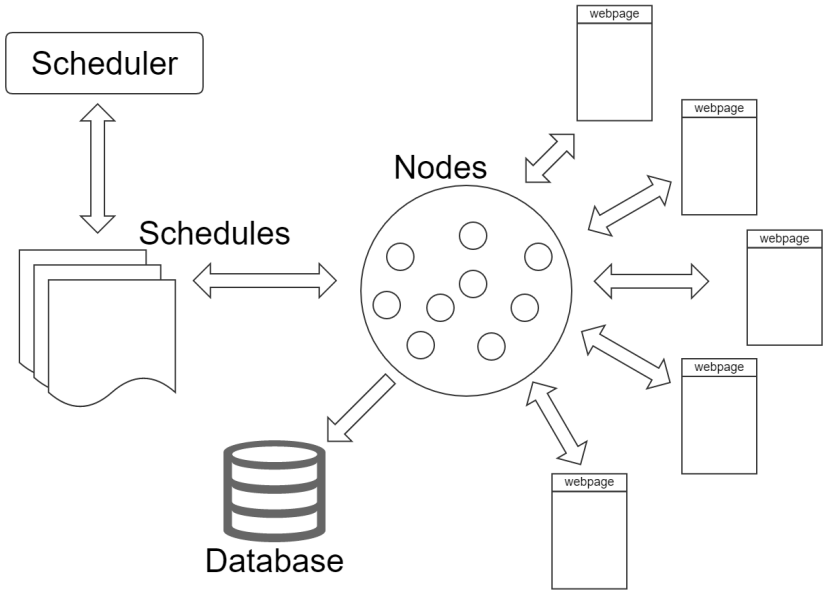
CHALMERS
UNIVERSITY OF TECHNOLOGY



Optimization of a Scheduler for a Web Scraping System

Master's thesis in Systems, Control and Mechatronics

Daniel Martinsson

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Master's thesis 2019:NN

Optimization of a Scheduler for a Web Scraping System

DANIEL MARTINSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Optimization of a Scheduler for a Web Scraping System

DANIEL MARTINSSON

Supervisor: Kristofer Bengtsson, Department of Electrical Engineering
Examiner: Knut Åkesson, Department of Electrical Engineering

Master's Thesis 2019:NN
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of the final system with the new scheduler implemented

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2019

Optimization of a Scheduler for a Web Scraping System
DANIEL MARTINSSON
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This master's thesis describes the improvements made to the scheduling of jobs in an existing web scraping system. Web scraping is a process where a program is collecting all info from a website ("scraping"), which in this case is used to collect ads from a range of Swedish media sites.
The characteristics of the system were examined to help present a new model of how to use an optimization approach to the scheduling, which would also help to make the scheduling automatic instead of the manual process used before. The problem consisted of a number of jobs that needed to be scheduled to a number of nodes, which is quite familiar with a job shop scheduling problem. The jobs were also of different importance, which needed to be implemented in a better way since it was difficult to ensure that high priority sites were scraped more often than others.
The resulting scheduler improved the system in several aspects, by automating the scheduling process and also making it easier to understand at what frequency certain sites where scraped. The input of the site descriptions (the priority and the time to scrape the site) became one key to improving the scheduling even further, since sites with similar descriptions could be grouped together to speed up the scheduling. Some different tests were done where the input was changed and the scheduling time was then exponentially increasing with relatively small changes. This then led to the conclusion that the new scheduler improved the system but that some restrictions had to be applied to the input to ensure a certain performance.

Keywords: Optimization, Scheduling, Web scraping, Job shop scheduling, CP.

Contents

1 Introduction

Most people is getting in contact with online advertisements every day. From reading the news and blogs to buying things in webshops, ads are always present. They come in many forms and sizes, such as covering the whole page as an overlay, in between paragraphs or in the sidebar where links to other sites also appear.

The web is accessible for advertising companies, since there are so many different kind of sites to advertise on and much easier to personalize the ads. However, due to the diversity of advertisement channels and complex personalization settings, it is challenging to evaluate how valuable it is in comparison to standard TV-commercials as discussed in [1].

Mathew [2] brings up that the web is a more diversified media to advertise in, giving the effect that there will be a large difference in pricing for different advertisements, which can be a problem for companies e.g. since it might be hard to really know how many their ads reach. This then raises the interest to really analyze and understand online ads and to what reach an online ad can be compared to a TV-commercial.

How often a certain ad appear or to what extent it reaches a certain target group has a big value to the media agencies that tries give a correct price when selling their advertisement spots to other companies.

Media agencies are also looking for new business opportunities by examining which companies that are displayed on certain sites. If e.g. a specific car company is not being displayed on one of the biggest car sites, while at the same time all of its competitors are shown on that site there is a chance that the car company is interested in expanding their advertisements to that web sites as well.

Ads are often generated randomly, changed everyday and sometimes also related to your browsing history. An example system of how to generate targeted ads is described in [3]. The challenge is therefore to monitor what ads that show up on a specific site and at what frequency.

For examining ads in news papers and tv-channels there are currently humans doing the work with reading through/watching and writing down what they encounter. This is time consuming as it is and would not be possible for the internet since it is

more dynamic.
Since the ads (often displayed as an image or video) already is digital, a way to monitor could instead be that a program examines the sites and reports the result. In this way the speed could be higher and more sites could be processed in the same time period.
Ads at on-line news websites and other media sites is today complicated to monitor. Small videos, flash and images are often easy to spot for a human but can be tricky to distinguish from the website by a program.
By opening the websites and "scrape" it for all of this information is one approach, and are currently being used today as presented in this case study [4]. Scraping in this case means, taking all the content of a certain site and store it down locally, like images, javascripts and HTML-files. The whole website is then said to be "scraped" of its info.
A programatic approach is for example to send only a HTML request to the server hosting the webpage. This will then return a HTML file of the page, where images and clips are represented with their respective tags and addresses to where they're hosted. If the webpage contains javascript these will only showed as text in the HTML file and will not be executed, meaning that any files associated with the javascript are not possible to find.
Since many the sites are different in amount of content and also type of content (images and videos), the time to scrape a certain site can differ. With different execution times but with the same priority of the sites, challenges occur when the same number of executions wants to be done for each site every day. To manage to keep the number of executions the same for all sites one could look into in what order the sites are placed in the sequence of execution. When using several web scrapers, how the sites are distributed amongst them can also be looked at, which then brings a new level of difficulty.
When some of the sites are at a greater importance (e.g. more commonly visited sites) their priority can be increased. This then means that the frequency in which a specific site is scraped needs to be increased. The difference in frequency between the sites becomes the way to respond to the difference in priority.

When scraping in general, the way tasks can have a specific order in which they need to be executed or they can be totally independent of each other, meaning that they can be executed in any order. The web scraping-system do not have any need for putting the tasks in a specific order, which is seen as easier to schedule than to having certain sites being scraped in a specific sequence.

1.1 Purpose

The main purpose of this thesis is to examine an already existing web scraping system and try to improve it by developing an automatic scheduler. Since the system today needs a lot of manual work, a newer system will be suggested to be implemented.
First the specifics of the current system will be looked at. Then an optimization algorithm for the scheduling will be created with the help of the literature on the subject. This will create a schedule for all the sites in the system that can be used in the actual system.

1.2 Goal

First, this thesis examined the current web scraping system and determined the characteristics. The possibilities for improvement was then be clearly stated alongside the weaknesses of the system in which improvement is not possible.
An improved model of the web scraping system was the next goal to be achieved. By using the possible improvements described earlier together with different aspects of scheduling a better solution was able to be produced.
How will a mathematical model look like that solves the optimization problem that the scheduling creates? How will different inputs change the output of the scheduler? Will a scheduler be able to produce a result within reasonable time, so that it can replace the current system?

1.3 Demands

The scheduling system developed in this thesis has to have a short execution time, meaning not longer than 10 minutes for a new schedule to be produced. All of the sites execution times needs to be known beforehand, meaning that no job task has an indeterminate execution time.
The deadline must always be bigger and can't be the same as a specific site's execution time.

1.4 Research questions

How will a mathematical model look like that solves the optimization problem that the scheduling creates? How will different inputs change the output of the scheduler? Will a scheduler be able to produce a result within reasonable time, so that it can replace the current system?

1.5 Ethics

In the project the aim is to automate a process which has previously been done by humans, and also perform a better result then what was previously possible. In a wider view in the society this project is related to the automation of simple work tasks where humans is replaced with computers or robots, where the question can be raised about whether this creates more unemployment.
If new jobs is not created at the same pace as simple jobs disappear then this is a valid claim. An opposing thought to this is that for every simple job that is discontinued, the energy that was needed for a human to do that job can now be directed into something else which can benefit the society in another way.
By automating the process, the complexity increases and the knowledge needed to supervise the computer or robot is higher than before. This leads to that the workforce needs more education, and if that is a sustainable solution is hard to determine, since the length of the education system can't be extended continually. A need for a PhD to just get a standard job in the future might be an unsustainable situation.

