

[19] Lines provided by `find . −name "*.kt"| xargs cat | wc −l` bash command in root folder of the project

Documentation and unit testing

Another thing, that must be improved, is documentation as well as unit testing of the application. Unfortunately, the current lack of code documentation could discourage from future development of the system. As for the unit tests, only crucial parts of application such as extension functions and prediction module are at least partially unit tested. The goal is to have documented and tested whole core module, which is responsible for the scheduling itself and public APIs of server modules.

Routes discovery library

Routes discovery library was very handy during the development of application's server parts and I believe that this way of routes registration in Ktor would suit to many developers as well.

Therefore I would like to refactor it from the base project and create open source project which will ensure future library development. I would like to also make it more generic, because right know, it depends on specific Ktor and Koin version. Although Ktor dependency is necessary since it is library developed specifically for Ktor, Koin should be replaced by generic way, how to obtain dependencies for routes.

I believe in future usage and development of the library and I hope, that there will be many future developers, that will agree with me.

Extension functions

During the application development, I created and tested many Kotlin extension functions. These functions are mainly not domain specific and for that reason I decided that it would be fine to publish them as well as Routes discovery library. These extensions could be useful when starting new project, because they are able to perform many operations in single line of Kotlin code.

Bibliography

[1] Docker compose reference. https://docs.docker.com/compose/overview/. [Online; accessed 7-May-2019].

[2] Koin github repository. https://github.com/InsertKoinIO/koin/tree/50929af636d1956a45882b795a29ace00eeac49d. [Online; accessed 7-May-2019].

[3] Kotlin reference. https://kotlinlang.org/docs/reference/. [Online; accessed 7-May-2019].

[4] Spring boot github reference. https://api.ktor.io/1.1.5/, . [Online; accessed 7-May-2019].

[5] Ktor web. https://ktor.io/, . [Online; accessed 7-May-2019].

[6] Optaplanner documentation. https://docs.optaplanner.org/7.6.0.Final/optaplanner-docs/html_single/, . [Online; accessed 11-May-2019].

[7] Construction heuristics. https://docs.optaplanner.org/7.6.0.Final/optaplanner-docs/html_single/#constructionHeuristics, . [Online; accessed 11-May-2019].

[8] Reflection library github repository. https://github.com/ronmamo/reflections/tree/084cf4a759a06d88e88753ac00397478c2e0ed52. [Online; accessed 7-May-2019].

[9] Spring boot github reference. https://github.com/spring-projects/spring-boot/tree/5aeb31700df8e15d1aa625b987ecca93385a7c2c, . [Online; accessed 7-May-2019].

[10] Spring boot reference. https://spring.io/projects/spring-boot/, . [Online; accessed 7-May-2019].

[11] Apache commons mathematics library. https://commons.apache.org/proper/commons-math, .

[12] Blindspot solutions s.r.o. http://www.blindspot.ai, .

[13] Mathematical finance library. https://github.com/finmath/finmath-lib, .

[14] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52, 2016.

[15] P. Eichler. Stochastic dynamic vehicle routing problem. Master's thesis, Czech Technical University in Prague, Czech Republic, 2018.

[16] F. Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3): 190–206, 1989.

[17] Google. About or-tools. https://developers.google.com/optimization/. [Online; accessed 16-January-2019].

[18] S. Gratton, A. S. Lawless, and N. K. Nichols. Approximate gauss–newton methods for nonlinear least squares problems. *SIAM Journal on Optimization*, 18(1):106–132, 2007.

[19] IBM. Algorithms for making load-balancing decisions. https://www.ibm.com/support/knowledgecenter/SS9H2Y_7.7.0/com.ibm.dp.doc/lbg_algorithms.html. [Online; accessed 29-January-2019].

[20] J.-A. LeFevre and J. Morris. More on the relation between division and multiplication in simple arithmetic: Evidence for mediation of division solutions via multiplication. *Memory & Cognition*, 27(5):803–812, Sep 1999. ISSN 1532-5946. doi: 10.3758/BF03198533. URL https://doi.org/10.3758/BF03198533.

[21] A. Makhorin. Glpk (gnu linear programming kit). https://www.gnu.org/software/glpk/. [Online; accessed 16-January-2019].

[22] Malik, Shahzad. Dynamic load balancing in a network of workstations. *95.515 F Research Report*, 2000.

[23] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[24] D. Namiot and M. Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27, 2014.

[25] A. Networks. Hardware load balancer. https://avinetworks.com/glossary/hardware-load-balancer/. [Online; accessed 30-January-2019].

[26] Papanikolaou, A. *A Holistic Approach to Ship Design: Volume 1: Optimisation of Ship Design and Operation for Life Cycle*. Springer International Publishing, 2018. ISBN 9783030028107. URL https://books.google.cz/books?id=xYJ-DwAAQBAJ. 296-301.

[27] A. G. Payal Beniwal. A comparative study of static and dynamic load balancing algorithms. *International Journal of Advance Research in Computer Science and Management Studies*. [Online; accessed 29-January-2019].

[28] D. S. K. Ramesh Prajapati, Dushyantsinh Rathod. Comparison of static and dynamic load balancing in grid computing. *International Journal For Technological Research In Engineering*, 2015.

[29] S. Sharma, S. Singh, and M. Sharma. Performance analysis of load balancing algorithms. *World Academy of Science, Engineering and Technology*, 38(3):269–272, 2008.