ACIT5900

Master Thesis

in

Applied Computer and Information Technology

(ACIT)

Cloud-Based Services and Operations

PAST - Profile-based Algorithm for Scheduling Tests

Intelligent Test Scheduling in the Cloud.

Håvard Hustoft

Department of Computer Science

Oslo Metropolitan University

May 2023

Acknowledgements

First and foremost I would like to thank my supervisor Kyrre Begnum for his advice and support throughout this process. His ideas and feedback has been invaluable to the development of this project. The work on this thesis has been a challenging process and would have been even more so without his aid.

I am also grateful to Oslo Metropolitan University for giving me the opportunity and the means to work on this project. All of the subjects I have gone through as a part of the ACIT program has helped me develop both academically and professionally.

I would like to thank my friends and family as well. Their continued support and advice helped me stay motivated and on track. They were there for me when my motivation faltered and helped me regain my footing. Lastly I want to express my gratitude to my girlfriend, for always being in my corner and pushing me to keep going.

Abstract

Computer technology is an essential part of today's society with more and more critical infrastructure and important services being moved to the Internet every day. This widespread use of the Internet has increased the need for scalable hardware and software that can process a lot of traffic at the same time while not using excessive energy. This has contributed to the meteoric rise of cloud computing, and many companies are moving their applications and services to data centers.

With computers and the Internet being such important parts of all areas of society, it is crucial that new software is tested before being shipped to the Internet. This is where software testing comes in, with most modern IT solutions having automated tests that often run in the cloud. These tests are triggered when changes to the code are uploaded to its repository, and are often running on the same cloud environment as the company's other applications and services.

This project seeks to combine efficient power-saving strategies for cloud environments with software testing by developing an algorithm that schedules tests according to how resource demands change during their run-time. It does this by using workload profiles that are defined by how the resource usage of a given test changes over time, which differs from traditional workload profiles that often represent the resource demand as a static number. The goal is to use these profiles to make a schedule that allows compatible tests to be executed simultaneously on one server without exceeding limits on resource use.

Contents

List of Figures

List of Tables

1 Introduction

The demand for new software is at an all-time high and is only going to keep increasing as ever more people gain access to the internet. Roughly 63.1 percent of the world population has access to the internet as of July 2022 [16], meaning that the market has the potential to grow even bigger. A large part of the internet is hosted on cloud services, which has seen an explosion in popularity over the last 20 years.

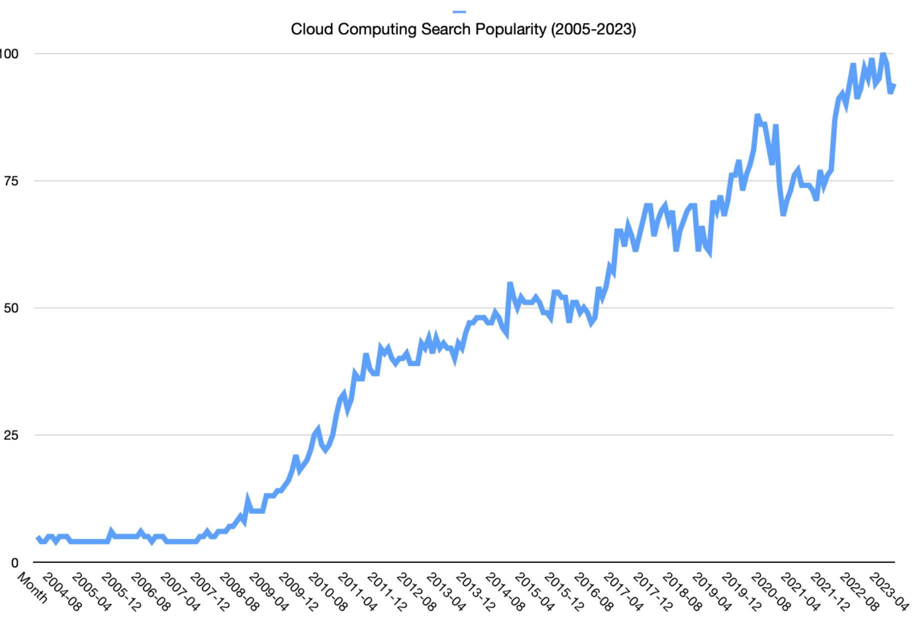


Figure 1: *Cloud Computing popularity based on the number of Google Searches per month between 2005 and 2023. The graph shows a substantial increase in popularity over the last 20 years. Data Source: https://trends.google.com/trends*

Such rapid growth means that there is a race to be first to market with new products, and people expect to constantly receive new and functional software. The enormous appetite for new software puts a substantial load on tech companies, and many struggle to balance quality and rapid delivery. This phenomenon has led to the creation of several software development practices that are intended to help developers rapidly deliver quality software, such as agile development. Agile methods differ from the old waterfall methodologies in that their development cycles are less rigid and more adaptable. This is important in the current technological climate where the needs of the customer frequently change and companies have to adapt to a volatile market.

Agile development encompasses many practices, such as continuous integration, that are intended to make the rapid delivery of software easier for the developers. Continuous integration, or CI for short, seeks to automate the merging of code from several developers and development teams into one final product. Continuous integration is often combined with continuous delivery, which is referred to as CI/CD [14]. An important part of any CI/CD pipeline is automated testing, which comes with its own problems and challenges.

Automated tests need to be as quick and efficient as possible, as the tests are executed each time code changes are uploaded to the repository. This is necessary in modern software development due to the constant influx of code changes from different developers and teams. The developers need to make sure that the changes they upload don't break the application, which is why all relevant tests need to run each time changes are made. Inefficient tests lead to wasted time and resources for the developers, but there are many ways to optimize the tests and the environments they are running in. Developers and testers need to balance the number of tests and the time it takes to run the pipeline containing those tests. This means that test optimization involves ensuring that important test cases are covered while not making too many tests, this can be measured with test coverage. Test coverage is essentially the proportion of application code that is covered by tests, calculated as lines of code covered by tests divided by the total number of lines.

$$\text{Test coverage} = \frac{\text{Lines of code covered by tests}}{\text{Total number of lines}}$$

Optimization of software testing involves maximizing test coverage while minimizing the time it takes to test the application. But there are also other factors to take into consideration, such as the hardware used to run the tests. This is where cloud resource allocation and power efficiency strategies come in, as most software testing is done in cloud environments.

High computational demand in modern society has led to most test environments being located on cloud services, as most businesses lack the necessary resources to host everything themselves and it is more practical to host all environments on the cloud if their other applications are hosted there as well. This has led to cloud providers such as Amazon Web Services, Microsoft Azure and Google Cloud becoming the premiere hosts of applications and associated infrastructure.

Cloud providers and their data centers have an abundance of resources ready to be allocated, which is practical for the customers but can easily become expensive if they are used in an inefficient manner. Inefficiencies such as test environments being allocated more resources than they need could be avoided by predicting how computationally demanding a pipeline is, and placing the runners in a consolidated manner, thereby using less of the available hardware. The programs that are running the software tests are what we refer to as runners, they usually come in the form of specialized virtual machines. Scaling is also very important in software testing, as the different tests in a pipeline will have varying computational demands, thereby increasing the need for an environment that can dynamically scale the runners up and down. This means that the environment can avoid unnecessary resource usage by scaling the infrastructure down if a test is using less resources than expected.

A common issue for businesses hosting their software in the cloud is using too much or too little of the available resources. Resources are used up by virtual machines (VMs), which are virtual operating systems that are confined to a portion of the physical machine, physical machines are often referred to as servers in cloud computing. VMs are different from normal computers, which usually use the entire physical machine. A common problem statement in cloud computing revolves around optimization of the partitioning of the physical machines in a way that avoids giving too much computing power to each VM while also making sure that they have sufficient resources to complete their tasks. This is one of the reasons why all the big cloud providers have implemented dynamic scaling in their data centers, meaning that their machines automatically assign more or fewer resources to the VMs according to the needs of the customer.

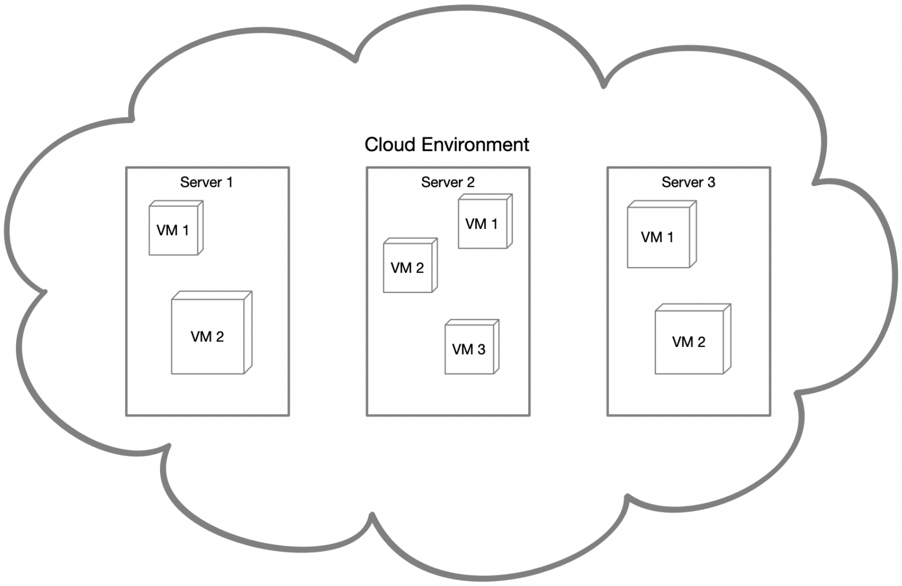


Figure 2: *Illustration of a cloud environment with three servers. Each server has a number of Virtual Machines of varying sizes.*

Another part of the dynamic scaling process in many data centers is workload consolidation, which consists of consolidating similar VMs to the same physical machine and idling other machines, thereby saving power and ensuring more efficient resource usage. Profiling the computational demand of a certain type of workload could help optimize this process, as knowing what kind of resource usage we can expect could make provisioning the correct amount of resources easier. This paper will be exploring the usage of workload profiling in software testing with the purpose of making a more intelligent scheduling algorithm that can facilitate a higher degree of consolidation.

1.1 Problem Statement

Explore the usefulness of workload profiling in order to facilitate green resource provisioning for software testing in the cloud.

Workload Profiling refers to the measurement and characterization of *resources* such as CPU, memory and I/O operations. I/O refers to input and output signals in a computer, measuring it means that we measure the rate of communication between a computer and the outside world. The measurements are used to create profiles that could be used to approximate resource usage of future tests so that we may facilitate the provisioning of resources in the cloud. Since *software testing* is such an integral part of every CI/CD pipeline, it is important to consider its resource usage if the goal is to minimize costs and decrease electricity usage, thereby making the cloud *greener*.

2 Background

Before proceeding with the theoretical background and literature review of the subject at hand, I would like to give an introduction to the subject, my personal background in software testing and what motivated me to choose this subject.

Efficient usage of cloud resources has become extremely important for developers that wish to host their applications in the cloud. Inefficient or careless use leads to greatly increased costs and potential security risks for the developers and the cloud provider, but can be avoided if precautions are taken. These precautions can be in the form of choosing the right kind of infrastructure for the job and optimizing the software and environment to not use excessive resources. Optimization can be done in many ways, but there is a noticeable lack of optimization of software testing in cloud environments.

This thesis seeks to provide a proof of concept of one such optimization, namely more intelligent test scheduling that can facilitate a higher degree of consolidation. My personal motivation for choosing this subject comes from my experience as a DevOps Engineer, where I was in charge of migrating CI/CD pipelines from one environment to another. I remember being surprised by the lack of thought being put into the test environments compared to the application environments. Applications do experience more traffic than the test pipelines, but it would make sense to optimize all environments if the goal is to save resources. The following chapter gives more information on cloud computing, software testing and where PAST fits in the bigger picture.

2.1 Software Testing in the Cloud

Testing has always been an important part of software development and remains one of the most popular research topics in software engineering [13]. The importance and popularity has led to a wide range of methods being developed to improve the quality and performance of software tests, such as search-based testing and random testing to name a few [13]. All of the new methods and paradigms are combined with other technologies such as cloud computing to achieve fast and maintainable on-demand testing.

The complexity and societal importance of computer software entails an increased need for quality assurance from software developers. The testing process has thus become increasingly sophisticated over time, with different tests being divided into several phases in order to distinguish between their purpose and importance. The most commonly used testing phases are as follows:

- **Unit Tests**. Unit tests are meant to test the most basic and fundamental functionality of the software. They are typically short and simple, and numerous.
- **Integration Tests**. Integration tests examine if the various components of the software are able to function together. They typically group components and test if they still comply with functional requirements after changes have been made to one or more of said components.
- **System Tests**. System tests are the logical follow-up to integration tests. They examine if the system as a whole still complies with functional requirements after changes have been made to the code.
- **Acceptance Tests**. These tests are meant to examine if the software is able to fulfill the needs of the user. Acceptance tests are difficult to automate, as they require the input of external parties.

The first three steps of the testing process are usually automated in a CI/CD pipeline, which is triggered each time code changes are uploaded. The final step, acceptance testing, is usually not automated due to the need for input from the users. This means that acceptance tests are not so relevant to the workload profiles introduced in this project, as the profiles are meant to optimize the scheduling of automated tests.

CI/CD pipelines are not only meant to test software before it is delivered to the users, but they are also meant to automate the delivery itself. This is where continuous delivery comes in, as it would be far too time-consuming for the developers to manually update their servers and services. This part of the process falls outside the scope of this project, as it is meant to explore the optimization of automated tests and the continuous integration part of CI/CD.