The view on this project is that the positive aspects is greater then the negatives. Where the system is used today, the people that work with it is needed in other parts of the project, meaning that an exclusion of humans is a positive outcome for this project.

2 Theory

2.1 Web sites

Todays standard for creating web sites is by using HTML[1]. An HTML-file is a text file constructed of several "tags" in which other information is inserted. Tags can describe e.g. images, paragraphs of text or videos.
To retrieve these HTML files it is common to use a web browser (e.g. Chrome or Firefox), where an address to a certain server is entered in the address field and a visual representation of the web site is shown on the computer screen.
What the browser does is sending a request for a specific HTML file, which then the server responds to. The response is often the specific HTML file but can also be other things like e.g. error messages that the file does not exist.
When the HTML file is tried to be interpreted to a visual presentation of the site, new requests are sent when new resources are needed (e.g. images or external HTML files.), which also are processed by the server and responded to in the same way as the initial request. The communication protocol that is used on the internet is called HTTP[2], and was designed to provide a method for web servers to transfer HTML files to web browsers.

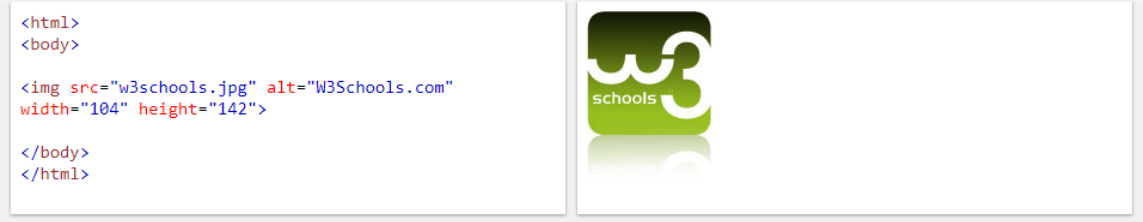


Figure 1: To the left an example of a simple HTML-file with an image-tag (img). To the right the visual representation of the HTML to the left.

As seen in figure 1 the difference between the HTML file and what is displayed is quite big. Different web browsers can interpret the HTML files differently putting

[1]HyperText Markup Language
[2]HyperText Transfer Protocol

more responsibility on the web designer of the web site to know how to have the web site being displayed equally on all web browsers.
To display e.g. the images of a web site the actual image has to be downloaded, which is done by the web browser when the web site is generated. By just opening the HTML-file in a text editor no images will de displayed or downloaded, but one can find the address to where the image can be found inside the HTML file (which is the way the web browser finds the image file).
To change the design of the web site, CSS[3] can be used or to add more functionality to a web site, javascript can be used.

2.2 Javascript

Javascript is a programming language which is a cornerstone in web sites, which is added to make the web site more interactive. Javascripts can be used e.g. to populate lists according to filters or get images in a specific order based on when they were created.
When the HTML file is opened in a web browser the javascripts that are included in the HTML file is being executed and the files associated with that downloads. To ensure that all content gets loaded, the javascript must be executed, like a browser is doing, by a javascript runtime.
By executing a javascript on the site, the scraping time increases, since it first needs to execute the script and also download all external content associated with that script.
The scraping is then relying on that a browser is opening the web site so that all content is downloaded. If only the HTML file was downloaded (which can be done by using an application that only sends HTTP requests and do not try to create a visual representation of the web site) then all of the content of the web site will not be requested and can not be measured in the scraping.

[3]Cascading Style Sheets

2.3 Scheduling

Scheduling problems are solved in a wide array of areas and is easiest described as a list of tasks that should be completed within a certain time frame. The tasks can have certain characteristics, e.g. longer execution time or demand relatively more processing power than other tasks, which then shape the scheduling problem in specific directions.

The tasks can either be of fixed time length and demand a certain amount of effort or they can have a varying time length depending on how much effort you put in. Fixed time length can be seen as easier to schedule since the time is known beforehand and does not change during the execution because of outer circumstances as is described in [5]. In the scraping problem all tasks (sites) are of fixed time length which makes it better for scheduling.

In the scraping system there are several nodes which all work independently of each other with their own unique site. They can be seen as nodes which fetches jobs from a list of available jobs. This is similar in some ways to a job-shop scheduling problem. The nodes that perform the web scrapes are in a job-shop problem the number of machines available and the sites is the jobs that are needed to be scheduled. The aim is to minimize the *makespan*, which is the total time it takes to perform all the jobs. The problem with scheduling the sites is not identical to this but is related, where the main difference is that the makespan is maybe not the most important rather than the different priorities of the sites.

Tasks can also have a priority to make sure that no important tasks get done too late in a tight schedule. This is even more important when there is not enough time to complete all the scheduled task inside the given deadline. The less important task will then be skipped and the more important tasks deadline will be met as discussed in [6] where the search for a feasible solution for a given scheduling problem is described. In this web scraping-system all of the sites needs to be scraped but there exists more important sites which is favorable to be scraped more often than less important sites. To be able to schedule as fast as possible, good scheduling algorithms are very important. To have a correct answer (in this case, the best possible schedule) is always

preferable, but if the trade off is that it takes 3 days to get there then there might be interesting looking into how to get a "close to correct" answer in a shorter time period.
In some scheduling problems the order in which the jobs are done is important but in others it does not matter at all. If the order does not matter then it can be seen as one type of simplification which can be done to help the algorithm to shorten the time period of the scheduling.
To be able to find these kinds of simplifications is often of great benefit to the scheduling and are what comes when looking at the characteristics of the given problem.

2.4 Optimization

Many problems has an element of maximizing output or minimizing loss of a certain process, which can be used for decision and allocation problems. By describing the problem by one objective function with interrelated variables that either should be maximized or minimized it can be seen as an simple way in dealing with wide variety of problems.
The variables is then directly related to the solution but are under certain constraints that limits the selection of the variable values. E.g. a storage room can never have fewer than zero boxes in store or a car might have a certain maximum speed that needs to be taken into consideration.
The complexity of the optimization lies in expressing these constraints and the objective function so that a solver can find a solution within a reasonable time.
When having jobs that should be scheduled on a node there are two possible ways to move forward in order to optimize the utilization of the nodes. One way is to look at the lowest possible number of nodes that is needed in order to cover all of the jobs during some period. The second way to go is to use all of the available nodes but instead try to schedule in a way that there will be a better robustness in the system, meaning that the time needed to finish all assigned jobs of a certain node will be as low as possible.

Since it in this project a number of nodes is already ready to do the "scraping" there is no need in minimizing the number of nodes since then some nodes might be unused. It is then better to focus on using the nodes as efficiently as possible, and by that also try to achieve a good robustness in the system.
The scheduled time must also be shorter than the shortest deadline of the jobs assigned to that particular node.
The best result in the optimization is then the solution with the biggest difference between shortest deadline and scheduled time. The node with the lowest difference is the only one considered since a good robustness is desired on all of the nodes, and all of the other nodes will then have better result then the one considered.

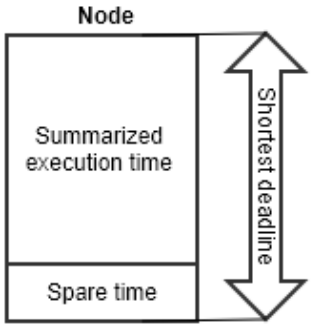


Figure 2: A graphical representation of the shortest deadline compared to the summarized execution time for a node

2.5 Linear Programming

Scheduling is a problem that arises in many different fields, and the solving it will require some form of optimization. Many different ways to achieve this exists but a common way is to describe the problem mathematically and then use linear programming(LP) to solve it which is described in [7]. LP can be used on a broad variety of problems but has also some restrictions on what it can achieve.
LP can only solve problems where the constraints are linear but the variables can take any value, all rational numbers, but it can not have variables that only takes the

form of integers. When a variable represents time which can be seen as continuous variable LP is good to use.
Integer linear programming (ILP) is related to LP, but where the variables only can take the form of integers. This is good optimizing the usage of a storage house with a certain number of storage slots for example.
Mixed integer linear programming (MILP) as described in [8], is even more complex because in the way that variables can now also be discrete, which then can be seen as a mix between the 2 already mentioned linear programming methods LP and ILP. Scheduling and packing problems are then favorable to solve with ILP since it focuses on integers. When the tasks being scheduled are atomic, there can never be a solution with a schedule consisting of fraction of tasks. This is then mostly related the ILP problem, since no continuous variables are needed.
When scheduling the sites a lot of different optimization methods are available to use. Integer Programming that needs all of its variables to be integers, is one way to go and can be related to the problem at hand since there is a number of nodes on each node that needs to be specified, in which non-integers do not exist. Its extension Mixed Integer Linear Programming also allows some of its variables to be non-integers, which could relate to a percentage of CPU usage on a node for example.

