



**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

1 Introduction	1
2 Problem definition	3
2.1 Formal definition	3
2.1.1 Load Balancer Requirements .	4
2.2 Motivation to solve it	4
3 Technical Background	5
3.1 Optimization Algorithms	5
3.1.1 Linear Optimization	5
3.1.2 Heuristic algorithms	6
3.1.3 Selected algorithms.....	7
3.2 Load Balancing.....	7
3.2.1 Static Load Balancing	8
3.2.2 Dynamic Load Balancing	9
4 State of the art	11
Bibliography	13

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

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

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

$$RC = \sum_{i=0}^T RC_i \quad (2.1)$$

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

$$RC_t = \sum_{i=0}^{t-1} RC_i \quad (2.2)$$

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

$$RC_{\max} \geq RC \quad (2.3)$$

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

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

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

- 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

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

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.

This is really
shitty introduc-
tion

We have many options how to solve optimization problems, I would like to present two of them - linear optimization and heuristics algorithms.

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 it 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

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 its convenient Java interface.

As a representative of heuristics algorithms I selected **TASP** because of its 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

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

some stuff about load balancing in general

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 performance information about execution node², 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.[RP15]

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** 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** or also called *Central Manager* algorithm uses the concept of a primary server and backup servers.[IBM] All tasks will be scheduled to be executed on primary server unless the primary server is down. Then 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.
- **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 easy 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** algorithm maintains a weighted list of servers and forwards new connections in proportion to the weight, or preference,

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

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.[IBM]

- **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 under loaded node 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 improvement in performance.[PB]
- **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.[IBM]
- **Randomized algorithm** uses random selection of the execution node without having any information about it.

■ 3.2.2 Dynamic Load Balancing



Chapter 4

State of the art



Bibliography

- [IBM] IBM, *Algorithms for making load-balancing decisions*, https://www.ibm.com/support/knowledgecenter/SS9H2Y_7.0/com.ibm.dp.doc/lbg_algorithms.html, [Online; accessed 29-January-2019].
- [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.
- [PB] Atul Garg 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].
- [RP15] Dr. Samrat Khanna Ramesh Prajapati, Dushyantsinh Rathod, *Comparison of static and dynamic load balancing in grid computing*, International Journal For Technological Research In Engineering (2015).
- [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].