The need for constant testing and uptime of related services has lead many companies to host their infrastructure in the cloud. This effectively offloads a large part of the responsibility of maintenance and monitoring to the chosen cloud provider, while also enabling a more sophisticated and computationally powerful infrastructure. The cloud provider is bound by Service Level Agreements, or SLAs for short, to provide a certain level of service to the customer. This means that the customer usually has guarantees of uptime, resource access, and more. An on-premise solution would mean that the company has no guarantees of uptime and it would be up to themselves to make sure that all server software and hardware is functioning correctly, this could be significantly more expensive and time-consuming than employing the services of a cloud provider.

It is common for companies to host their entire infrastructure in the cloud, which includes the machines that are running software tests. The associated increase in traffic in data centers has led to more research into the resource usage of applications that are hosted there, with high resource usage entailing increased costs for the customers and higher electricity consumption in the data centers. Some researchers focus on the environmental impact of data centers, a field known as green cloud computing, while others focus more on the economic aspect. These fields overlap quite a lot, as saved resources would entail both decreased costs for the user and less electricity being used by the data center thereby contributing to a greener cloud.

Software testing has been a part of the development process for a long time, but there has not been a significant effort in the optimization of testing environments. The testing process itself has been streamlined and optimized due to the need for frequent and thorough testing, which is a time-consuming and resource-intensive process, but there is still room for improvement in the environments that are running the tests. Developers tend to just throw resources at the testing environment so that they can be sure that all tests are completed and don't crash as a result of insufficient computing power. People generally don't look too closely at what kind of tests they are planning to run on a given environment but they need to be sure that there is sufficient CPU power and memory available, so over-allocation of resources is common.

The problem of over-allocation can be illustrated with a more practical example, where virtual machines are viewed as cars that are parked in a garage (data center). *Imagine a parking garage, we don't know what kind of cars will be parked there, so we make all parking spots big enough to fit a truck. Now we can be sure that all kinds of cars are able to park there without issue, but now the problem is that each parking spot uses a lot of space and the parking garage itself will need to be very big. A better strategy would be to investigate what kind of cars usually park in that area, and design parking spaces that will accommodate them. This way we can avoid having to allocate so much space for each parking spot and thus be able to fit more parking spots in the same area.*

Workload profiles, like the ones presented in this project, can be viewed as parking spots for software tests. By outlining the typical resource usage of different tests, one could potentially use less computing power and fit more test-runners on the same hardware, leading to saved resources and a more efficient testing environment.

2.2 Cloud Computing and its Role in Society

Cloud computing has become one of the most important technologies in the world. Most modern web pages and applications are hosted on cloud-based infrastructure, which includes many critical services that are used every day. If all cloud services were to fail at the same time, society would cease to function and the Internet would become inaccessible. The cloud itself is not a new thing, but this high degree of dependence is. Cloud-based services have been growing in popularity since their conception and is only going to keep increasing if the technological climate keeps its current trajectory.

Cloud-based infrastructure lets companies scale their services without having to upgrade their own infrastructure, as the cloud providers have an abundance of resources available that can be provisioned by the customers. This is a substantial improvement for many companies compared to earlier times when they had to host services on their own infrastructure, which meant that they had to perform their own maintenance and installation of updates and upgrades. All of this can be handled by cloud providers these days, the most popular ones being Amazon Web Services (AWS), Microsoft Azure and Google Cloud. These three companies dominate the market with AWS having a market share of 34%, Azure 21% and Google 11% as of 2021 according to Statista [15].

Cloud computing depends on several other technologies in order to function the way it does in the data centers we use today. One of the most important of those technologies is virtualization, as it allows us to host several virtual computers on the same hardware. These computers are referred to as Virtual Machines, or VMs for short. Customers can provision VMs at their chosen cloud provider, they can decide what kind of operating system to use, how powerful the machine should be and much more. This leads to a highly customizable environment that developers can use to create the perfect conditions for their applications.

The popularity of cloud computing and variety of demands from customers has led to cloud providers offering many different services in order to be able to cater to the increasing demand for new types of infrastructure. AWS has many different services available depending on what the purpose of the VM is, buckets are used for storage, EC2 instances are general-purpose VMs to name a few. Most modern companies use a complicated combination of many different services to host their applications and web services, leading to a need for specialists whose job it is to maintain and upgrade the infrastructure. This means that the popularity of cloud services has led to a whole new profession, the Cloud Engineer. These engineers have an important responsibility, and doing a poor job could mean that the entire application stops functioning. All this goes to show how important cloud services have become and research into the subject has thus increased significantly over the years.

Cloud services are especially useful for applications that require a lot of computational resources, such as big data and artificial intelligence applications. Such applications are constantly analyzing and processing large amounts of data and require sophisticated and efficient infrastructure to do so without crashing or slowing down. Big data and artificial intelligence are becoming increasingly popular and more and more companies are attempting to use them in their own applications. Those applications tend to use a lot of resources, so this further compounds the need for the expansion of cloud infrastructure services. Large amounts of data traffic and computation in data centers also increase the need for electricity, which has led to data centers becoming one of the largest consumers of electricity worldwide [10].

Green Cloud Computing

Green cloud computing is a fairly new and popular paradigm that has become a highly researched topic after people started noticing the large amount of electricity being consumed by data centers. It has been estimated that data centers account for as much as 1.5% of global energy usage, a number that is expected to keep increasing as cloud computing grows in popularity [10]. Mastelic et al. outlines recent trends in energy efficiency within cloud data centers in their paper *Recent Trends in Energy-Efficient Cloud Computing* [10], showing how academia and researchers have been counteracting the ever-increasing demand for electricity in cloud computing. They mention how a data center is comprised of multiple parts that all require a large amount of electricity, with the computers and cooling being the most demanding. Any technology or methodology that reduces energy usage or improves energy efficiency helps further the goal of greener cloud computing.

An important part of energy efficiency and reducing the resources used in a data center is efficient VM scheduling and resource allocation. Virtual machines are placed on physical machines that have available resources, but there are usually multiple available physical machines at any given moment, so which one should be used? It is ideal to keep the spread of VMs as small as possible, meaning we want to consolidate the VMs to as few physical machines as possible. Physical machines that are not in use can be placed in power-saving idle modes, so it is beneficial to keep them idle whenever possible. Idle machines can easily be switched back on if more applications are being started or if existing ones need to scale up due to an increase in traffic. VM consolidation requires sophisticated resource scheduling techniques in order to ensure minimum resource usage without negatively impacting the performance of the applications and services hosted on said VMs.

Network usage is also a big factor in electricity consumption, as network devices tend to use a lot of electricity when there is a lot of traffic. Data centers need highly sophisticated network equipment, such as advanced routers, to handle the incredible amount of traffic coming in. One way to handle this is through power-saving modes and adaptive transmission rates [10].

Energy-saving methods are beneficial in several ways by not only saving electricity and thereby helping prevent global warming but also by reducing costs for cloud providers. Mastelic establishes improving the network infrastructure as the first step towards more energy-efficient data centers. An energy-efficient data center will need to be optimized in both hardware and software, as all parts of the cloud environments tend to use a lot of resources. It is also possible to save energy by reusing thermal energy produced by the data center in creative ways.

Some cloud providers have developed conceptual data centers that can reuse the energy produced by their machinery to reduce the climate impact of their data centers. The machinery produces a lot of thermal energy as a result of the high amount of computation being done at any time, also as a result of high internet traffic on their network equipment. This energy can be used to heat up water that can then be sent to nearby building to be used as heating [1], or recycled and reused either in the same facility or elsewhere. Both of these methods and more are being used by Amazon in their data centers when it is possible, according to a blog called *Water Stewardship in Data Centers* from their website [2]. Reusing energy in this manner shows that energy efficiency improvements are not limited to the software and computer equipment, but can also be implemented in the infrastructure of the data center itself.

Novel methods of utilizing green energy in the cloud have been developed as well, in terms of software, hardware and data center placement. Some data centers are built close to sources of green energy, such as hydro or wind power. Planning data centers this way could decreases their impact on global warming and potentially their costs as well. Software has been developed to take advantage of such data centers, more specifically VM placement and migration algorithms. Placing VMs on data centers that have access to green energy lets environmentally aware companies reach their goals of reducing the climate impact of their services.

Access to green energy can vary depending on the conditions of the surrounding environment, energy from wind is for example dependent on the current weather in the area. It is possible to check the availability of energy in the power grid of a given area, which opens up the possibility of keeping track of where there is extra power available. Movement of VMs is not limited to the data center they are currently hosted at, they can also be migrated to other data centers over the internet. Combining this with the previous point opens the possibility of constantly moving VMs to data centers that have available green energy, thereby letting them run solely on green energy even if the access to renewable energy varies. This concept was explored by Ingvild Stølen in her thesis titled *The CAST-algorithm bridging green energy with continuous testing* [17].

2.3 Literature Review

There is extensive research available in the field of workload profiling in cloud environments, but most do not focus on software testing. Researchers tend to focus on application performance and establishing profiles that can be used to predict resource usage of various software. Studies such as *An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Model* by Moreno et al. [12] outline the high variability in cloud computing workloads, owing to the fact that there are so many different kinds of applications and services being hosted on cloud infrastructure. They bring up several factors that need to be considered when measuring the computational demand of applications in the cloud, such as metrics related to the users of the application. This includes the number of users, user behavior, and how often users need the application. These factors can be ignored when one is specifically looking at software testing profiles, as the amount of traffic and application use cases are predefined by the developers. This predictability could increase the usefulness of software testing workload profiles, as resource demand should remain mostly consistent across subsequent runs of the same tests.

Some tech companies have also become interested in cloud workload profiling, with giants such as Google participating in studies that seek to classify workloads and use them to improve efficiency in cloud environments. Mishra et al. sought to use insights from Google Compute clusters to characterize workloads and then use them to explore applications in capacity planning and task scheduling [11]. Capacity planning seeks to select resources such as CPU, network and memory to minimize cost while under the constraint of applications being able to meet their service level objectives while using as few resources as possible [11]. Task scheduling is the process of optimally placing application workloads on available hardware, which is viewed as a multi-dimensional bin-packing problem in their paper, *Towards characterizing cloud backend workloads: insights from Google compute clusters*. Bin-packing is an optimization problem that revolves around fitting as many items as possible in as few bins as possible, and is common in computer science projects that focus on resource allocation. Mishra et al. divided their approach into four steps, first they identified the workload dimensions, then they constructed the workload profiles using known classification algorithms, then they determined break-points in the measurements they received before finally merging similar tasks into the same profile.

Do et al. bring up the importance of resource utilization in their paper *Profiling Applications for Virtual Machine Placement in Clouds* [8]. They sought to examine the relationship between the application workloads and resource utilization in the cloud, and determine the degree of correlation between them. Ideally we would want there to be a high degree of correlation because it would indicate reasonable resource usage, but this is not always the case. Their research paper highlights the importance of more efficient resource utilization in data centers, as over-provisioning of resources is a very common problem, with some cloud providers having resource utilization below 50% [8].

Some researchers bring up the fact that virtual machine consolidation can lead to service degradation if we do not take the necessary precautions when it comes to the resource usage of each machine that is being consolidated. Ye et al. highlight this point in their paper, *Profiling-Based Workload Consolidation and Migration in Virtualized Data Centers* [18]. There is also significant overhead associated with virtual machine migration and consolidation, which is why they also propose ways of minimizing the number of migrations. This topic is highly relevant in green cloud computing, where the goal is more efficient power- and resource-saving strategies.

Green cloud computing is a highly researched topic, with several of the major cloud providers also becoming more and more interested in reducing their environmental impact. The power consumption of cloud data centers accounts for a substantial share of global power usage, and is expected to keep increasing in the future. Strategies such as virtual machine consolidation, efficient cooling and creative use of thermal energy produced by data centers have been implemented to curb the growing energy demand of data centers. The subject of this paper ties into virtual machine consolidation, as the goal is to produce workload profiles that can be used to consolidate machines that are running software tests. This approach is intended to facilitate the process of finding machines that have the capacity to host several test runners at the same time without affecting the performance of the runners, this could potentially save energy in data centers because other machines would be freed up and made idle, thereby using less power.

Cloud data centers were estimated to account for about 1.5% of the global power consumption in 2010 [6], and the demand for cloud services drastically increased since then. This highlights the need for efficient power-saving strategies in cloud data centers, as the energy demand would otherwise become unmanageable. This multi-faceted issue can be handled in a myriad of ways, and the combination of different methods has proven to be very efficient in tackling the increasing demand for power. This is evidenced by the fact that even though the use of cloud computing and the associated expansion of data centers has been increasing, the share of the global power consumption attributed to cloud data centers has not increased as much as we would expect. A study commissioned by the EU in 2020 reported that cloud data centers accounted for 2.7% of the total energy consumption within the EU [5]. One would expect this number to be higher in 2020 after years of significant growth in the cloud sector, but efficient power-saving strategies such as the ones mentioned earlier have made sure that the power consumption of data centers has not grown out of control. The same study brings up the fact that the power consumption in data centers is expected to increase to 3.21% by 2030 if we keep the current trajectory [5].