2.6 Constraint programming

To solve optimization problems, constraint programming is one technique that can be used. The problem which is described by a number of a variables are solved by writing down constraints in how the variables relate to each other which is explained in [9].
Constraint programming also uses logical operators as is also seen in imperative programming, but differs in the way that a constraint does not describe a step or a sequence of steps to execute.
The constraints that describe the problem can come from a wide area of domains such as

- boolean domains, variables are true or false, such as SAT problems

- integer domains
- linear domains, that is describing linear functions
- finite domains, where constraints are described with finite sets
- mixed domains, a combination of the domains described above

Constraint have been used before for scheduling problems, and then in combination with MILP. The result in that case was found efficient and could neglect infeasible solutions early in the execution process with the "first-fail" principle described in [10].

Constraint programming are more related to programming than linear programming and many programming languages are either built for or have a good way to support it. Linear programming is more built to be expressed and solved mathematically and is not supported to the same extent in programming languages as constraint programming.

Constraint programming has also no limitation in what on the arithmetic expressions that make up the constraints, where as linear programming is limited to a specific class of problems which is decided by the mathematical programming engine used.

2.7 Robustness

Robustness is a measure in how good a given system handles errors during run time. Errors can occur both in the way of unforeseen events to the system or that the input is not recognizable to the system.

Unforeseen events can be things like power failure or delays for maintenance for example. If the system desires integers as input but receives strings of text instead, that could be seen as an error. How good then the given system copes with these things is a measure of how robust the system is.

A schedule which is insensitive to disturbances from unforeseen events is considered a robust schedule as defined in [11]. One main thing when striving for a robust system is the ability to measure how robust a system is in comparison with other systems. Two possible ways to achieve a measure of robustness, either the difference between the best possible time and the actual time is considered or the actual time and the

deadline is considered. In the first case there is a scale between a perfect system and a bad system, but in the second case there is only two possibilities with either making the deadline or not.

2.8 Job-shop

Job-hop scheduling describes the problem where many small jobs needs to be done inside a specific time frame (makespan) with many machines that can be assigned to a job which is described in [12]. A job can be represent anything from saving files to storage or booting up a computer. A larger job can often be broken down into smaller jobs

Minimizing the makespan is the focus of job shop scheduling which also means that the goal is to finish as the given sets of jobs in the smallest possible time frame. Every job has a predefined execution time which makes it an optimization problem when the task should be fit inside the time slots available.

The scheduler will then need to fit all the jobs in to the available time slots, somewhat similar to a tetris game where the idea is to fill up empty space with blocks of different size and at the same time have as few empty slots left.

In order to solve such a problem, some restrictions are needed to be inserted to insure that the solution fulfills the specifications of the problem. Without restrictions the scheduling would still be a challenge, but it might not correspond with the actual system quite as well which is discussed in [13].

Two jobs can not be allocated to the same time slot for the same machine, meaning that all time slots are unique.

If there are certain jobs that needs to be done in a certain order, that order must stay intact through out the whole scheduling. The jobs must also be finished before the next one can start, since all time slots are atomic there is no possibility to have two jobs running at the same time on the same machine.

All deadlines needs to be met, meaning that the job needs to be finished before a certain time.

By first meeting all of these requirements the focus is then to minimize the makespan

described in [14], which is the time it takes to finish all jobs. If the requirements can not be met in the first place then there exists no solution for the given job shop problem.

3 Current system

For media agencies, a company that works with selling advertisement spots to other companies, finding business opportunities and setting correct prices is a challenge. To help out with this Kantar Sifo has a system scraping a big range of web sites and collecting the data about which ads is shown on which sites and also how often they appear.

The end product is then an application which displays the data, the amount of ads found and also at which frequency they were shown for specific sites. This is then used by media agencies to develop their business offer. This is not the only type of information they use to decide on the pricing for the ads, but one of several and it is seen as valuable so a demand for this system is seen to be maintained for some years to come, therefore it is of high interest from Kantar Sifo to develop the system further.

The system as it is has some flaws and needs a bit of attention from humans on a daily basis to operate as expected. Making changes to the system (e.g. adding new sites or nodes) is then time that is needed to be added on top of the daily maintenance, and as the system grows it becomes more and more complex to do this.

Collecting ads with the system is a very static job, and is not suited for scaling. The number of sites to scrape are approximately 250 and adding more sites will take linear time for the system to process, but exponential time in the maintenance by humans since every new site is affecting all of the previous ones. When adding a site it needs to be assigned to a specific node, and to still maintain a good quality of the system

Every site is assigned to a specific node, which gives each node a predefined list of sites. The task for each node is to execute a scrape for each of the assigned sites and then repeat the same list over and over. If few sites are assigned to a node, the total execution time of these sites will be low.

If a site is important, the only way to increase how often it is scraped is to assign it to a node with few sites. If many sites needs to be scraped often, then the number of nodes have to be increased as well. In some cases there is a specific amount of

scrapes for a specific site that needs to be done each day, which makes it possible to calculate the how often it needs to be scraped and number of required nodes.
The existing system consists of a set of nodes which are running 24 hours a day. Any problem that occur with any of the node is instantly causing a delay and loss of information.
A node are given a certain number of sites to scrape, which are all done in one sequence with no restriction on the total time scraping. If any of the nodes then stops, the sites assigned to that node would not be scraped for the entirety of the delay.
This was designed with the fact that certain sites were similar and that the web scraper on the node could then be designed a bit different for each node to match the specifics of the sites assigned to it, making the web scraper faster and smaller in size.
The system also have problems with scaling since each web scraper occupies one node, which means increasing the number of web scrapers the same amount of new nodes is needed. This is also becomes a problem when a node needs maintenance and no other node claims a bigger workload, these sites will then not be scraped for the time it takes to perform the maintenance.
The current system scrapes every site once every two hours in normal conditions. For high priority sites this is needed to be done more often, meaning that 12 scrapes per day is not enough to catch all advertisements that is displayed on the site. An advertisement can change often, or be a part of a big "ad-pool" (meaning that a random advertisement is loaded each time the site is loaded). If the ad-pool is bigger than 12 or the advertisement change every hour there is a big possibility some advertisements will not be scraped.
The aim is to scrape as many different advertisements as possible meaning that the number of scrapes for the high priority sites needs to be increased.
A representation of what the current system looks like can be seen in figure 3 in which each instance has been assigned a certain number of sites. If one instance is lost so is also the sites assigned to it.

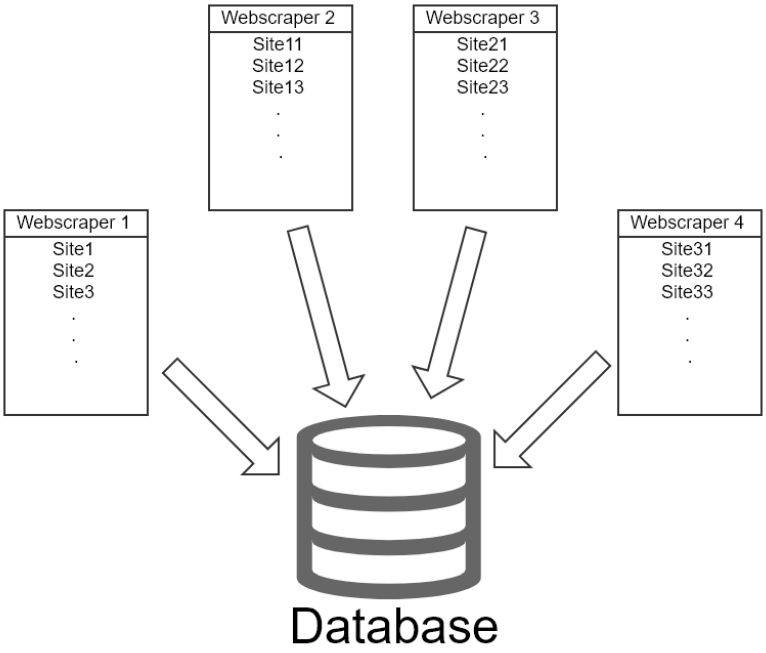


Figure 3: The current system, here shown with four webscrapers

To capture ads on websites Fiddler[4] is currently used in order to separate images from the other things downloaded when a site is loaded. Fiddler is a desktop appliction whichs monitors all HTTP and HTTPS requests that are being sent and received for the time period in which it is used, this is represented in a list with the requests that includes data such as headers, body, HTTP status code etc. .

