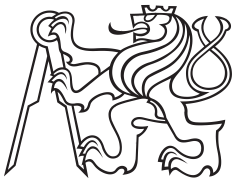

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Unassisted project report

Lukáš Forst

Supervisor: Ondřej Vaněk, Ph.D.
January 2019

Contents

Chapter 1

Introduction

Whole text does not seems to be right, maybe I will need to rewrite it.

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

Chapter 2

Problem definition

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 compared with scheduling costs.

2.1 Formal definition

- $T_{\max}$ - maximal optimization job execution time provided by user and specified before execution started
- T - actual optimization job execution time, when no execution time optimization is being used $T = T_{\max}$
- RC - *Resource Costs* - all hardware costs used for executing optimization job by some algorithm

$$RC = \sum_{i=0}^{T} RC_i \tag{2.1}$$

don't know how to say that - costs that you actually pay for hardware

- RC_t - *Resource Costs* in time t - accumulated costs from beginning of execution to time t

$$RC_t = \sum_{i=0}^{t-1} RC_i \tag{2.2}$$

- $RC_{\max}$ - maximal resource costs specified by user in advance

$$RC_{\max} \geq RC \tag{2.3}$$

- V - *Solution Value* - value of the found solution, since this paper focus on cost optimization, *Solution Value* is cost of found solution

$$V = min\{V_t\}, \quad t = 0 \dots T \tag{2.4}$$

- V_t - *Solution Value* in time t - best solution provided by algorithm since the beginning of the job execution until time t

Then load balancer optimizes following function

$$min\{\alpha RC + \beta V \mid \alpha, \beta \in \mathbb{R}\} \tag{2.5}$$

Where α and β are coefficients that are balancing RC and V.

2.1.1 Load Balancer Requirements

2.2 Motivation to solve it

Chapter 3

Technical Background

3.1 Optimization Algorithms

This work does not contain any own algorithm implementation for generic optimization problems, instead I would like to use pre-prepared and already implemented optimization solver.
We have many options how to solve optimization problems, I would like to present two of them - linear optimization and heuristics algorithms.

This is really shity introduction

3.1.1 Linear Optimization

Linear optimization (or linear programming) is a method to achieve the best outcome in a mathematical model whose requirements are represented by linear relationships.[Wik19] The algorithms are widely utilized in company management, such as planning, production, transportation, technology and other issues.

Advantages and disadvantages of linear optimization

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.

Existing solutions

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

GLPK - *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.[Mak]
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. [1]

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.[web] 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.*

3.1.2 Heuristic algorithms

Heuristics algorithms are designed to solve optimization problems faster and 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.[Pap18]

Advantages and disadvantages of heuristic algorithms

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.

Existing solutions

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. OptaPlanner is written in pure Java and runs on JVM, therefore it can be used as Java library. While solving optimization task, it combines and uses various optimization heuristics and

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

metaheuristics such as Tabu Search or Simulated Annealing.

TASP - *Task and Asset Scheduling Platform* is a lightweight framework developed by Blindspot Solutions 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 and runs on JVM.

3.1.3 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.

do I have to mention that I'm working for Blindspot?

3.2 Load Balancing

There will be some info about how should server balance itself.

- prioritisation - mainly done by priority queues
- handover
- instance sizing
- algorithms - following are methods used in network balancing -> probably can't be used because we need to manage scheduling which is heavy on computer resources like CPU/RAM/IO
  - The Least Connection Method
  - The Round Robin Method
  - The Least Response Time Method

some stuff about load balancing in general

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

3.2.1 Static Load Balancing

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

the nodes [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. Then, depending on the performance and load of the nodes, balancing server decides which node will execute task. 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.
The main disadvantage of this approach is

3.2.2 Dynamic Load Balancing

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

Chapter 4

State of the art

Bibliography

[Mak] Andrew Makhorin, *Glpk (gnu linear programming kit)*, `https://www.gnu.org/software/glpk/`, [Online; accessed 16-January-2019].

[Pap18] Papanikolaou, A., *A Holistic Approach to Ship Design: Volume 1: Optimisation of Ship Design and Operation for Life Cycle*, Springer International Publishing, 2018, 296-301.

[web] *About or-tools*, `https://developers.google.com/optimization/`, [Online; accessed 16-January-2019].

[Wik19] Wikipedia contributors, *Linear programming — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Linear_programming&oldid=878407127`, 2019, [Online; accessed 16-January-2019].