System-wide optimization methods have been developed by researchers seeking to increase hardware utilization and thereby reduce the carbon footprint of data centers. Chen et al. presented one such method in their paper, *Profiling Energy Consumption of VMs for Green Cloud Computing*, involving profiles that characterized energy usage of VMs [3]. There are many avenues to consider when seeking a greener cloud environment, researchers such as Chen focus directly on the energy usage of VMs, while others focus on resource utilization. In spite of their focus being on energy usage, they ended up focusing more on resource utilization, with a modeling technique called *vMeter*. This technique was developed in another paper, where they discovered a positive correlation between a system´s power consumption and the utilization of its components.

This gives credence to the idea of using workload profiles of test-runners to improve resource utilization and thereby reduce the power consumption of data centers.

The idea behind this project is closely related to bin packing in that it revolves around fitting more VMs on one server. There are two major differences however, which distinguishes this project from previous research. The first is that the algorithm does not actually try to place VMs on physical machines by itself, but it could be used as more of a facilitating agent in the bin packing process by letting more VMs fit on the same machine than what would otherwise be possible. The second way it differs is in the fact that the resource usage varies over time and the algorithm tries to look at the pattern of the usage instead of representing it as a static number. De Cauwer et al. introduces a similar concept in their paper, *The Temporal Bin Packing Problem: An Application to Workload Management in Data Centres* [7]. They refer to the problem of temporal bin packing as a generalized approach to the classic problem, where items have a lifespan in addition to their size. They bring up the fact that data centers tend to over-provision VMs in order to ensure that there is always enough resources available to the customers, but this is problematic in that only a fraction (6-12%) of electricity is used in productive computation [7]. The over-provisioning and associated excessive electricity usage also strengthens the thought behind this project. Their approach involves a set number of bins (servers) and items (VMs) where each item has an associated resource requirement and a lifespan.

De Cauwer defines the temporal bin packing problem with a slightly different purpose than what is discussed in this thesis. They focus on using the time an application uses rather than how the resource use changes over time. The goal is still the same however, namely fitting more VMs on the same server.

2.4 Application Programming Interfaces

Application programming interfaces, or APIs for short, represent a way for computer programs to interact with each other across platforms. APIs are designed for reuse by other developers in their applications and are thus modular and extendable in nature. Modular code consists of code that is separated into independent and interchangeable modules, which means that they can be reused by other programs independently of each other. APIs are usually made using object-oriented programming, which is a programming paradigm that revolves around persistent objects and data rather than functions. An object is usually represented as a class that contains multiple fields of data as well as methods that can be used to add, remove or change the data contained in the object. Classes can be reused in other parts of the same program or imported and used by another separate program, which makes it an ideal programming paradigm for the creation of APIs. Having the ambition of turning a program into an API right from the start of development is likely to increase the reusability and user-friendliness of the program, even if it does not turn into an API at the end. In order to facilitate this, it is usually a good idea to design the program using object-oriented programming.

APIs are usually exposed to the public by having the users make an account before receiving an API token that they can use to identify themselves and gain access to the API. The user can then make queries to the API or use methods and classes provided by the API and use them in their own programs. A common form of API is one that lets users retrieve certain data from a website or application, this could for example be a weather reporting site letting their users retrieve data about the weather for the upcoming week. If a user wants to make a website that can display the weather in a different way than what is currently being done by the weather reporters, they can retrieve the data through the API instead of having to generate it themselves. This saves a lot of time and resources for everyone, as otherwise everyone would have to gather their own data about the weather, which would be incredibly time-consuming.

Programs that are meant to help facilitate an already existing process could benefit from being programmed as an API. The program discussed in this project revolves around the facilitation of more efficient resource allocation and scheduling, which opens up the possibility of it being developed as an API that can be used by resource scheduling programs. If there already exists a program that does resource scheduling in cloud environments, it makes more sense to make an API that can be utilized by said program instead of making a brand-new resource scheduling program. Making an API instead of a self-contained program also opens the way for further development through direct and indirect input from the users of the API.

2.5 The Balance of Speed and Quality in Software Development

Time is of the essence when developing applications and web pages, and developers need to keep a balance between delivery time and software quality. Updates are churned out at a rapid pace in most modern IT companies, requiring a robust infrastructure to make sure that deliveries go smoothly and development is constantly moving to the next task. Developers can't be idle after completing one task, they have to move on to the next one to keep up with the demands of the users and the market. The robustness of the software and the infrastructure around it comes from high software quality, thorough testing and a secure environment, among other things.

A secure environment is most commonly accomplished by hosting the infrastructure on a cloud service, thereby receiving certain guarantees from the provider. Guarantees provided by the cloud service are not enough to ensure high-quality software and uptime however, the developers also need to test the software and thereby provide quality assurance. This is commonly done through automated test pipelines, which can take a lot of time to run depending on the thoroughness and types of tests contained within.

There are many factors that can have an effect on how fast or how slow a pipeline completes its tests, such as how many tests there are and the amount of resources allocated to them. A lack of allocated resources can lead to slow and unstable test execution, which can be very harmful to a business if it happens repeatedly. There seems to be a lack of concern for the condition of the environment running the tests, as many developers seem to just launch the tests and let the cloud service handle the rest. My own experience in the industry has indicated that we tend to simply rerun the pipeline if it fails for an unknown reason, without investigating the reasons behind too thoroughly.

2.6 Bin-packing

Bin-packing is an optimization problem where the goal is to fit as many items as possible into as few bins as possible, and is highly relevant to resource allocation in cloud environments. There are several algorithms that can approximate solutions to large instances of the problem, such as first-fit and best-fit algorithms. The version that is used in resource allocation problems is known as Virtual Machine Packing, or VM-packing for short. This problem is different from standard bin-packing in that it allows bins to share resources with other bins, which is common when hosting virtual machines on the same physical machine. The virtual machines can share the computational resources of the host machine, as long as there is room to do so. Several machines being present on the same hardware can reduce the resources required to run the virtual machines, as the machines can share certain elements that only need to be stored once.

May variations of bin-packing algorithms are used in cloud environments, mainly in resource allocation and virtual machine consolidation. The idea of bin-packing is directly analogous to the process of virtual machine consolidation, as the goal of consolidation is to fit as many VMs as possible in as few physical machines as possible. Workload profiles are also relevant in this process, as such detailed knowledge about the demands of a VM may help facilitate the consolidation of VMs by making it easier to see how multiple VMs can fit on the same hardware.

The research in this field is abundant, but there is not much to find when it comes to bin-packing within software testing. Even though bin-packing is not part of the main theme of this project, it is still relevant to the underlying problem of resource utilization and efficiency in cloud environments. Christensen et al. [4] bring up many different approaches to bin-packing in the cloud in their paper, *Approximation and online algorithms for multidimensional bin packing: A survey.* The survey brings up the large amount of mathematical research that has been done, as bin-packing is an NP-hard problem and its solution has only been approximated as of today. One of the fields of research brought up in the survey is d-dimensional vector bin-packing, in which each bin is a d-dimensional vector and vectors have to be placed into unit-vector bins. The parameter d represents the number of different resources being measured, such as CPU, memory and I/O statistics. This form of bin-packing has many applications in resource allocation and virtual machine placement, as each VM or job has multiple resource requirements [4]. A machine has a set upper bound for each of the resources, the goal then becomes to assign the vectors, where each vector represents a workload, to machines without exceeding the bounds set by the machine. The problem then becomes identical to the vector packing problem, which has many applications outside of cloud resource usage [4].

3 Approach

The preceding literature review gave some insight into the current research being done in the field of resource utilization in cloud environments, showing us that there are many avenues for further improvement. Research is typically not restricted to a specific type of application, instead having a more general view of applications that run in cloud environments. This leaves room for more focused research on specific types of programs, such as test-runners, which brings us back to the problem statement: *Explore the usefulness of workload profiling in order to facilitate green resource provisioning for software testing in the cloud.*

The goal is to improve the resource utilization of servers in cloud environments, specifically with regard to virtual machines that are running software tests. Workload profiles could be used to improve the process of resource provisioning, mainly by offering more knowledge about the demands of the VM before provisioning the appropriate amount of resources. Another benefit of this knowledge is the ability to potentially host more virtual machines on the same hardware if one first makes sure that their profiles are complementary. Complementary meaning that the resource usage of the profiles follows patterns that fit together. This can be illustrated with an example; say we have two tests, test 1 and test 2, test 1 has a spike in CPU load 30 seconds into its runtime, while test 2 has low CPU load 30 seconds into its runtime, then the profiles for those tests are compatible if the rest of their runtime also follows the same pattern.

Another more abstract way to visualize the concept could be to view the profiles as keys, each profile having a unique pattern in their resource usage, with the spikes in for example CPU load being the pattern of the key. If two keys have complementary patterns, meaning the spikes of one key correspond with the valleys of another, we could fit those keys on top of each other. In other words, we could fit two virtual machines on the same hardware if the spikes in resource usage of one VM correspond with dips in resource usage on the other. This kind of temporal profile differs from other workload profiles, where resource demand is often represented as a single number instead of a pattern that changes over time.

The problem can be attacked from many angles, so the exploration part of the problem statement refers to the examination of the different directions as well as a review of the usefulness of such profiles. The profiles are meant to facilitate efficient resource allocation, the focus of the paper is the profiles themselves, which is only the first step in resource allocation. This means that this chapter revolves around the development of the profiles, and further research and applications are discussed in the results and discussion chapters. The problem was approached as a development of a proof of concept, meant to show that application-specific temporal workload profiles can be useful in resource allocation. This thesis revolves around using such profiles to schedule software tests in the cloud, but the concept could be generalized to other applications as well.

3.1 Research Methods

Research can be either exploratory or comparative, with each method being suited to different use cases. The usefulness of a comparative research method depends on the quality of available research in the given field, and sometimes a lack of research can make comparison very difficult. Exploratory research charts and investigates available research in order to learn valuable insights and reveal possible directions for further research. The following sections will explain these research methods a bit further.

Exploratory

Owing to the first word in the problem statement, *explore*, the research in this paper is mainly exploratory. This means that the main goal is to gain more insight in the field and explore the usefulness of the technology. The first step of the exploratory research is to browse the available literature and gain an overview of the subject, which in this case is resource allocation in cloud environments. Exploratory research can be viewed as a journey where we figure out which direction to take underway, finally arriving at a conclusion after exploring several possible paths. The downside of this method is that the researcher does not commit to developing a solution to the problem, but rather focuses on the exploration and charting of possible solutions and their feasibility.

Exploratory research helps researchers understand the current status of the problem and field, and gives them an overview of possible directions to take when working on the problem. It is difficult to determine if a solution is feasible without first examining the current practices in that field, and one may gain valuable insights that can significantly improve the results by reviewing relevant literature. The literature can help the researchers in providing possible angles that the project can explore, resource allocation could for example be aimed at reducing the environmental impact of data centers or reducing costs for cloud providers and their customers. Exploration and literature review can be used as a first step when working on a project in applied sciences such as information technology, it does not solve the problem but it may help the researchers in deciding which direction their project is going to take.

Comparative

Comparative research involves the comparison of different methods or objects of research with the purpose of discovering each object's advantages and disadvantages. It could for example involve the comparison of different methods of resource usage optimization in cloud environments or different ways of planning software tests. This can be a highly time-consuming process, as one would need to find multiple studies on the same subject and test the relevant technologies to their limits, instead of just focusing on a singular technology or method. Comparative research methods are best suited when there is an abundance of available methods and technologies to test, which is not always the case. The objects being compared need to be relevant to the topic at hand if the results are to be useful in the research being done.

3.2 Assumptions

Most research begins with an assumption about the problem domain, and this project is no different. The underlying assumption in this case is that there is significant wastage taking place in cloud environments that are running software tests. This ties into a more general assumption about resource waste in data centers, meaning that we assume that the computational resources in cloud environments are not being fully utilized, an assumption that is backed up by the research presented in the literature review.

The abundance of research into resource allocation and more general energy efficiency in data centers shows us that there is room for improvement there, but there is a noticeable lack of research into the resource usage of specific types of applications such as test runners. Herein lies another assumption, that there is room for application-specific optimization methods that might improve utilization even further. This project focuses on methods that are specific to software tests, but it might be possible to generalize them to other applications as well. The assumption of inefficiency in software tests that are run in the cloud leads to many possible solutions, but there is no guarantee that the solution will be useful for the industry. This being a short project meant that there were some constraints on the possible scope. The problem statement says that the project will explore the usefulness of workload profiling in resource provisioning, this is done in a mostly theoretical context as implementing it with existing technologies might be too time-consuming. Due to this, one of the main goals is to answer the question of whether this kind of technology could be used to alleviate the assumed resource waste in data centers.

3.3 Goals

There are many goals to think of when developing a project like this, and the list had to be narrowed down to realistic proportions. The result was a list of four main goals that are outlined here.

- First goal: Find software tests that can be used as examples of how much computational power typical tests use.
- Second goal: Design a script that can run the tests repeatedly while also recording CPU and memory usage.
- Third goal: Use the recorded data to define profiles that describe the resource usage pattern of a given software test.
- Fourth goal: Design a scheduling algorithm that uses compares the profiles and finds out if they can run concurrently on the same server without overloading it.

3.4 Similarities to Bin-packing

Bin-packing is similar to the algorithm of this project in some ways, but also quite different. Bin-packing algorithms usually view the workloads as static boxes whose dimensions are determined by the number of metrics that are taken into account. The boxes are static in that their resource requirements are viewed as a static number, a VM can for example be represented as a two-dimensional box where the height and length are equal to the CPU and memory demands respectively. The problem with this approach is that CPU and memory demands tend to fluctuate during the lifetime of the VM, which means that a static number is not a realistic representation of resource usage. The static number is mainly meant to make sure that VM is allocated enough resources to complete its workload without any bottlenecks or crashes, but might also lead to a waste of resources due to the number most likely being modeled on the peak resource usage of the VM. There is likely room for more efficient resource allocation if, for example, the peak resource usage only lasts for 1 minute while the total lifetime of the VM is 10 minutes. In reality, the resource usage profiles are more similar to puzzle pieces than rectangular boxes, and modeling them as such would enable far more efficient use of space on physical machines.