By using multiple applications simultaneously that are using HTTP/HTTPS requests the list will be filled with requests from different sources, without any possibility to distinguish which request that originates from which source.

This means that a way to speed up the web scraper by scraping two sites at the same time by the same node is not possible since the HTTP/HTTPS requests will be mixed up with one and other, which are rendering the scrapes information unusable since the web page i originates from cannot be determined.

In order to only download the needed content certain hosts can be blocked, en-

[4]https://www.telerik.com/fiddler

abling only information from sources that is known to display advertisements to be downloaded. For that, one needs to identify these hosts manually, checking what information that has been downloaded and and then determine if its an ad or not and block all information from the same host in the future. The program can't determine what information that is ads or native images and videos. The only thing the program sees is what type of file it is and that does not very often help in deciding if its an ad or not.
Another approach is that the full web page could be saved as an image, and then a program runs an image recognition algorithm with the help of a set of previously defined ads, in order to see if any of them reappeared at the site. The speed of this execution would then depend a lot on the algorithm and also on if the program were able to learn some frequently used spots in where the website displayed its ads.
Some websites has an overlay over the web page, that has to be clicked on in order to show the full page. This makes the "screen shot"-approach harder to maintain since the overlay can change and it is also different from page to page, making it more likely that all sites needs a unique solution.

3.1 Example problem

A small example of the problem could be that 3 nodes are available and 10 sites are needed to be scraped. 1 of these sites is a high priority site and 2 other sites that take 3 times as long time as the remaining 8 sites. In this case we will have to look at the two parameters time and priority in order to get an overall scheduling.
Being a high priority site means that it needs to be scraped more often than the other sites or that it needs to be scraped within some kind of frequency time frame (e.g. every hour).
Throughout the report this example will be reused and the high priority site is called site 1 and the two sites with longer execution are called site 2 and 3 respectively. The time for scraping a site is then t and $3t$ respectively. The nodes do not differ in any way so their numbering does not mater and will be called node 1, 2 and 3. The whole example problem is visualized in figure 4.

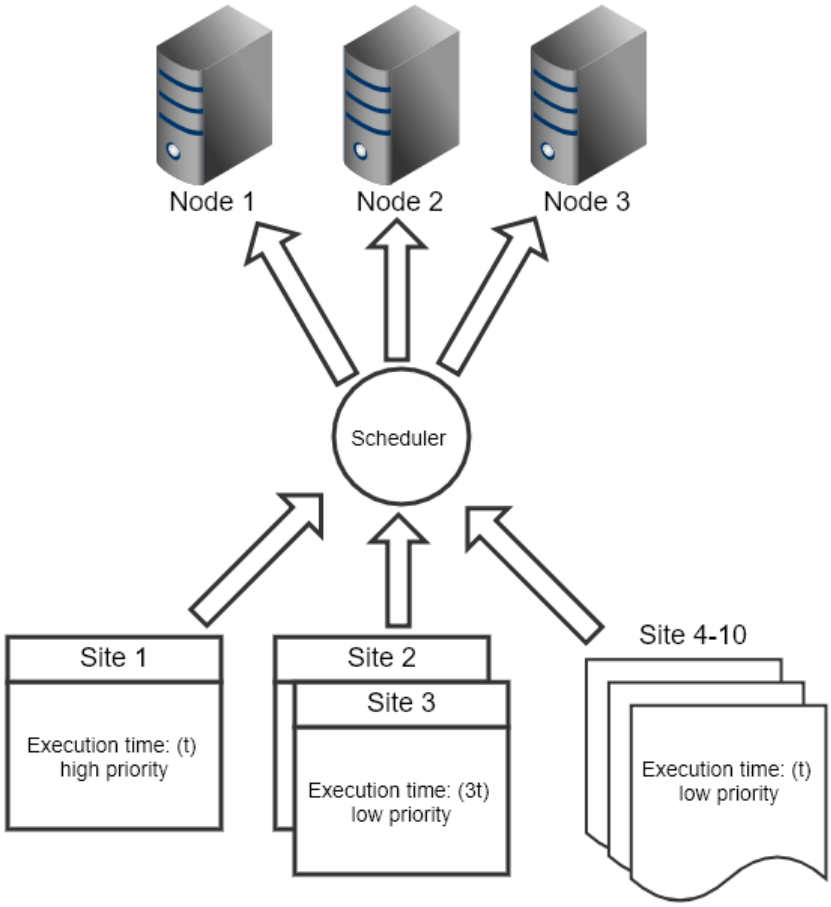


Figure 4: A graphical representation of the example problem

3.2 Scheduling example problem with current system

With the example problem explained in section 3.1, each of the sites needs to be assigned to a given node manually. The high-priority site and the 2 sites with extra long needs to be considered first, and after that the remaining 7 sites will fill up the nodes to complete the scheduling. Since they have the same priority and execution time they can be placed on any of the nodes since there is no need for order between

them.

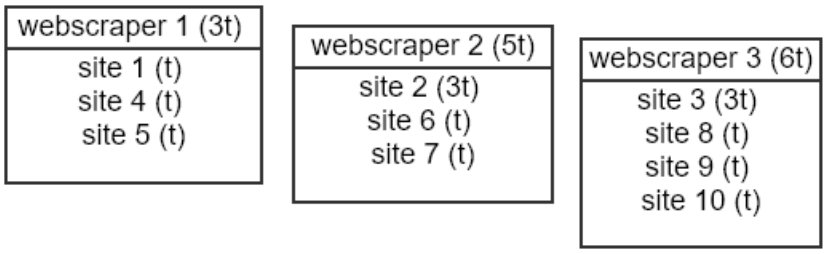


Figure 5: Sites scheduled with the existing system

Since site 1 is of higher priority it needs to be on a node with shorter execution time, which also means the frequency will be higher for that site. As seen in Figure 5, site 1 have ended up on web scraper 1, and therefore the total time to scrape the sites on that web scraper needs to be lower than the other web scrapers. Since we have no other high priority sites web scraper 1 is the only who needs to have a shorter total time.

Web scraper 1 with site 1, has 2 additional sites, which gives it a total execution time of $3t$. Site 2 and 3 which have longer execution times will then be placed on web scraper 2 and 3 respectively to try to make them as even as possible. They could also have been placed on the same web scraper, which would have made no difference in the final total times. The remaining nodes has then 5 sites to split between them which will be done as evenly as possible. Since site 4 - 10 have the same execution time there is no need to distribute them in any specific way. Site 4 and 5 is added to web scraper 1 which gives it roughly half the execution time as the other two web scrapers if site 6 - 10 is added to them. An option here could be to just have 2 sites on web scraper 1 which would give it a total time of $2t$ which then would be a third of the other web scrapers total time. But as for now we are satisfied with our first schedule. This then summarizes the execution times of the web scrapers $3t$, $5t$ and $6t$.

One remark to be made is that the high priority site (site 1) is only high prioritized

in respect to the other sites. If there is an amount of scrapes that needs to be done each day for a specific site then this is not considered.
To get this high priority site to be scraped more often it is currently scheduled manually and the execution time for the given node is calculated and then seen if it meets the given criteria that is set up for the site.
If for example site 1 is 10 times higher prioritized than the rest of the sites this solution does not give a correct result. Also changing the priority of one site does lead to a whole lot of rearranging and recalculation which has to be done manually.

3.3 Possible enhancements of current system

Automatically scheduling the scrapes would be a big benefit to this project since the earlier mentioned difference in priority. The scheduler can then make sure that each site get the amount of scrapes every day that matches its priority, which could have been overlooked if it was done manually. Another positive aspect with automatic scheduling is that if a new site was added, a computer would calculate the new schedule instead of human doing manual work to move all of the sites around between different nodes.
When determining what parameters that should be considered when scheduling, how they relate to each other according to their specific priority is important to look at. Priority is important because of the difference in importance of exported data from the system, and it can also be an easy way to follow up certain quality criterias e.g. is the highest prioritized site scraped more than double the amount of times as a low priority site.
The time consumed for a certain site is also an important factor, and these two combined has been shown to be the two most essential parameters[15]. Sorting on only the length parameter with shortest time first will cover as many sites as possible but some of the time consuming ones are maybe the most important. Sorting only on the importance will cover too few sites, the scheduling algorithm will be too greedy and the coverage will be too narrow.
The specifics of a certain site can be important but other things that also effect the

efficiency of the scheduling are also in play. Since some parameters can shift it is important to reschedule at some frequency, and that frequency can effect the result of the scheduling. The scheduling algorithm can then run on a subset of the available task in order to get a faster result and be able to do a new scheduling quicker as described in [15].

