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

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

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

- 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

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

- 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.

Chapter 3

State of the art

3.1 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[Net]. 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*.

3.1.1 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 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

[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

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

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 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[IBM]. 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.

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.

[3] **Task and Asset Scheduling Platform** - proprietary optimization software developed by Blindspot Solutions, described in3.2.2

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[IBM].

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[PB].

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[IBM]. 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.

3.1.2 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[Mal00].

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[Mal00].

- 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[Mal00]. 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[Mal00]:

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[Mal00].

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 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[SSS08].

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

[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

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

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.[SSS08]

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.

3.1.3 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 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.

3.2 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.

3.2.1 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.

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[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. [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[Goo]. 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

3.2.2 Heuristic algorithms

Heuristics algorithms (or HA) 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].

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 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.

3.2.3 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.

Bibliography

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

[IBM] 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].

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

[Mal00] Malik, Shahzad, *Dynamic load balancing in a network of workstations*, 95.515 F Research Report (2000).

[Net] AVI Networks, *Hardware load balancer*, https://avinetworks.com/glossary/hardware-load-balancer/, [Online; accessed 30-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).

[SSS08] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma, *Performance analysis of load balancing algorithms*, World Academy of Science, Engineering and Technology **38** (2008), no. 3, 269–272.

[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].