Figure 3 shows an example of the benefits of a more fine-tuned approach to a VM consolidation problem. The traditional way of seeing the resource demand as a static number is represented by the rectangular boxes, where two bins are needed to store all of three of them. Whereas the puzzle pieces represent the profiles modeled on resource usage that varies over time. The puzzle pieces fit neatly together in one bin.

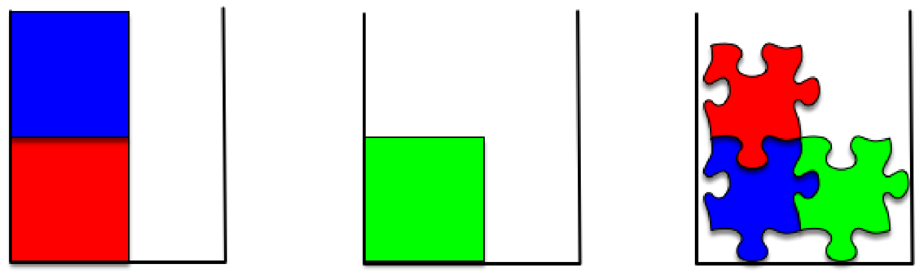


Figure 3: *Traditional bin-packing versus approach where resource demand varies over time. The traditional method needs two bins because the three boxes are too big to fit into just one bin. Letting the resource demand vary over time lets all three items fit into one bin by checking which ones are compatible.*

This method of modeling workload profiles could be referred to as a kind of temporal bin-packing, similar to the paper written by De Cauwer which was referenced in the literature review. In the case of this thesis, time is introduced in the form of letting resource use change over time and looking at the resulting pattern of resource use. This perspective opens up some new opportunities, but deciding which ones are feasible might require more technical testing. One could for example imagine a version of bin-packing where the amount of items that can fit into each bin changes over time, that avenue won't be explored in this paper but it could offer some interesting opportunities in the future.

The approach in this paper is to sum the resource use of one workload at time t with the resource use of another workload at time t and then repeat this for all timestamps. A threshold is chosen, for example 90% CPU load, and the workload profiles are assumed to be able to fit in the same "bin" if there is no timestamp where the sum is greater than the chosen threshold. This means that the general idea is to try to fit two compatible tests on the same server, but the concept could be extended to include more tests as well.

3.5 Procedure

Proceeding with exploratory research entails certain risks, mainly relating to the uncertain nature of such projects. This means that through the research, many possible paths to guide the project forward will emerge, thereby running the risk of choosing the wrong direction or investing in a direction and then changing direction later on. The lack of available research in the specific field gives little choice but to employ an exploratory strategy, as a comparative one requires objects that one can compare. This strategy increases the importance of guidance and discussion, as the path forward may not always be clear to the writer, and tunnel vision can easily occur. The lack of available research shows that there is room for new ways of thinking and improvement in cloud software testing, and the coming work will hopefully unravel new ways to facilitate better resource allocation for test-runners.

Chapter	**Description**
Results	The Results section starts by giving an introduction to the initial goals of the project and how they changed as time went on. Then it presents each of the steps the implementation went through, starting with monitoring, then modeling profiles, then the algorithm before mentioning some special cases and issues.
Discussion	Further explains the results from the previous chapter and discusses their implications in a broader context. Starts by discussing each aspect of the project, from the problem statement all the way to the end result. It also discusses the feasibility of the results, further testing possibilities and opportunities for future work.
Conclusion	Concluding remarks and reflections on the goals of the project and whether or not the problem statement was adequately answered by the thesis.

Table 1: *Approach used in the following chapters.*

4 Results

The goals of this project were subject to many changes, this chapter will explain how and why they were changed as a consequence of the results that were obtained. The main goal of the project was to define profiles that would be useful in facilitating more efficient resource allocation in software testing in the cloud, while this goal was important at first, the focus quickly shifted to the algorithm itself as time went on. Another goal was to find existing software tests used in large well-known applications and use them to form a baseline of expected CPU load in software testing, but this proved to be less important and more time-consuming than other tasks. There was a hope of implementing the algorithm from this paper along with a test environment like Jenkins or Gitlab to show the benefit of this method of VM scheduling. This proved to be a time-consuming task that might be better suited for future work that builds on this project. The algorithm itself and its possible implementations turned out to be the most important technical aspect of the paper, thus most of the other goals mentioned were explored in theoretical terms.

The first step of the process was to define the resource profiles themselves, as this could be done in many different ways and it was important to establish which method was best suited early on in the process. Profiles would have to be based on how the resource use changes over time, as opposed to the approach taken in bin-packing where a static number is the most common way of defining a profile. It would be optimal to base the profiles on several metrics at the same time, such as CPU, memory and network usage. This would give a result that is more representative of how the associated workload behaves in practice. CPU is used as the sole defining metric in the examples in this project, but the profiles are able to process memory as well. It was found that using more metrics was not necessary to illustrate the point of the paper, but the algorithm could be extended to include other metrics as well.

The next step after defining resource usage profiles was to evaluate whether or not it is feasible to use them to make an algorithm that can help a scheduler to fit more virtual machines on the same server. Different ways of gathering resource usage data were explored, as well as various methods of displaying the gathered data. After a method of data collection had been chosen, it was possible to compare data from different sources. This comparison could be used to find out if the profiles were compatible to fit on the same server. A resource usage threshold was chosen, for example 90% CPU load, and the usage of two profiles were added together to see if the total usage stayed below the threshold. If the sum was below the threshold at all times during the execution of both tests, then one can assume that it is possible to run those tests concurrently on the same server. The possibility of delaying the start of one of the tests was also added, this was beneficial when two profiles were not immediately compatible but could be if one of the tests were delayed so that computational spikes from both tests did not occur simultaneously.

The final product is a proof of concept of an intelligent scheduling algorithm that could be used to improve VM consolidation. This is in line with the goal of the project, as the main objective was to explore the usefulness of this kind of temporal workload profile in test scheduling. That goal was mostly achieved, but further verification of the results is something that should be done in a test environment like Jenkins or Gitlab and is suitable work for a future thesis.

4.1 Monitoring Resource Usage

Monitoring the resource usage of a virtual machine is fairly straightforward, but there are several factors that need to be taken into account. There are many different methods and possible interpretations that could affect the results, such as the hardware being used and the time between each measurement. In this case, the most important resource to monitor is CPU usage, which is usually measured as an average usage over a certain time.

Hardware

Performance is highly dependent on the hardware being used, thus it only makes sense to compare profiles that were tested on the same hardware. If a test is monitored and its resource usage is saved in a log file, the numbers in the file will only make sense in the context of the hardware it was running on. CPU load for example, is measured as a percentage of how much of the total capacity is being utilized at a given time, meaning the number would vary depending on the amount of computational power that is available at that time.

It would be ideal to do all data gathering on the same server under the exact same conditions in order to avoid any unwanted interference and variations in the data. This can be achieved by doing all calculations on my personal machine, the only issue with this is that it would likely entail a substantial amount of interference from other processes that are running in the background. For this reason, a virtual machine running in the cloud seems to be the best option if we are monitoring a real test. The VMs on the same data center are usually running on the same type of hardware even if they are on different servers. The VMs should have similar performance if they are of the same size and flavor. Running the program on a VM also has the benefit of allowing the user to choose the flavor, opening up possibilities in testing with varying amounts of resources. Most of the testing in this project ended up being done on my personal computer mostly for convenience and due to time constraints.

Resource monitoring in Linux

Most tools that help with CPU monitoring let the user choose the time frame from which to calculate the average usage, this is also how the built-in monitoring tools in Linux machines do their calculation. I considered using built-in tools such as sysstat or the top command, but encountered some complications. The top command gives the user a list of the ten processes that are using the most resources right now, but the main problem with this approach is that applications such as a Gitlab-runner spawns multiple processes when running a test pipeline. This means that it is difficult to determine the total resource usage of a pipeline, especially when using the Docker executor which spawns multiple containers for each pipeline. The command can be very useful if we want to check which processes are hogging the CPU or memory, but are unsuitable for logging the total usage.

Python

I ended up using Python to measure resource use, so Linux tools such as Sysstat were not needed. But the Linux tools were still worth mentioning to illustrate the different ways that resource use can be monitored, showing that some methods are more lightweight or otherwise specialized to suit different use cases.

Psutil is a Python library that offers a wide range of functions that can be used in the monitoring of resource usage and utilization. I opted to use Psutil to record CPU usage through a Python script. The script needs to be running while test simulations are being executed, during which it will record the percentage of CPU power being used and save it to a log file. The code snippet on the next page shows the script that monitors resource use and writes it to a text file. The script takes a threshold and a filename as arguments. The threshold is used to stop the recording once a certain amount of time has passed.

```python
def record(threshold, name):
    timestamp = str(datetime.datetime.now())
    filename = 'logs/{}/{}.txt'.format(name, timestamp.replace(' ',
    '-'))
    os.makedirs(os.path.dirname(filename), exist_ok=True)

    with open(filename, 'w') as outfile:
        print('Recording resource utilization...')
        start_time = time.time()
        while True:
            elapsed_time = time.time() - start_time
            if elapsed_time > threshold:
                print("Stopping recording")
                break
            outfile.write('{} {}\n'.format(psutil.cpu_percent(
    interval=1),
             psutil.virtual_memory().percent))
```

It is worth noting that most modern computers have multiple cores in their CPU, this includes the virtual machines used in cloud services. The architecture of the machine needs to be taken into account when calculating resource usage, as many monitoring programs report CPU usage of each individual core. This means that the CPU usage of a process is often shown as above 100%, which indicates that the process is using more than one core. In other words, a CPU with 4 cores can use 400% CPU. We only need to consider this fact if we are monitoring CPU usage of individual processes, not when monitoring the entire machine.

Setting up an environment where software tests can be executed in a realistic manner, such as in a Gitlab environment with multiple runners, turned out to be a time-consuming process. This led to a decision being made to simulate CPU load to generate some data that could be used to define example profiles. The data is sufficient for defining example profiles as long as it is similar to real-world CPU load statistics. For this reason, the data used in the examples further down is either generated through a script that does some mathematical operations many times or another script that generates some pseudo-random numbers that look like CPU load statistics.

Simulating CPU load

The nature of the actual processes being monitored is unimportant to the end result, as my interest lies in examining the possibility of using workload profiles to efficiently allocate resources. Test simulation is done in three ways, building a Docker container, running a script that generates some CPU load or running a script that generates some numbers that look like CPU load statistics. The time used by the simulations was fixed, as opposed to tests in real life that can use more or less time depending on various conditions during the execution. This was done in order to keep the examples as simple as possible while also being complex enough to illustrate the points being made.

Measuring CPU load from building a Docker container generated the statistics that can be seen in figure 4. the script calculates the graph based on log files found in a folder, it takes all of the files in the folder and produces a graph that shows the CPU usage from each execution of the logging script.

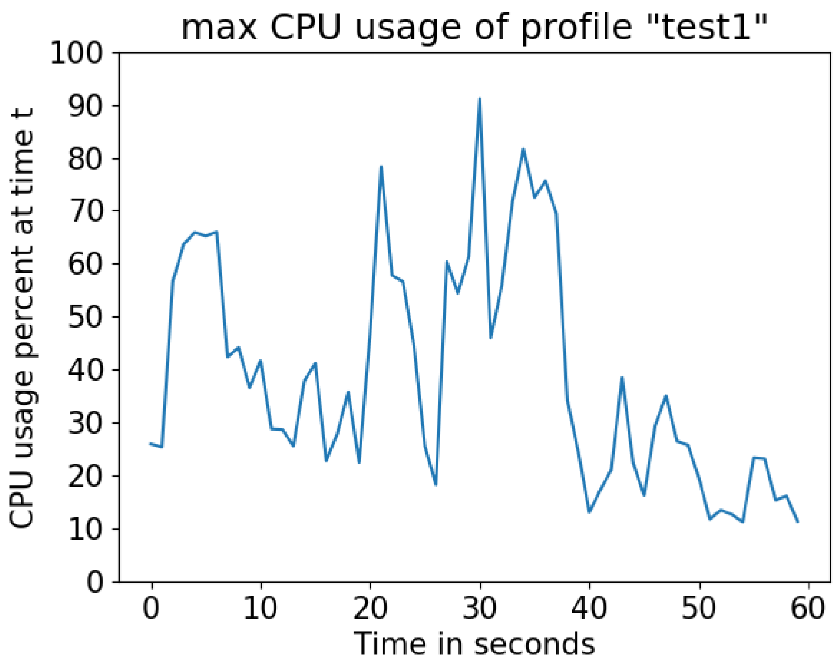


Figure 4: *CPU usage over time, this graph shows the maximum values registered for each timestamp over several executions of the same script. This graph shows the erratic nature that CPU load sometimes exhibits.*

4.2 Modeling

The workload profiles need to be precisely defined before they can be used to facilitate resource allocation. There are many ways to do this but the first step is always to decide which metrics to base the profile on. The two metrics that are usually considered when designing workload profiles are CPU and memory usage, but there are others whose inclusion may make the profiles more sophisticated and produce better results when the profiles are used in a resource allocation algorithm. Including more than one metric further complicates the task of comparing profiles and finding out which of them are compatible for concurrent execution, compared to the one-dimensional problem resulting from using just CPU load. The procedure is mostly the same but the task of finding a match between profiles becomes much harder, due to the fact that all metrics need to be within certain parameters at the same time.