Energy consumption of the nodes used can also be an important parameter to look at. An aim should always be to produce such an application that energy consumption is kept on reasonable levels. This does effect the performance, and remaining inside all time constraints can be a challenge if the most energy efficient way is chosen as shown in the comparison in [16].

4 Conceptual system

For the new system a few points of improvement have been highlighted from the old system

- Making the scheduling automatic, to ensure no errors from humans can arise and to speed up the process.
- Separate sites from nodes, making it possible to scrape all sites even with a shifting number of nodes.
- An easy way to prioritize sites, to ensure that high priority sites will be scraped more often than lower priority sites.
- Create as much space as possible in the schedule to make it less vulnerable to unforeseen events.
- Take advantage of that many sites have the same priority and execution time.

4.1 Making scheduling automatic

To improve the performance of the system, the scheduling time needs to be reduced and the robustness of the system needs to be improved. One big factor to improve these two aspects would be to exclude humans from the process and let it be done automatic by a computer instead.
Prioritizing in the old system was only done manually and needed to be done whenever the input parameters changed (site execution times changed, number of sites increased/decreased etc.). The number of times it would be needed to be rescheduled would not change since the input parameters would change at the same frequency. The average time when scheduling would also deviate less from the average with a computer scheduling, meaning that scheduling could be counted on to perform at a certain level in a completely different way than if a human would have done it.
If the schedule were calculated by two separate humans there might be two separate ways to interpreting the priorities of the sites. With an automatic scheduling this would not be possible since so long the input parameters (execution times/priorities etc.) the schedule would not be changed.

4.2 Separate sites from nodes

As seen in the old system sites are assigned to a specific node. Each had then a list of sites which were maintained by a human which could increase or decrease in length at any time. A better version of this would be to let the sites be assigned to a list or groups of sites which then the nodes can fetch and execute in sequence. The number of lists would then match the number of nodes meaning that each node would have its unique list, but with the addition that any node could take any list of sites.
If one webscraper needs to be taken down for maintenance there is no specific sites affected since the schedule it was executing could have been assigned to any of the webscrapers. This then decreases the amount of time of manual work since a new scheduling reassigns all of the sites to a new schedule.
Relating this to the example described earlier the 10 sites would belong to specific scrape lists instead of the actual nodes. As a result the sites would maybe not change place from where they are scheduled in Figure 5 when separating sites from nodes, but the big change would arise when the number of nodes and/or sites change and a rescheduling is needed.
Another possibility could be to go even one step further and disconnect sites from separate scrape lists and schedule a single list of sites. The nodes would then fetch a new sites once they are finished with their previous one. The rescheduling would be done once the list became close to empty and would not be needed when the number of nodes changed.

4.3 Prioritize sites

In the new system being able to set some real priority requirement is needed. In that way the given schedule can be examined to see if it gives the the certain output that the system is suppose to give. It will also be easy to spot how many nodes that will be needed and also how many sites one can afford to have a higher priority on.
Each site can be given a fixed number of points which then the scheduler will try to maximize, or you can set a deadline or period time that describes how often the site needs to be scraped (once every hour, once every day etc.).

In our system we decide to go with the latter since then an exact number could be given to how often the site would be scraped, and this was also easier to understand for someone not involved in the code, they could relate to a timeframe but had more trouble with what a priority point meant in how often a site was scraped. Also if points were to be used instead, one could only affect how often in relation to each other the sites were scraped.
For this to work the scrape time of each site needs to be known, since the scheduler will use it in order to summarize the execution of each node. In this system the execution times are known beforehand which then makes it possible to summarize a set of sites execution time.
In the earlier described example problem the sites only have a subjective priority and no deadlines to meet. Deadlines needs to added if the new system should be able to schedule the sites. This must then be done by a human and is a subjective process where a higher prioritized site must be translated into some kind of deadline. To be able to a scheduling we add the following deadlines to the sites: For high priority sites the deadline is $6t$ and for low priority sites the deadline is $12t$.

4.4 Create space in the schedule

To have a robust system creating an overcapacity is a good way to go. When scheduling the sites there will be some spare time between the deadlines and execution times as long as there is enough nodes to run all sites. By knowing all of the deadlines and and execution times there is possible to measure how much spare time a certain schedule has.
When a lot of sites are scheduled on the same node there might be a lot of different priorities on the sites. If the site with the shortest deadline is considered, then it is known that all deadlines will be met for the given server. Summarizing the execution times and comparing this to the shortest deadline is one way to see how much spare time is available at the node. A bigger difference the more robust the system is.
By comparing the spare time between the nodes and trying to average them out by scheduling certain sites to certain nodes can then be as an improvement to the

robustness of the system. Comparing a certain schedule of all sites to another then be done by comparing the the smallest spare time because that is the smallest margin of error that the given schedule has.
In the example problem no deadlines are decided but with introducing them as in section 4.3 it can be scheduled and then the spare time can be examined which can be seen in section 4.6.

4.5 Sites with the same priority and execution time

In the actual problem that is described in section 3 there is approximately 250 sites that are all different from each other in terms of their content. If looked at it from a scraping perspective there are very many of these sites that are identical in aspect of execution time and priority.
When scheduling the exact order in which the sites end up is of no importance, and also which scrape list that a certain site is assigned to does not affect the performance of the system. Taking this into consideration when scheduling there will be a lot of different possible schedules that never needs to be evaluated since one know that it will not increase the performance of the system.
If the sites were grouped by execution time and priority, the groups could be seen as unique entities instead of treating each site as unique. In the actual problem there are 15 different site groups if they were grouped by execution time and priority. This is a quite big difference from the earlier mentioned 250 sites, if they would all be seen as unique entities. The same total time frame is still to be filled with sites but for the scheduler there will be a lot fewer things to choose from making it easier.
In the example problem as can be seen in figure 4, they sites are already grouped into 3 different groups as a result of their different characteristics.

4.6 Example problem with the conceptual system

As described in section 3.1 a small example version of the scheduling problem constitutes of 10 sites with 3 nodes that is available for scraping.

Making the scheduling automatic a shifting number of nodes would not be a problem, because the only thing needed would be to run the scheduler again, which has roughly the same execution time no matter the number of nodes. This also makes the sites totally decoupled from the nodes, making it possible easy to do maintenance on the nodes or even add new ones. With the example problem this is hard to measure the exact positive effects, but excluding the manual steps in scheduling is something that hopefully will bring a better stability and a better average performance from the system.

Adding the deadlines to the sites is a subjective process, since there is nothing that says that certain sites needs to be run double as often as other sites other than human opinions. If $t = 1h$ the deadlines can be interpreted as that high priority means 4 times a day and low priority 2 times a day. This is a good representation of some of the deadlines that are appearing in the actual system, and are chosed with that in mind.

The ratio between number if site groups and total number of sites is maybe not that representative of the actual problem, which is used in the example problem. The idea here was to keep the ratio below 50% so that it would show a scheduling of a number of sites which had a lot of similarities but still became grouped in a few different groups so that this also could be examined.

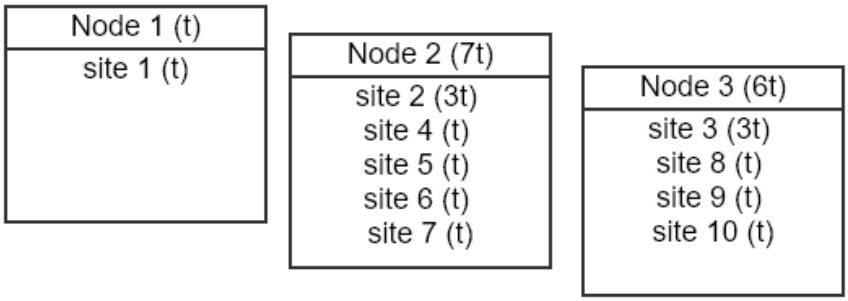


Figure 6: Finished schedule of the example problem.

When scheduling with the new system the sites end up in a schedule as can be seen

in figure 6, which can be compared with figure 5 were it has been done by a human. One big difference is the fact that site 1 now has its own server to be able to cope with its high priority. Even though node 2 and 3 are taking 7 and 6 times longer to complete its schedule it is still the best solution in regard to the rules previously stated.