Metrics

The metrics are modeled based on how they change over the runtime of the workload, this runtime varies depending on how fast the hardware is able to execute the program. One of the goals is to facilitate better resource utilization by allowing compatible tests to run concurrently on the same hardware without negatively affecting performance for either one. This means that it is necessary to record the time it takes for each test to complete and make sure that the runtime does not increase after implementing the new scheduling strategy in order to completely verify the results. It was found to be beneficial to base the profiles on several executions of the same workload, as that should give a better view on the typical resource usage.

The three main metrics that are usually considered when designing workload profiles, from most common to least common, are CPU, memory and I/O statistics. The profiles are meant to represent the assumed resource usage over the lifetime of a workload. Examples of workloads include various applications, such as web servers, online video games or software tests. Profiles that describe the resource usage of applications can be used to improve resource scheduling and efficiency in data centers by facilitating more accurate resource allocation. Considering multiple metrics instead of just CPU for instance, leads to more sophisticated profiles that may describe actual resource usage more accurately when compared to the one-dimensional version. Viewing a metric as a statistic that changes over time yields a graph where it can be displayed as a one-dimensional function *f*. Using CPU as an example gives a function that shows the percentage of CPU being used at time *t*.

$$f(t) = \text{CPU load at time t}$$

The metrics can be represented with three different strategies: using one metric, multiple metrics separately, or combining multiple metrics into a one-dimensional aggregate. Each strategy has its own benefits and downsides, with the single-metric one having the benefit of being the most simple and straightforward one while also having the least room for error. Using aggregated metrics, on the other hand, will lead to some inaccuracies as a result of errors in the calculation, such as rounding errors caused by the computer. Whether or not the benefits of more sophisticated profiles outweigh the added complexity and possible errors is something that needs further investigation.

Measuring a single metric such as CPU usage has the benefit of simplicity and a small chance of error in calculation and measurement, but the result may not be indicative of actual resource demand. Profiles derived from a single metric are not representative of total resource usage, as applications consume several resources at the same time and not just the CPU. While several metrics would be beneficial, CPU has been shown to be the primary bottleneck in cloud environments and is thus the most significant factor when predicting resource demand. This project seeks to make profiles that can help facilitate consolidation and allocation on cloud servers, in which case it would be beneficial to consider multiple metrics so that the profiles are as realistic as possible, but this being a short project means that certain compromises had to be made. The single metric profiles should be adequately representative of the real resource demand, but there is room for improvement through inclusion of more metrics. Even if the algorithm in this project mainly looks at CPU load, it can still be extended to look at several metrics. The results shown here are mainly meant as a proof of concept, and CPU is the most logical metric to use for that purpose since it is usually the most important metric to consider.

The d-dimensional bin-packing problem, brought up by Christensen et al. in [4] is especially relevant in the modeling of workload profiles. Using vectors to represent a profile with multiple resource requirements could open up a world of mathematical methods that may be utilized. If we are measuring m different resources, one can denote the resources as $r_1, r_2, ..., r_m$. Measuring the current resource usage n times gives the following set of resource vectors.

$$R_1 = \begin{bmatrix} r_1 \\ r_2 \\ . \\ . \\ . \\ r_m \end{bmatrix}, R_2 = \begin{bmatrix} r_1 \\ r_2 \\ . \\ . \\ . \\ r_m \end{bmatrix}, ..., R_n = \begin{bmatrix} r_1 \\ r_2 \\ . \\ . \\ . \\ r_m \end{bmatrix}$$

Which can be denoted as

$$R = \Big[R_1, R_2, ..., R_n\Big]$$

This way of modeling resource usage can be modified to fit the constraints of this project. The profiles are based on lists of resource usage, if for example the r_1 values represent CPU load then CPU load is stored a list of all r_1-values. r_2 could represent memory usage, thus creating a list of all r_2-values. This means that one profile could be defined by a unique set of resource vectors, R.

One of the benefits of modeling resource usage in this way is that it is then possible to see a distinct pattern for each workload by looking at the values of R. Similar to the key metaphor mentioned earlier, one can view the collection of vectors as a key. The point of the metaphor is to illustrate part of the goal of the project, namely to see if it is possible to fit the *keys* on top of each other by lining up the ridges so that they don't collide. In other words, matching workload profiles with each other based on corresponding r-values, a low r_1 value from one profile is matched with another with a high r_1 from another profile. This process is made more complicated by increasing m, as all of the r values have to correspond with each other at the same time. In order to limit resource usage, a threshold is chosen. This threshold could for example be 90% CPU load, the trick is then to make sure that the total CPU load remains below the threshold through the entire process. A profile with high r_1 and low r_3 at time a given time has to correspond with a low r_1 and high r_3 at the same timestamp for the profiles to be considered a match, the profiles are considered a match if the sum of their resource usage at any time remains below the chosen threshold.

The most common metrics to consider when designing workload profiles are CPU and memory usage, yielding a 2-dimensional profile with $m = 2$ where r_1 is CPU and r_2 is memory.

$$R = \left[\begin{bmatrix} r_1 \\ r_2 \end{bmatrix}_1, \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}_2, .., \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}_n \right]$$

Focusing on both CPU and memory utilization gives more realistic workload profiles, but there are still many other metrics that could be included to further improve the profiles. Metrics such as I/O statistics, network utilization and storage capacity are also important factors in virtual machine performance and should be considered if the goal is to have as realistic profiles as possible. Network utilization is more difficult to monitor, as the information received is usually the bytes sent and the bytes received. This could be measured as a single number by taking the rate of bytes sent versus bytes received, but the usefulness of such a metric is debatable. Introducing more metrics complicates the profiles, so the usefulness of each metric needs to be considered carefully before introducing them to the model. Due to this, I consider the CPU and memory utilization to be the most essential metrics and they are therefore the only ones to be included in this project, with CPU being the main focus. Disk usage could be included as well, but it does not seem to be essential for the goals of this project.

Disk usage can be useful in scheduling, as knowledge of the storage needs of a given pipeline can help the scheduling software allocate the appropriate disk size so that it can avoid the pipeline crashing due to a lack of available disk space. An important part of this project is to look at how the resource demand changes over time, and disk usage is most likely not going to change as much as for example CPU when running software tests. Network usage is also relevant when looking at software tests, as many tests are meant to examine how the application reacts to high network traffic and investigate how much bandwidth the application needs. Using too much bandwidth can lead to increased costs for the company hosting it, while too little bandwidth can lead to the application crashing at crucial moments of high traffic. This discussion of the value of each metrics leads to the greater problem of deciding which strategy to use when monitoring the metrics of a test runner. One can endeavor to monitor as many metrics as possible in order to obtain the most realistic models or limit the monitoring to a select few crucial metrics, thereby reducing the complexity of the models without losing too much of the information contained in the metrics.

Deciding which metrics to keep is usually not a simple process, but there are exceptions. CPU and memory are intuitively the most important metrics to consider, as the usage of those resources is highly volatile and potentially unpredictable when running applications. This unpredictability means that it is usually a good idea to allocate more CPU power and memory to an application than it actually needs, as this could potentially avoid crashes and other issues. Network and bandwidth also have this characteristic, as network traffic can be highly variable and depends on many outside factors that are inherently unpredictable. Measuring network usage is not as straightforward however, and one would also need to run tests that simulate network traffic in order to generate data that can be used in the profiles later on.

Statistical methods could be employed in the decision process of determining which metrics to keep and which ones to discard. There are some prerequisites to this method however, as one would need a substantial amount of data in order to make a model that could make the decision. It would be necessary to have data that show the effects of the various resources on the likelihood of an application failing or losing performance. Such data that can be hard to find or generate. But if one were to find data that fulfill these criteria then it would be possible to use regression or other supervised learning methods to examine the significance of metrics such as CPU or memory in predicting the performance of an application. If a metric is significant to the performance of the application, it is reasonable to assume that the metric is also important in resource allocation. If the metric happens to be insignificant to the performance of the application, then it is also reasonable to assume that it could be discarded from the model without losing too much information.

Using statistical methods to determine which metrics are important might be overkill since it is fairly obvious which metrics are needed, depending in the type of tests being run. Stress tests for instance, meaning ones that are intended to test how the application performs when receiving high traffic, need network information. So for those tests it would be beneficial to measure CPU, memory and network traffic to make accurate resource usage profiles. Other tests usually don't depend on the network to complete their main tasks, so bandwidth metrics are not as useful as CPU and memory. CPU and memory are important metrics for all cases and should be included in the profiles regardless of test type. Therefore most of the examples in this project will be concerned with CPU and memory, but they could be extended to include other metrics as well.

The amount of resources saved is often quantified by the reduction in the time it takes to complete the task, with a quicker execution entailing a greener operation. Time is not the only deciding factor when checking if the algorithm was able to improve the resource scheduling, with the goal of this project being to improve resource utilization it makes sense to look at the resource usage instead. Resource utilization can be improved without changing the time it takes to run a test, but it can still save time by letting the users run more tests concurrently on the same server. Again referring to the key metaphor from earlier, meaning that we try to fit more tests on the same hardware if their usage patterns are complementary to each other.

All of the previous points led to CPU being the sole defining metric for this project, with support for processing memory being included in the profiles as well. The CPU load is measured by executing the same workload several times and then using either the average CPU load or the maximum value registered at each time. Since the time used is fixed in the simulations, the maximum values are found by taking the highest value found at each timestamp and returning a list containing all of the highest values. The average values are found by taking the sum of all the observations at each respective timestamp and dividing by the number of times the simulation was executed.

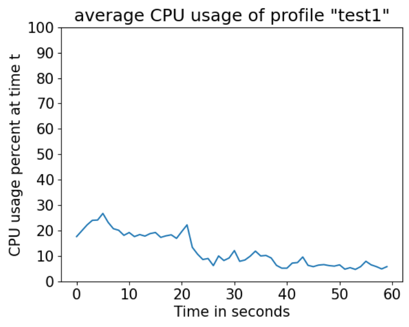
Establishing Typical Resource Usage For A Test

Variations in resource usage are common, depending on the state of the physical machine, other applications running at the same time and other factors. This variety in the results means that basing a profile on a single test execution´s resource usage may not give an accurate description of typical resource usage for that test. The uncertainty can be counteracted by measuring the resource usage of several runs of the same test and aggregating the results into a single measurement that can be used to model the test´s behavior. An aggregate can account for variations in the data and there are several methods that can be used, each suited to different conditions. Running the test several times and taking the average of each timestamp may give more insight into typical resource usage, but is unsuited to some cases. Large variations and spikes in the data can skew the average to one side or the other, giving some misleading results. It is therefore prudent to examine the data before deciding on which method to use. There are other alternatives to using the average, such as the median, mode or maximum value.

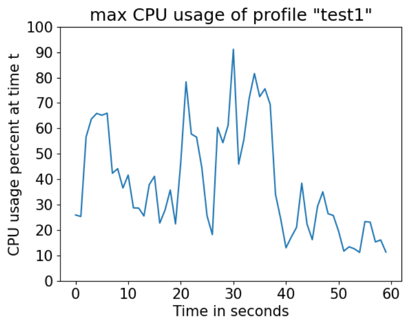
The median does not really give that much information about resource usage so it is most likely unsuitable in this case. It is important to know how high resource usage can get during test execution to prevent bottlenecking and throttling. The median does not provide this information, the average might be more indicative of this. Taking the highest value found among the measurements and using that as a baseline for the profile is the safest option. It is reasonable to assume that using the maximum value can prevent throttling, but it may make it more difficult to find compatible profiles. Maximum value can be used as a conservative estimate for the actual resource usage and is much safer than using the average value. Whether or not it is beneficial or necessary to use the maximum value as an estimate depends on the type of application being hosted, as some have much higher spikes than others. Online video games are an example of applications that would benefit from this method, as they can have unpredictable high spikes in traffic. This stems from the unpredictable nature of user-driven traffic, the traffic can be predicted to a certain degree by looking at which time during the day usually has the most traffic but there can always be sudden unexpected spikes regardless.

The maximum value can be found by taking all of the samples and using the maximum value for each timestamp. The resulting list contains a compilation of the highest resource usage found for each time t. Using these values decreases the likelihood of critical errors, the most likely error would be that the tests use fewer resources than predicted, which is not a critical issue.

Maximum values can be used as a conservative estimate, and is unlikely to result in crashes from underestimating the total CPU load. While the average values might be more similar to the real resource usage they are also risky to use due to the increased risk of miscalculation. Both estimates are valid ways of representing resource use, and their differences might be better understood by looking at figures the figure in 5. Both graphs are based on the exact same data, except one uses the average while the other uses the maximum values. It is easy to tell the graphs apart and each of them would produce different results when used in PAST.



(a) *Average CPU usage of a test based on mock data. The graph is much smoother and less prone to spikes when compared to the graph based on max CPU usage.*



(b) *Max CPU usage of a test based on mock data. This graph has a lot more spikes and dips when compared to the one based on average CPU usage.*

Figure 5: *Max vs average CPU load displayed in graphs that show their patter over time.*

Gathering Metrics

There are many ways to gather metrics about system performance while running software tests, some with more overhead than others. The challenge lies in monitoring resource usage while also running tests without having the monitoring script or other processes interfere too much with the resource usage. Tests need to be simulated many times in sequence while a script records the resource usage of each test. The logs need to be separated so that it is possible to distinguish between different tests when the workload profiles are being created.