4.7 Comparing the new scheduler to a general case

To be able to fully evaluate the new scheduler, a general scheduler needed to be created by which the new scheduler could be compared to. This would be a way to see if it was be worth the effort to develop the new scheduler or if a general scheduler would perform just as good for this particular problem.
In our case a greedy algorithm was decided to be used for the general scheduler. The greedy algorithm was created in such a way that the sites were assigned in a sequence and the next site was always assigned to the node which would have the highest spare time when the new site's deadline is considered (if it is lower then the current lowest deadline on the node).
Each site was compared to all the of the nodes, and the deadline of the site was compared to the already lowest deadline on the node to see which one was lower. The lower of the two was then compared to the sum of the already assigned sites to the node (giving the current sparetime if the site was added), and the site was then added to the node with the lowest sparetime.
The order of the sites in which they are compared and assigned to the nodes, were random since ordering them after either execution time or deadline would mean that the sites were prioritized even further than just the deadline, since being assigned early would mean more influence on which site being assigned to that node later which could be seen as a high priority for that site.

5 The new system

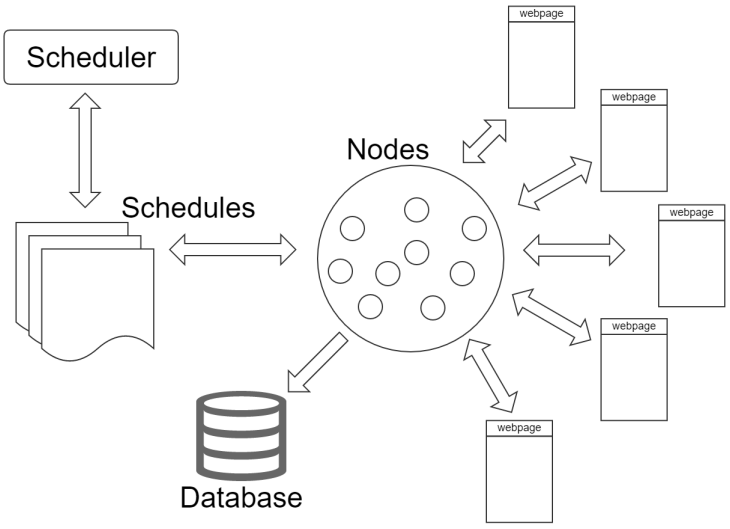


Figure 7: The new system with a scheduler added

The existing webscraper is currently written in *C#* in which the optimizer had to be integrated. One could write the optimizer directly in *C#*, but it was found to be easier if created in a program and language designed for optimization, which then would be integrated to the already existing system.
The optimizer was designed in the program *MiniZinc*[5]. In *MiniZinc* a range of different optimizers can be used, each serving a different purpose.
When coding in MiniZinc a list describe the system using two separate groups, inputs and variables. Inputs are given by the user to the system and are not changed during the optimization and variables are empty beforehand and are changed during the process to find an answer as good as possible.
The optimizer which was used in this occasion was G12 (which can be read about in [17]) which is an open source constraint programming platform, developed for large scale industrial problems.

[5]http://www.minizinc.org/

5.1 The finished model

The problem consist of a set of k sites $S = \{s_i\}_{i=1}^{k}$, which all should be scheduled on a set of m nodes $N = \{n_i\}_{i=i}^{m}$. For each site s there is a corresponding execution time e_s and deadline d_s. The sites can only be scheduled to one node, and since the nodes are identical the site can be scheduled to any of the nodes.
To model this each node has it own set of sites S_n, in which the site with the shortest deadline d_n acts as the deadline for the node as well. The site with the shortest deadline is the on with the highest priority and is therefore considered first.
The model has the following decision variables:

- x_{sn} is a Boolean variable that models if site s is assigned to node n.

maximize T_L subject to

$$T_L = \min T \qquad T = \{\Delta t_1 \dots \Delta t_m\} \tag{1}$$

$$\Delta t_n = d_n - \sum E_n \qquad n \in N \tag{2}$$

$$x_{sn} \rightarrow e_s \in E_n \qquad s \in S, n \in N \tag{3}$$

$$x_{sn} \wedge d_s < d_n \rightarrow d_n = d_s \qquad s \in S, n \in N \tag{4}$$

$$x_{sn} \rightarrow \neg x_{sn'} \qquad n \neq n', s \in S, n, n' \in N \tag{5}$$

If always the lowest difference T_L is made better (e.g. a task is moved to another node to change the schedule), one of the nodes with a higher difference Δt_n will be made little slower. This makes the workload more evenly distributed.

5.2 Symmetry breaking

When doing optimization the aim is to reach the best possible solution as fast as possible. When an optimizer checks for the best solution, it also needs to prove that a given solution is the best in order to if the best one has been reached or if we have a better solution that is not yet tested.
One way to come around the fact that every possible input needs to be tested, symmetry breaking can be used where similarities in the input helps the optimizer

to overlook certain inputs that is known to equal or lesser good solution.
Since many of the tasks have the same execution time and deadline, these can be grouped together to make the optimization a easier. The optimizer do not have to test all different possibilities and can instead focus on a smaller set of different execution times to fit in to the possible time slots.
This changes the problem to consist of a set of j groups of sites $G = \{g_i\}_{i=1}^{j}$ to be scheduled over a set of Nodes N. For each group g there is a corresponding execution time e_g and deadline d_g.
The model could then be rewritten with the following decision variables:

- x_{gn} is an Integer variable that models the amount of sites from group g that is scheduled on node n

maximize T_L subject to

$$T_L = \min T \qquad T = \{\Delta t_1 \dots \Delta t_m\} \tag{6}$$

$$\Delta t_n = d_n - \sum_{i=1}^{j} x_{in} * e_i \qquad n \in N \tag{7}$$

$$x_{gn} > 0 \wedge d_g < d_n \rightarrow d_n = d_g \qquad g \in G, n \in N \tag{8}$$

$$\sum_{i=0}^{m} x_{gi} = |g| \qquad \forall g \in G \tag{9}$$

A site s^g from group g will always produce the same schedule on a given node as if another site from the same group was considered. This is seen in equation 10 where one site differs but since it is from the same group the set is still the same.

$$\underbrace{\{\dots, s_x^g, \dots\}}_{S_i} = \underbrace{\{\dots, s_y^g, \dots\}}_{S_i} \tag{10}$$

Since the nodes are identical there is also no need to test a certain set of sites on a different nodes. This is also implemented in the optimizer, which makes it faster without cutting down on quality. This can be seen in equation 11 where the same set of sites is scheduled to two different nodes and are still equivalent.

$$\underbrace{\{s_1, \ldots, s_n\}}_{S_i} = \underbrace{\{s_1, \ldots, s_n\}}_{S_j} \tag{11}$$

The nodes are then assigned a certain amount of jobs from each of the groups, which is shifted through the optimization to find the optimal solution. Sites from a specific group must be scheduled the same amount of times as there are sites in that group as can be seen in equation 9.

5.3 Discussion

Assigning sites to a specific node and then summarizing the points that they are given as a way to create a good schedule could be a hard task, since how the points are set beforehand manually can play a very big role. It can even lead to that if the final schedule is not satisfactory then the points for some sites might need to be changed to get a better result.

Using deadlines instead are easier since one can precisely pinpoint the needs of priority of the full system, since you know exactly how often the site needs to be scraped and not how often it needs to be scraped in respect to other sites.

The points would be subjective and needed to be tweaked in order to have the desired schedule, but by looking at deadlines instead, an exact time limit could be set which would not be dependent of what any other sites have as a deadline. The points would all need to be in relation to all of the other sites.

Various aspects are needed to be considered in order to have the fastest and most reliable optimizer. How is the input designed, how often is it needed to do a new optimization, is a suboptimal solution accepted if it is much faster to calculate are some possible different ways to attack the problem.

The input to the system consist of the time that it takes to complete a scrape of a certain site, and though many sites have a similar look their respective time to complete is the same. Some sites can have a lot of javascripts and moving HTML ads where as other sites might only be static pages with images and text, and even though this difference these sites take in most cases the same amount of time to

scrape.
Their deadline is also an input to the system and is directly correlated how prioritized a web site is. This is given in seconds and describes how often the site needs to be scraped to have a satisfactory result.
This then affects the way the problem can be solved since the sites can be grouped together by their respective completion time and the optimizer do not have to take into account which site that should fit where for every site, but instead a range of sites can fill up the schedules and their respective order in between does not matter.
The result of the optimization are described by the values of the variables after the optimization is done. In this project one of the variables described which job went to what node, since that list easily can be implemented into the existing system.
First step in the optimization is to group the sites with identical scraping times and identical period times together and represent this in a list with the count of how many of each specific case. The list will then always be of the same length or shorter than the original list, and as in our case it will much shorter since there is a high number of sites that have the same execution time.
This is then sorted by the lowest period time and distributed along the available nodes. Each task on every node will then be compared with its period time with the combined scrape time of the node. The lowest difference on each node will then be considered which is the "difference" of the node. The lowest difference on all of the nodes will then be the final result of the optimization, which is aimed for to be as high as possible.
The optimization then runs through this multiple times and finds which variables affect the result the most and tune these into a better result than the time before. It is then also able to see when the final result is reached since the variables impact the result in a negative way when reaching certain point. To be completely sure of a correct answer all possible varibles has to be tested though.
After the optimization are done the variables are set to their final value which then can be read and used in the next stage of the scheduling.