All of this can be automated in Python, resulting in a single script that starts recording resource usage, then simulates a test and stops the recording once the test has finished. There are many ways to simulate tests, as the only thing that is really needed is a way to generate some resource usage. CPU load can be generated in many ways, one of which is simply to do some math a set amount of times, the script can also tell the computer to idle for a set amount of time to simulate low CPU load. A shell script that is executed from a Python program is suitable here, this method allows one Python program to generate CPU load and record the CPU usage at the same time. The advantage of simulating resource usage in this way is that it enables the customization of results, which can be used to illustrate tests with specific properties.

The shell script that simulates CPU load is very simple, see the snippet below this paragraph. The script does a simple calculation $2000 * 10000$ times, this creates some spikes in CPU performance that are similar to the ones seen when the computer is running tests. The Python script that executes the shell script also includes a function that records the CPU load over a set amount of seconds, the default being 60 seconds. The interval between CPU measurements can be customized, with every second as the default interval. Measuring the CPU load often gives a more fine-grained look into the performance of the computer, but results in larger log files.

```
for n in {1..10000};
do
    for m in {1..2000};
    do
     s=n+m
    done
done
```

Finding a way to measure CPU and run the simulations at the same time was a challenge, Python´s threading module was found to be a suitable solution. This allowed the script to start the function that records the resource usage before starting a thread that runs the script that generates CPU load. The same script can be used to run more accurate simulations, meaning actual tests or building real applications, by changing the function that is executed by the thread. The function from the threading module takes a function as a parameter, so it is easy to run different simulations (see snippet below). The code runs the simulation and records CPU load a set number of times, numerous executions of the code are needed to account for variations in CPU usage.

```
def main(n_samples, name):
    for i in range(n_samples):
        t1 = threading.Thread(target=simulate_cpu)
        t1.start()
        record_cpu(60, name)
        t1.join()
```

A more sophisticated alternative to just generating CPU load through a shell script would be to perform operations that are often performed during tests, such as building containers and applications. There are many open-source projects that can be downloaded and built locally, this could be used to more accurately simulate the CPU load from software tests. One approach to this is to make a Dockerfile that downloads and builds an application, which is a very common part of CI/CD pipelines. It is logical to assume that many tests have the highest resource demand at the start of their execution, therefore using Docker to build an application should be fairly similar to running a test on that application when it comes to resource usage.

This idea was explored to a small degree through gathering some data by having Docker build a simple container and recording the CPU and memory being used. The container built a Java application, this was done because of the widespread usage of that language and the fact that building a complex Java application can be a demanding task for a computer. Gradle is a very common build tool for Java applications and is used by many companies worldwide. This leads to it also being a common part of testing pipelines, as the applications need to be built before they are tested. The build part is a test in itself, as the developers need to make sure that the application can still be built after they have made changes to the code. There are many open-source projects on Github that can be cloned and downloaded for this purpose, for example Mockito, which was used for some experimenting in this project. The choice of application was mostly arbitrary, as any Java application combined with a build tool such as Maven or Gradle would be adequate for testing purposes.

More data was obtained by generating random numbers between 0 and 90, with certain constraints. This was done to construct specific cases that could illustrate the functionality of the algorithm without having to spend an inordinate amount of time tracking down applications that could generate specific patterns of CPU load. More specifically, it was used to make two profiles whose

CPU usage followed patterns that were inverse of each other. An example of two such profiles can be seen in figure 6, where the profile ”high_low” has high then low CPU load and ”low_high” has low load followed by high CPU load. A further exploration of this case can be found in the section titled PAST.

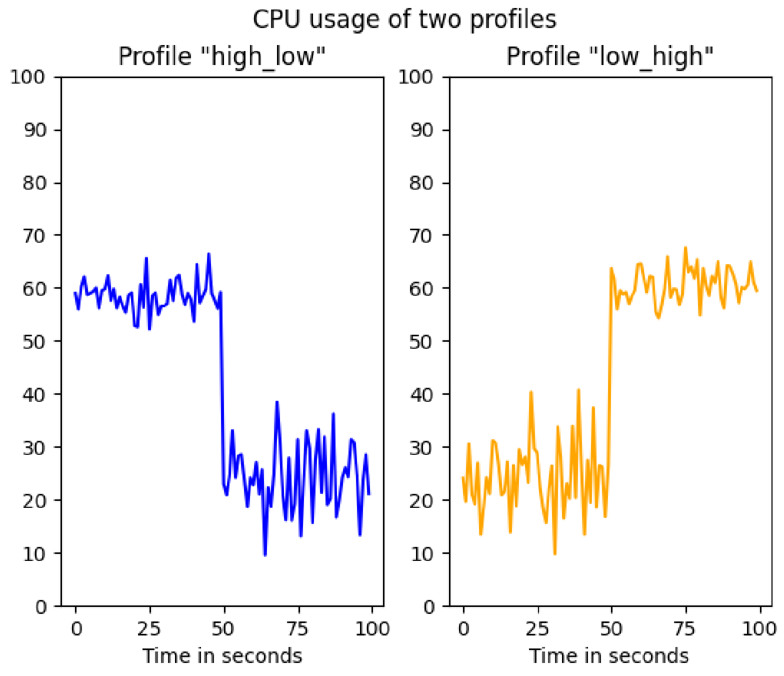


Figure 6: *Two profiles that are based on artificial data. The profiles have inverse patterns of CPU usage, the first having high then low usage and the second having low then high usage.*

Technical Representation of Profiles

The code in this project was developed in Python and Bash, the bulk of it being Python code as the Bash scripts are mostly there to generate CPU load. The profiles that were outlined in the previous section can be represented as instances of a Python class, where each class contains all of the metrics that were measured for that software test. This means that each instance of the class represents a software test, ideally based on multiple executions of the same test. The class needs to have an identifier that can be used to check what kind of workload it represents so that it is possible to differentiate between the instances when comparing them to each other.

Object-oriented programming is a logical choice of method concerning technical representation of the models, as each model could be represented as a class that contains information about resource usage for that test. Programming the profiles in this manner means that it may be possible to reuse them in the future through other applications in an API, since object-oriented code is by nature more modular and extendable than other code. This is why I chose to model the profiles using classes, a snippet of code which shows how the profiles are defined is shown on the next page. The profile takes a directory, a name and a boolean as arguments. It then calls a method to process the log files found in the given directory, these files can contain CPU and memory or just CPU. The contents of the files are stored in the lists defined in the constructor.

```python
class Profile:
    def __init__(self, dir, name, include_memory=True):
        self.name = name
        self.include_memory = include_memory
        self.cpu = []
        self.memory = []
        self.matches = []
        self.start = 0

        self.process_logs(dir)
        self.l = len(self.cpu)
```

The profiles should be flexible in the amount of information they can receive from raw data, as some log files may contain more metrics than others. Some metrics may be more useful than others depending on the test in question, network statistics are for example useful for some test types but completely irrelevant for others. The data needs to be formatted correctly in order to be processed by the algorithm. The metrics need to be stored in a text file where each line represents the metrics at time t and different metrics are separated by a whitespace. The current algorithm assumes that there are two metrics, where the first one is CPU load and the second is memory usage. The source of the metrics does not really matter, as all the algorithm needs to check is that the sum of two metrics from two tests at time t does not exceed the chosen threshold. CPU load is used in most of the examples but memory can also be included by setting the "include_memory" flag when creating an instance of the Profile class.

4.3 PAST

Imagine that there is a queue of tests, some of them are computationally demanding while others are not. The tests that are not dependent on each other are often executed in parallel on separate servers. They are executed on separate servers because it is assumed that there are not enough resources to run them in parallel on the same server or simply because we do not consider the possibility of running them on the same server. The assumption is based on measurements of the resource usage of similar programs where the sum of their resources would be too high for one server. The problem is that the assumption is often based on a single number, meant to represent the resource usage of that program. However, resource use tends to change over time, and if we were to represent the resource use as a pattern instead of a single number we might be able to place the programs on the same machine in such a way that the total resource use does not exceed the limits of the machine. This is what the algorithm tries to achieve, by examining and comparing the patterns from different profiles and then finding a start time for the tests so that their spikes in resource use don't coincide.

Workload profiles need to be comparable if we want to find out which ones are compatible with each other. The comparison can be a lengthy process, as each profile needs to be compared with every single other profile, meaning that if we have n profiles the program needs to make n^2 comparisons. One approach to this problem is to have each workload be represented as an instance of the Profile class and make a list with all the instances before iterating through the list and comparing all of the elements with each other.

There are many ways to compare profiles and the process becomes more complicated when more metrics are included in each profile. The main problem remains the same no matter how many metrics are introduced, namely deciding which profiles can fit on the same hardware at the same time. This can be decided by "stacking" the profiles on top of each other and seeing if the resource usage exceeds a predefined threshold. If the profiles can be stacked on top of each other, it means that their corresponding tests can probably be executed simultaneously without crashing the server. This concept is similar to the "key" metaphor mentioned earlier, meaning that if the two profiles are viewed as keys, we can check if they fit together by lining up the ridges on the keys so that they fit neatly together. Two profiles that are intuitively compatible in this manner can be seen in figure 7. Figure 8 shows the result of adding the CPU load of the two profiles together.

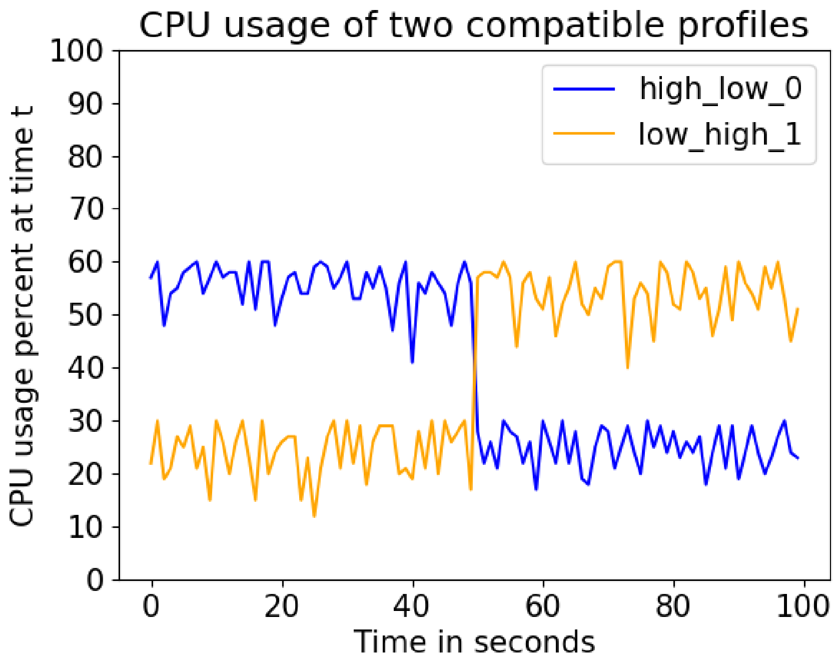


Figure 7: *Example of two profiles that fit neatly together. The first profile, colored in blue, has high CPU load at first and low towards the end. The second profile, covered in orange, has low CPU load at first and high towards the end.*

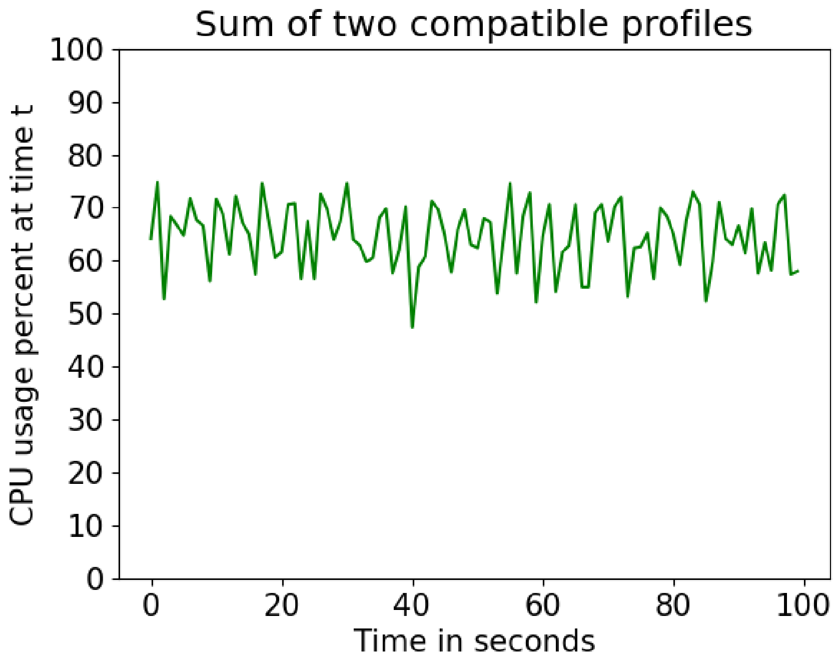


Figure 8: *The total CPU usage if the two profiles were to run in parallel. This is a case where it is obvious that the profiles are compatible, as the sum never reaches the threshold of 90%*

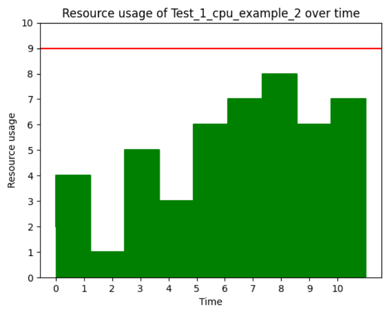
Representing the resource use as a function that takes the current time as a variable and gives the resource use for that timestamp makes it possible to compare the resource use of several profiles at different timestamps. This can be displayed as a graph where the x-axis is the timestamps where resource use was recorded and the y-axis is the resource usage. Shifting a profile to the right or left along the x-axis can help save resources even if the profiles are not initially compatible. The resource usage of a profile can be represented as a list of numbers, the profiles can be stacked on top of each other by summing each element of the list from one profile with the corresponding element of the list from another profile. This method assumes that the lists are of equal length, but the algorithm can be altered to include lists of varying lengths. If one of the pairs of elements sum to a number above the chosen threshold, we can shift the associated profile to the right. Meaning that if elements at index i are too large, we can try with index $i + 1$ at the first list and index i on the second. This process can be repeated until a version that does not exceed the threshold is found or the profile is shifted all the way outside of the range of the other profile.