6 Results

As seen in figure 6 the scheduler was able to solve the given example problem, and it also differed from the scheduled done by a human in figure 5. When we first brought up the example in section 3.1 and scheduled it in section 3.2, the high priority site was given a node with low execution time and was therefore scraped more often. Since the example problem lacked an exact measure of how the to interpret the priority then it is hard to evaluate how good the schedule is. The high priority site is in this instance scraped about scraped twice as much as some of the other sites.
When scheduled with the new system, the deadlines had to be introduced and result could also be evaluated easier. The difference between the execution times and the lowest deadline for each of the nodes were very even and ranged between 5 and 6. The high priority site is then scraped 6 times more often than some of the other sites, which is a big difference from the original scheduling where it was only twice as many scrapes for high priority sites. If the same deadlines were to be used in the original scheduling than the "spare time" on the nodes would differ a lot more (between 3 and 6).
The new scheduler is more interested in making the difference between execution times and deadline as even as possible across the nodes, while the original scheduling more focused on making the execution times as even as possible (with the exception that the node with the higher priority sites had roughly half the execution time as the other sites). The two scheduling systems had both pros and cons, but the new system incorporated more qualities that were favorable to the system as a whole (e.g. robustness and the ability to decide exactly how often a site needs to be scraped).
Results were also gathered with input from the real system, to see if it could solve tha actual problem and if it could be done in reasonable time. The input was exported from the real system and some deadlines had to be decided upon trying to match as close to an actual situation as possible, which gave 258 sites to test with. The current system had 15 site groups where many sites where included in 1 group and several groups with only 1 site. In order to test how the number of site groups changes the scraping time, some sites in the real data got changed deadlines to create new site

groups. The new deadlines were very similar so that it would not change to much on the total scheduling problem.
The original data consists mainly of 2 types of sites, standard sites and video sites. The standard sites have an fixed time for scraping (set to 40 seconds) and have some different deadlines ranging from 1000 up to 3600 seconds. Some standard sites take bit longer time to load so there are some sites with an extra seconds added. The video sites on the other is set up so that 10 clips are played and each clip is played for the beginning advertisements to be shown, which usually is around 60 to 75 seconds, which then is repeated 10 times. Since the video scrapes take longer total time to execute none of them have a short deadline.
The final scheduler was able to solve the problem within reasonable time, and was therefore seen as a success. In our case this around 1.5 minutes which acceptable since this could be compared to adding on single site to a single server without rearranging anything. A list of which job that were scheduled to which node was also developed as an output aswell as the maximum spare time in order to be able to compare two different inputs to each other.
The results with an input of 258 different sites that is a representation of how the system looked today and can be seen in table 1.
One flaw the optimization has is if one of the sites needs to be run all the time then the scheduling will not work. A site that needs to be run all the time is identified with having the same scrape time as deadline, which means that there will never be any possibility to have spare time on that node. This can be solved by excluding that site from the scheduling and assigning it to a separate node which then also is excluded from the scheduling.
Since the optimizer is looking for as much spare time as possible it can't find any solution when sites have no difference between deadline and execution time.
This also leads to the insight that the best possible solution will never be better than the site with the lowest difference between execution time and deadline, since leaving it alone on one node will never produce a higher "spare time" since only the node lowest difference is considered.

Results from test on original data

Site groups	Scheduling time (s)	Time until final solution (s)
15	1m 23s	45s
16	3m 6s	2m 10s
17	3m 8s	2m 20s

Table 1: Scheduling times with real data (and some small changes to the amount of site groups)

When examining the results in table 1, it is quite clear that having to many different site groups would not benefit the system since the time it took to schedule increased when the number of site groups increased. This is because of the increasing complexity of the scheduling since more possibilities needs to be considered.

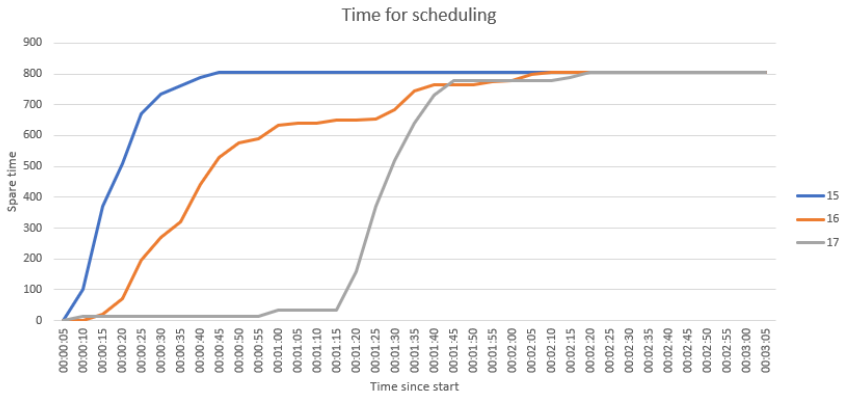


Figure 8: A graph over how the development of the lowest spare time was in respect to execution time. This was done for 15, 16 and 17 site groups respectively. 805 was the final value for each of the executions.

Another way to evaluate a optimization is to see how fast the optimizer finds its final value and how much it than uses to see if the current result is the best possible result. In table 1 the column named "Time until final solution" shows how long time it took for the system to find the best solution the first time for each of the different amount of site groups. This is also described in figure 8 where the spare time as compared to execution time is shown. This gives more info about how the optimizer

is performing, since it shows how close to the final answer is for each instance of time.

It can be seen that for 15 site groups the optimizer finds an answer quite fast and than uses some more time to ensure that the answer was correct. For 16 site groups the optimizer uses some more time but gets up to about 75% of the final answer before using some more time to get up to the final answer. For 17 site groups, it does not find any good solutions early but almost solves the problem halfway through the execution. Then it takes quite some time to find the final solution so it ends up near 16 site groups in time.

One thing to keep in mind is that this is a very specific problem solved and for a more generic problem the result would probably look a lot different. On could make the argument that the more site groups had, the longer time the scheduling will take, because of examining figure 8 one could interpret it as that 16 site groups gets quite close to the final answer but then 17 site groups catches up maybe due to something in this example that makes it almost equally hard to solve those two cases.

If one were to look at how close the solver were to the final answer after 60 seconds than it would be easy to state that a higher number of site groups increases the complexity of the optimization and therefore also the scheduling time.

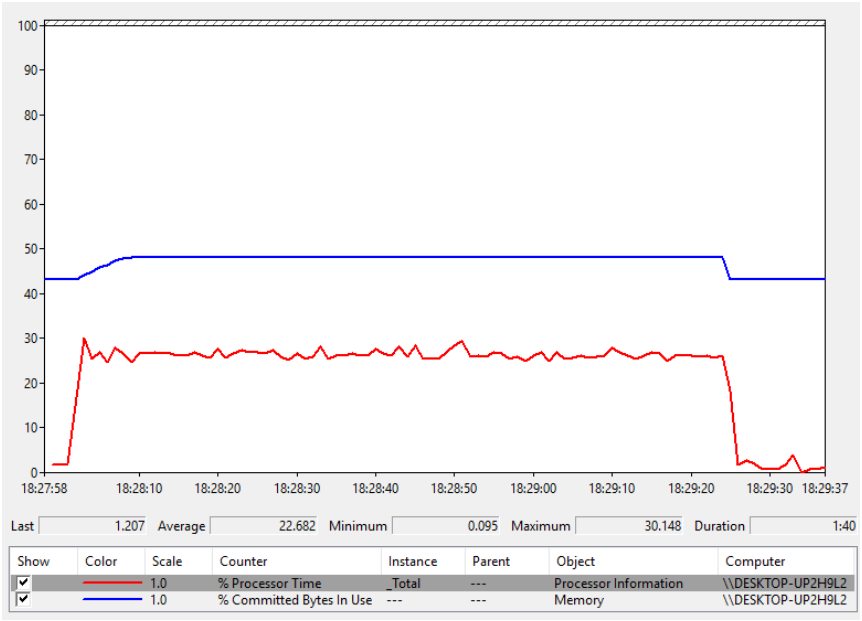


Figure 9:

To see if the scheduler used all the resources available, on could look at the Performance Monitor[6]. The Performance monitor samples system values every second which is then displayed in a graph with varying length. In our case the CPU and memory usage is interesting to look at since it shows how many calculations the CPU can handle and how much information that is currently stored in the "easy to access"-memory. In figure 9 it is displayed how the computer handled the scheduling with the new scheduler, and with a quick look it can be seen that the computer did not use close to all of its resources. The CPU reaches about 25 - 30% of the maximum and the memory only shows a 5% increse during the scheduling. This could be due to restrictions in MiniZinc or in the optimizer selected in minzinc, since there is no other preferences set that would indicate that the scheduler wouldn't use all of the available resources.

[6]Pre-installed in windows

6.1 Comparison with the general scheduler

The general scheduler was decided to be implemented in C# which was different to the new scheduler, but was decide to be used since it would be faster due to experience.

To evaluate the scheduler a comparison was done with a general scheduler with an easier approach. The general scheduler as discussed earlier with the greedy algorithm was much faster to schedule with since it did not need to loop through different possibilities of schedules, and only looped through the sites once. The time it took to produce a schedule was roughly 1 second which makes it almost incomparable to the 1.5 minutes it took for the new scheduler.

The result of schedule was that the general scheduler produced a schedule with a spare time which was half the size of the spare time that the new scheduler gave. This could not be brought to a specific number for the general scheduler since it was affected by in which order the sites where scheduled. If the sites were ordered by their deadline (both highest first and lowest first) the scheduling was unsolvable since it produced a schedule with a spare time lower than zero for some nodes. If the sites where ordered randomly then the scheduling was solvable and gave a lowest spare time around half of the spare time for the new scheduler (due to the fact that the order of the sites played a role and that affected the lowest spare time.).

From using a general scheduler and going to the new scheduler a increase to the double amount of spare time must be seen as a good improvement. This was one of the areas to focus on when improving the system so it a big benefit when deciding to use this new scheduler. The scheduling time for the two different schedulers were almost incomparable since it only took a fraction of the time for the general scheduler. Even though the scheduling time is important, a maximum of 10 minutes were set beforehand so even though the scheduling problem was possible to solve in just a few seconds, 1.5 minutes for the new scheduler is still seen as a good result.

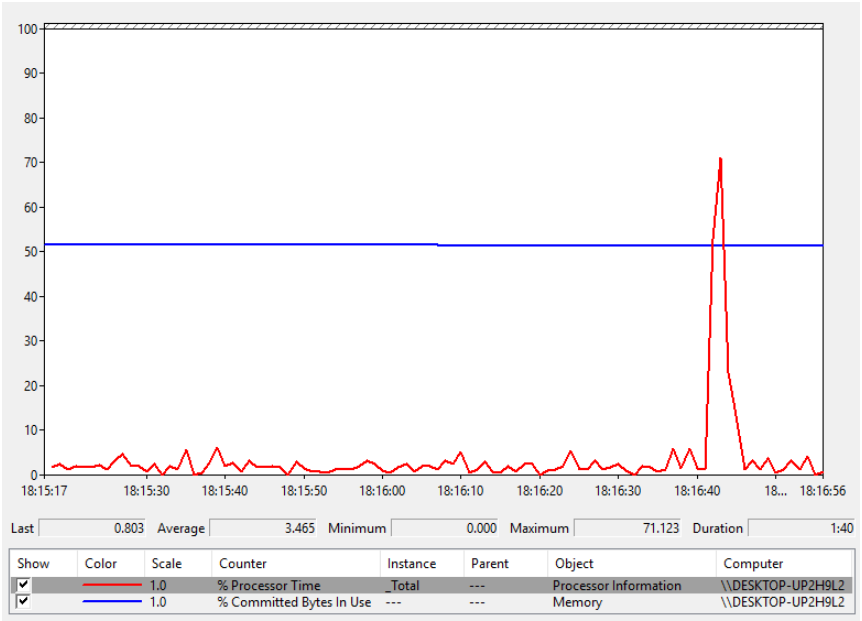


Figure 10:

To compare the general scheduler to the new scheduler, the CPU and memory usage was looked in this case as well. Since the execution time of the general scheduler was so quick it is hard to see exactly how the computer was used since the sampling for Performance Monitor couldn't be lower than 1 second. The CPU was used more than in the new scheduler, and it could even be as high as 100% since it could be between to sampling instances. The memory is not used or is barely used since the scheduling is done so much faster. The difference in CPU and memory usage could also be related to that the general scheduler was written in C# and .NET which might be more efficient how it uses the CPU related to MiniZinc.

The new scheduler done with constraint programming were then seen as better than a standard scheduler since it gave a better result in the areas where the scheduler needed to improve.

6.2 Answers to research questions

How will a mathematical model look like that solves the optimization problem that the scheduling creates?

To solve the central problem to this project which is to create a scheduler which solves an optimization problem to then create the best schedule possible, a mathematical model had to be created which then is presented to the optimizer which solves the problem at hand. In section 5.1 the mathematical model is presented and discussed further. The result presented their is seen as good enough to solve the problem.

How will different inputs change the output of the scheduler?

As a way to see what the new scheduler can handle, different inputs where tried in order to see what the outcome would be. As mentioned in section 6 the time to create a schedule changed a lot in regard to how the input looked like. When increasing the number of site groups the time to create the schedule increased in an exponential way.

Will a scheduler be able to produce a result within reasonable time, so that it can replace the current system?

One objective that was needed to be obtained is that the schedule would be needed to be created in a reasonable time, which meant that even if a good result was acquired if the time to obtain it was too long than it was not good enough to replace the current system. As can be see in table 1 a result good be found in around 1.5 minutes, which is also seen as a reasonable time frame.

7 Conclusion

The new model as described in section 5 managed to improve the system and also an on paper better robustness. Even though some fundamental changes were made, the output were not too different to what existed before and it could then be implemented to the actual system without to much effort.

The decisions to change from having a site set to a specific node and now instead have it assigned to a specific schedule were seen as positive.

One drawback for using the new model is that the sites could have been even more loosely coupled from a specific node or schedule to achieve even more security that all of the will be scraped evenly. The new model will have to schedule everything new if the amounts of nodes or some parameter for a specific site change. This could maybe have given more focus in an improvement of the system which would have led the development in another direction.

By focusing on the characteristics of the given system and also examining those parts in literature the system as a whole could be improved to a level where it was worth the effort to create a new scheduler.

The scheduling problem were not identical to anything found in the literature so a lot of ideas had to be used from different scheduling and optimization problems.

References

[1] Fuyuan Shen. Banner advertisement pricing, measurement, and pretesting practices: Perspectives from interactive agencies. *Journal of Advertising*, 31(3):59–67, 2002.

[2] A.J. Mathew, B.A. O'Meara, N. Surpatanu, and R. Taneja. Online keyword buying, advertisement and marketing, January 3 2008. US Patent App. 11/427,030.

[3] C.A. Eldering. Advertisement selection system supporting discretionary target market characteristics, April 10 2001. US Patent 6,216,129.

[4] Eloisa Vargiu and Mirko Urru. Exploiting web scraping in a collaborative filtering-based approach to web advertising. *Artificial Intelligence Research*, 2(1):44, 2012.

[5] Stanisław Gawiejnowicz and Bertrand M.T. Lin. Scheduling time-dependent jobs under mixed deterioration. *Applied Mathematics and Computation*, 216(2):438 – 447, 2010.

[6] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[7] David G Luenberger. *Introduction to linear and nonlinear programming*, volume 28. 1973.

[8] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57(1):3–57, 2015. An optional note.

[9] B. Mayoh, E. Tyugu, and J. Penjam. *Constraint Programming*. Nato ASI Subseries F:. Springer Berlin Heidelberg, 2013.

[10] Shelley M Heist. A comparison of constraint programming and integer programming for an industrial planning problem. 2003.

[11] V. JORGE LEON, S. DAVID WU, and ROBERT H. STORER. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994.

[12] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[13] Mark S Fox. Constraint-directed search: A case study of job-shop scheduling. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1983.

[14] Alan S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.

[15] Carlos Castillo, Mauricio Marin, Andrea Rodriguez, and Ricardo Baeza Yates. Scheduling algorithms for web crawling. In *WebMedia and LA-Web, 2004. Proceedings*, pages 10–17. IEEE, 2004.

[16] Stylianos Zikos and Helen D Karatza. Performance and energy aware cluster-level scheduling of compute-intensive jobs with unknown service times. *Simulation Modelling Practice and Theory*, 19(1):239–250, 2011.

[17] Peter J Stuckey, Maria Garcia de la Banda, Michael Maher, Kim Marriott, John Slaney, Zoltan Somogyi, Mark Wallace, and Toby Walsh. The g12 project: Mapping solver independent models to efficient solutions. In *International Conference on Logic Programming*, pages 9–13. Springer, 2005.