A Simplified Example

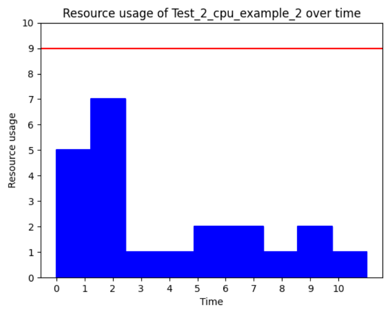
Similar to bin-packing, the resource usage at time t can be viewed as a rectangular box where the area of the box is for example the CPU usage. By placing the box along a horizontal line, representing the time axis, other boxes can be stacked on top of each other to illustrate the resource usage if they were to run simultaneously. If the height of the stacked boxes exceeds a pre-determined threshold, the boxes on top are shifted to the right. After they have been shifted we check again if the new height is below the threshold. If all boxes are below the threshold we assume that the tests can be executed simultaneously. If the boxes had to be shifted to the right, the number of steps shifted is kept track of, as that number represents the number of seconds or minutes that the second test has to wait before starting.

The examples in figures 9 and 10 are constructed to illustrate the case where the tests are shown to be compatible on the first iteration of the algorithm, but this is often not the case. Also, many tests start with something being installed and thus tend to have a CPU load at the start of their execution. If the algorithm encounters a case where the total resource usage at time t exceeds the threshold, it will try to shift test 2 to the right. Meaning that if the first iteration does not yield a match, it will then try to start an iteration by comparing test1[1] with test2[0], test1[2] with test2[1], and so on. If this does not work it tries starting with test1[2] and test2[0], it repeats this process until a match is found or test 2 is shifted all the way to the right outside of the of range test 1. Shifting test 2 all the way outside of the range of test 1 means that the tests are executed sequentially, showing that the tests can not be run at the same time on the same hardware with the constraints as they are.

The graphs in figures 9 and 10 are simplified examples of how the CPU usage of software tests can change over time, showing how different tests have different patterns in their resource usage. The graphs in figure 9 seem to show that the two tests can be run at the same time, since their resource usage patterns are complementary to each other. Stacking the graphs on top of each other, like in figure 10, reveals the total resource usage if we were to run the two tests at the same time. The resource usage is below the threshold at all points, if only barely, which shows that the tests can be run at the same time on the same server without adversely impacting performance.



(a) *Test 1 CPU usage*



(b) *Test 2 CPU usage*

Figure 9: *Example CPU usage of two tests. The tests seem to be compatible upon first inspection, as their resource usage patterns are inverse.*

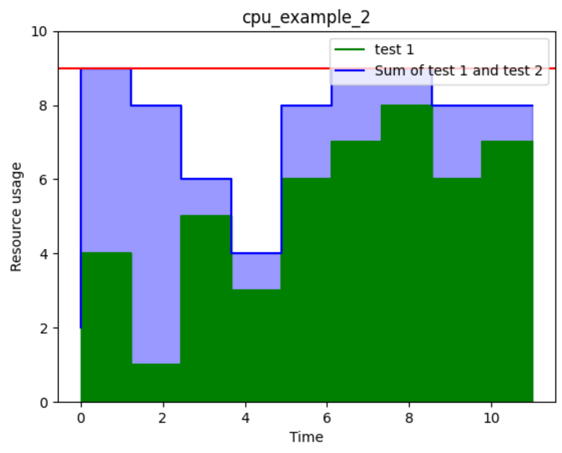
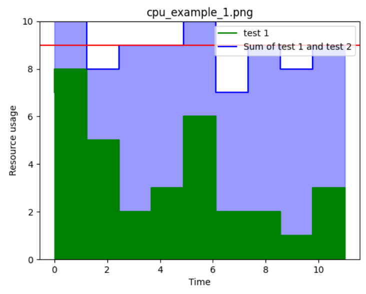
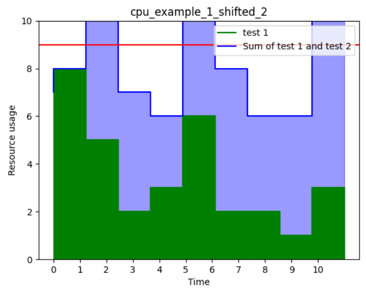


Figure 10: *Test 1 and Test 2 together, with test 2 stacked on top of test 1. The dark green represents test 1, while the light blue is the sum of test 1 and test 2.*

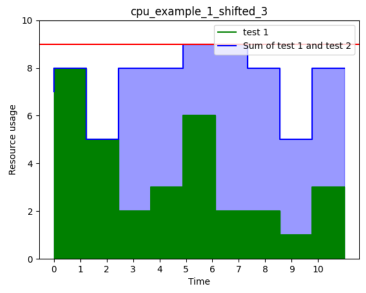
The case in figure 9 is not realistic, and reality often demands that one of the profiles must be delayed to avoid exceeding the threshold. The process of shifting a profile towards the right, which was explained in the paragraphs above, is illustrated in figure 11. The first plot shows how merging the resource usage patterns of two incompatible profiles could look, with the blue section being the sum of the two profiles while the green section is Profile 1 alone.



(a) *Combining profiles without shifting*



(b) *Profile has shifted been shifted to the right*



(c) *Profile 2 has been shifted another step to the right.*

Figure 11: *Illustration of how shifting one profile to the right can avoid excessive resource usage. The blue section represents the sum of resource usage of Profile 1 and 2, showing that the graph from Profile 2 has been stacked on top of Profile 1. Figure (d) shows that delaying Profile 2 by several steps, it is possible to start it while Profile 1 is running without exceeding 90% CPU load.*

Applying PAST to a test schedule

An example test schedule T consists of four tests arranged as [high_low, low_high, t1, t2], the first two tests are the same ones mentioned before in figure 6, while the two last tests are new ones made from synthetic data. Figure 12 shows the CPU load of the first two profiles, while 13 shows the last two. Processing them in the PAST script reveals that the first two profiles can be executed at the same time, given that the second one waits 49 seconds before starting. It also compared the second and third profiles and found that they could not run together at all. While the third and fourth were found to be completely compatible, as in they can start at the same time and run in parallel without exceeding the CPU limit of 90%. All four profiles are based on synthetic CPU load data, as in the data was generated to illustrate this example Each test takes 100 seconds to complete, and the CPU usage patterns were found by generating the data five times and taking the average values.

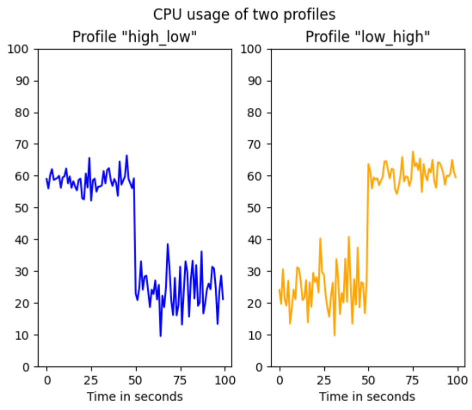


Figure 12: *Two profiles before PAST has been applied. They are scheduled to run sequentially even though they most likely could be executed in parallel.*

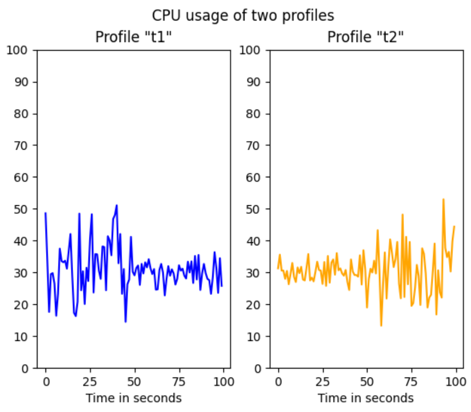
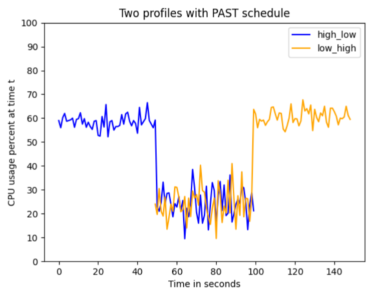
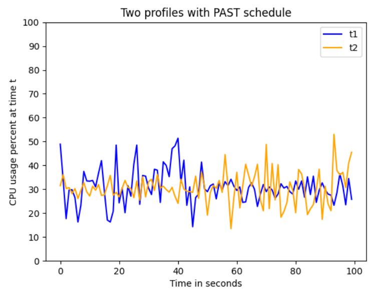


Figure 13: *The CPU load of the other two tests in the pipeline. The tests are seemingly compatible, as they both have fairly low CPU load and low values of t1 correspond with high values in t2 and vice versa.*

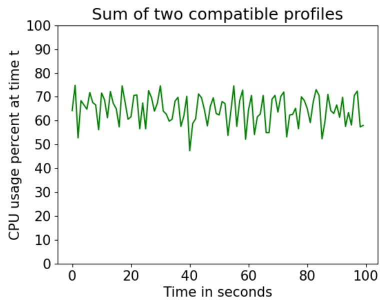
Figure 14 shows each of the pairs of tests from T after applying PAST to the schedule. PAST compared the profiles in a pairwise order, first high_low with low_high, then low_high with t1, and finally t1 with t2. It found that the runtime of the first pair can be shortened by letting low_high start after high_low has been running for 49 seconds. In the next step, it found that low_high and t1 were incompatible, as in they can't run at the same time on the same server without exceeding a CPU load of 90%. The final pair, t1 and t2, were found to be completely compatible and can start at the same time without using too much CPU. All tests in T take 100 seconds each, and the total runtime which would be 400 seconds is shortened by 149 seconds after applying PAST.



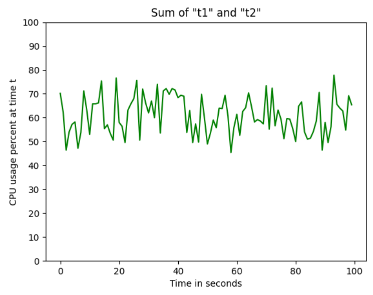
(a) *Profiles high_low and low_high plotted together. Low_high is able to start after 49 seconds instead of having to wait for the other profile to finish*



(b) *T1 and T2 plotted together. T2 is able to start at the same time as T1, thereby reducing the time used by 100 seconds.*



(c) *The sum of CPU load from high_low and low_high, notice that the sum is below 90 at all times.*



(d) *The sum of T1 and T2, notice that this sum is also below 90 at all times.*

Figure 14: *Test schedule with two profiles after PAST has been applied. Application of PAST showed that the profiles could not run completely in parallel, but the low_high profile can start before high_low is finished.*

The original schedule and ordering of tests in T did not change after applying PAST, but the timing of when the tests are scheduled to start does change. Both low_high and t2 start earlier than they would otherwise, thereby shortening the time it takes to complete the pipeline without causing too high CPU load. See table 2 for the specific numbers on this.

Tests	**Normal**	**PAST**	**Improvement (%)**
High_low and low_high	200s	151s	24.5%
t1 and t2	200s	100s	50%
Entire pipeline	400s	251s	37.25 %

Table 2: *This table shows the time taken to complete the pairs of tests with and without PAST. The final row shows the time it would take to complete the entire pipeline*

The Coding and Logic of PAST

Resource usage can be measured at different intervals, such as every second, every five seconds or every minute. Measuring every second gives the most accurate description of resource usage, but results in much larger log files if the metrics are measured over a longer period of time.

Imagine that there are two lists of CPU load from two different profiles, the following steps show how the algorithm works. The starting index in the second list is always zero, but the starting index of the first list is continuously incremented if the profiles do not match. If the starting index of the first list is i, it is incremented until i is equal to the final index of the second list or is out of bounds. The second profile is shifted to the right each time the starting index of the first list is incremented. Shifting the profile to the right by x elements essentially means delaying the execution of the test by x intervals, for example if the recording interval is every second the execution would be delayed by x seconds. This method saves resources and potentially time by allowing the tests to be run at the same time and is flexible in the way that it adapts to tests that would be incompatible if they were started at the same time.

The goal of the algorithm is to find a profile combination where the following is true; for any combinations of timestamps t_1 and t_2 and threshold m there is a sum $s = r_1[t_1] + r_2[t_2]$ where $s < m$. r represents the resource usage of the given test. The first iteration of the algorithm starts with $r_1[0] + r_2[0]$ and increments the t-values until it finds a sum that exceeds the threshold or runs out of items to compare. If a case where $s > m$ is found, it tries starting with $r_1[1] + r_2[0]$ before iterating through the lists again, but it does not reach the end of r_2, it only reaches $r_2[n-1]$. If that iteration also encounters a sum that exceeds the threshold, it tries again starting with $r_1[2] + r_2[0]$ and ending with $r_1[n] + r_2[n-2]$ and so on.

The first version of the algorithm used a recursive function, whose code can be found in the appendix. It was eventually replaced by an iterative function, which is shown below the next paragraph. The algorithm follows the principle of "first fit", meaning that it stops when it founds a spot where the profiles are compatible. This principle is used instead of other principles such as "best fit", because the first spot found is also the one that entails the shortest execution time, so it doe snot need to keep searching after finding the first fit.

The iterative version uses a while loop instead of recursion, it was found to be less prone to errors and was easier to troubleshoot compared to the recursive function. It iterates through both lists and increments the starting index of a if the sum exceeds a threshold. It returns a boolean that states whether or not the two profiles are compatible as well as an index that represents how long the second test has to wait before starting. Troubleshooting an iterative function is often easier than recursive ones, as recursive functions can be difficult to interpret and more prone to errors. See the code snippet on the next page for more details.

```python
def PAST(a, b, threshold):
    i = 0
    j = 0
    start_index = 0
    compatible = False
    while (i < len(a) and j < len(b)):
        compatible = True
        sum = a[i] + b[j]
        if sum > threshold:
            compatible = False
            i += 1
            j = 0
            start_index = i
            continue
        else:
            i += 1
            j += 1
            continue
    return compatible, start_index
```

The while loop starts over if a sum that exceeds the threshold is found, but only after adding 1 to i and resetting j. If no sum was too high then both indexes are incremented and the while loop proceeds to the next iteration. Finally it returns a flag stating the compatibility of the profiles and the index where profile b has to start.

Tests can often be executed in parallel on separate machines, so the focus of this algorithm is to save resources while trying to keep the runtime as low as possible, but the runtime might have been even lower if the tests were to run on separate servers. This means that the algorithm does not necessarily save time as it might delay the execution of a test in order to avoid high CPU load. The consequence of this fact is that the algorithm can actually make the pipeline slower if that test would otherwise run in parallel on a different runner. The decrease in resource usage is dependent on whether or not the increased consolidation enabled by PAST is able to counteract the loss of complete parallelization.

The fact brought up in the previous paragraph highlights the relationship between this project and green cloud computing, in that it tries to reduce the number of active servers while also keeping the runtime as low as possible. It is possible that the tests could take longer to complete after applying PAST if the tests were scheduled to run in parallel on separate servers. That problem is avoided entirely if the tests being executed in tandem on the same hardware are from separate pipelines. This means that even if one of the tests starts after the other, the time until the completion of the pipeline is not negatively impacted. The prerequisite for this is that each test "knows" where other tests are being executed and which tests it is compatible with, it can then be moved to a machine where it knows that it will be able to run without impacting the other test that is running there.

4.4 Special Cases

There are always special cases where additional steps need to be taken to ensure an acceptable result, but whether or not measures are taken to account for them depends on their relevance and how often they occur. Some cases would also take too much time to fix and is therefore omitted from the algorithm. The most obvious special case in this project is the case of dependencies between tests in a pipeline, this occurs when there are dependencies between the code components that are being tested. Such tests can not be started at the same time and must wait for the other to finish before starting. To illustrate this problem imagine three tests, T1, T2 and T3, where T3 depends on T2 and T1 is independent. T2 can be started before T1 has finished, but T3 must wait for T2 to be finished before it can start. This issue would need to be dealt with if there are dependencies present in a pipeline, as there often is, because otherwise the algorithm might try to start T3 before T2 has finished or even started. There are two possible solutions to this issue:

1. Join dependent tests and view them as one singular test. In this case T2 and T3 would be viewed as a singular test from the outside and analyzed as such in the algorithm. This means that the new composite test, $T2^*$, can be started before T1 has finished.

2. If a suitable spot is found where T2 can start before T1 is finished, it can do so but under certain conditions. T3 would still have to wait for T1 and T2 to finish but it would be possible to save time by running T1 and T2 at the same time.

These solutions were originally intended to be included in the algorithm but were omitted due to time constraints. Option 1 is the most simple solution here and leaves less room for error, but option 2 has the potential to save more time. Option 1 also has the benefit of being easier to implement. In a general sense, the simplest solution is often the best, so option 1 seems most viable in this case. The problem can be illustrated with some simple graphs, see figure 15. The blue block represents T1, while the green represents $T2^* = T2 + T3$. Figure 16 illustrates option 2, where T2 is executed concurrently with T1 while T3 waits until we know that T2 has finished. The indexes where T2 starts and stops are marked in order to highlight when T2 is finished, T3 is free to start any time after we have passed the index where T2 stopped.

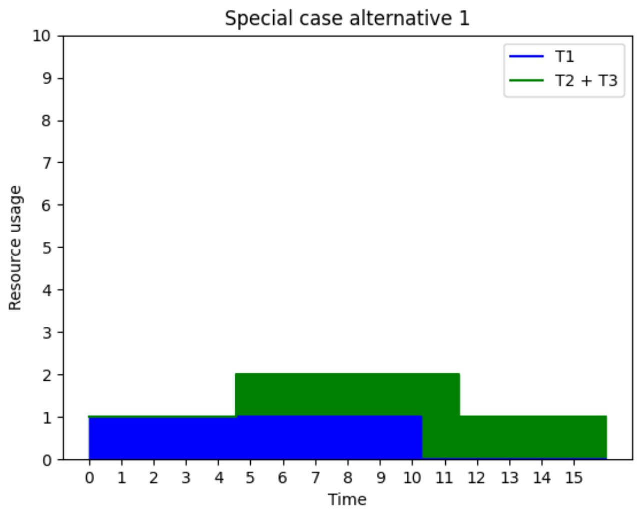


Figure 15: *Option 1, joining T2 and T3*

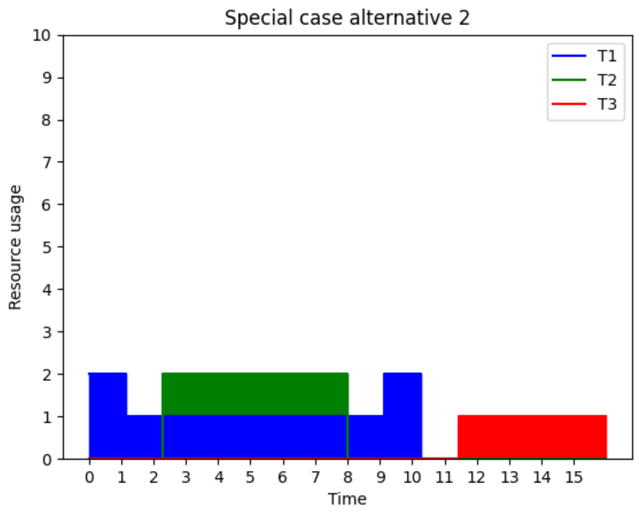


Figure 16: *Option 2, finding a spot for T2 within runtime of T1*

4.5 Issues

Measuring CPU in percentages means that it is difficult to determine if it is possible to run several tasks at the same time, as most tasks tend to use a lot of CPU at times. Multiple cores can mitigate this, but different hardware also leads to different execution times. For this reason, all tests being compared should be executed on the same hardware, ideally on a computer with several cores as that is the most realistic scenario. Processes frequently use resources from more than one core of the CPU, which means that it is possible to use the workload profiles to make sure that a computer can run multiple tests at the same time without impacting performance.

As mentioned previously in the paper, tests can be run in parallel on different servers and usually are if there are servers available. This saves a lot of time and is essential in some pipelines in order to reduce the time it takes to deploy an application to the internet. The algorithm in this paper aims to decrease resource usage by allowing certain tests to run in parallel on the same hardware, but it is often necessary to delay one of the tests in order to avoid conflicts in resource usage. This means that it might take longer to complete the tests by using the algorithm in some cases, as many of them would usually run entirely in parallel.

Resources might be saved by avoiding having to start up a VM on a different server and instead stacking more VMs on the same server, but the increased execution time might negate this. Longer execution time means using more resources and whether or not this outweighs the resources saved by consolidating VMs is uncertain. This issue only occurs when running tests from the same pipeline, it is not relevant when talking about tests that come from separate pipelines. Separate pipelines usually don't know anything about other pipelines, so they do not know if they can run on the same hardware. The VM scheduling programs in the data center knows how much computational power is available on a given physical machine and can schedule VMs to start where there is sufficient power available. But the VMs are viewed as static boxes defined by pre-determined resource limits, such as limits on CPU and memory.

This project introduces another dimension, time, in that CPU and memory can change over time. VMs that were previously thought of as incompatible by the scheduling algorithms can be scheduled to run on the same hardware if their resource usage patterns allow it. If the developers know that a program has a memory usage peak of 3gb they know that they have to provision a VM with at least 3gb of RAM, but the VM does not use 3gb all the time. It might use 3gb at one point but under 1gb the rest of the time. Another VM might use 4gb at it's peak, meaning that it needs a VM with at least 4gb of RAM. A server with 6gb RAM would be considered too small to fit these two VMs at the same time, but this might change if a more fine-tuned approach was used. If one makes sure that the peaks of memory usage from the VMs do not occur at the same time, it would still be possible to run those two VMs on that server.

This problem highlights the main contribution of the algorithm proposed in this paper, namely the ability to run tests that would otherwise run separately of each other on the same server. Parallel tests from the same pipeline might make the inclusion of the algorithm entail more computational resources being used, as the time it would take to complete the tests might increase in a some cases. The jury is still out on whether or not there is a net gain in performance when using the algorithm in those cases, as more testing is required to see if increased resource use from longer execution time outweighs the decrease in resource use resulting from VM consolidation.

The current implementation of the algorithm only supports pairwise processing of tests, not entire pipelines. It looks at a pair of tests and checks if they can be executed simultaneously by combining their CPU usage patterns and checking if it is below the threshold. Using the algorithm in a pipeline would mean checking all of the tests and finding a schedule where they can be started at the same time. The problem arises if a match is found between two tests, test 1 and test 2, where test 2 can start after test 1 has been running for 5 seconds. This changes the resource use pattern and time used by test 2. This could again change the compatibility of test 2 with another test, test 3 for example. It would then be necessary to compare test 3 with new pattern that arose from the combination of test 1 and test 2. The algorithm does not currently support such cases, as it only checks two algorithms and does not consider the rest of the pipeline or the effect that a merging of two tests has on it.

4.6 Summary of Results

The result and how it fits into the bigger picture of cloud software testing can be seen in figures 17 and 18. As stated before, the final result is not a complete implementation of a scheduling algorithm but rather a proof of concept that can be expanded upon, and through expansion might be used in other applications that perform scheduled tasks as well. The belief is that PAST can be especially helpful in improving test scheduling in CI/CD systems such as Jenkins or Gitlab, exactly where the algorithm would fit is shown in figures 17 and 18. Figure 17 shows a simplified view of how a CI/CD system operates. Developers push code through commits into a central repository which is managed by Gitlab in this example. The repository has a pipeline of tests built into it which is triggered every time a commit is pushed to the repository. The tests are arranged in a schedule and sent to the runners for execution. The runners are monitored and both the results of the tests and various statistics about the runners are sent back to the repository.

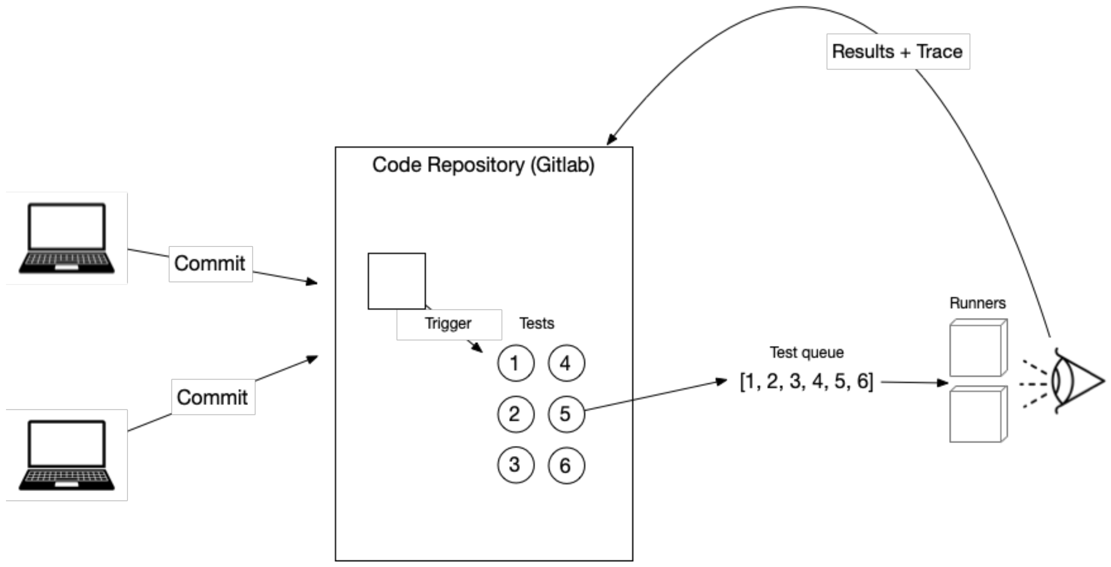


Figure 17: *Illustration of a repository with a built-in CI/CD pipeline. The developers commit new code to the repository, which then automatically triggers a series of tests that are executed on two runners. The runners are monitored and the results of the tests as well as resource usage metrics from the runners are sent back to Gitlab.*

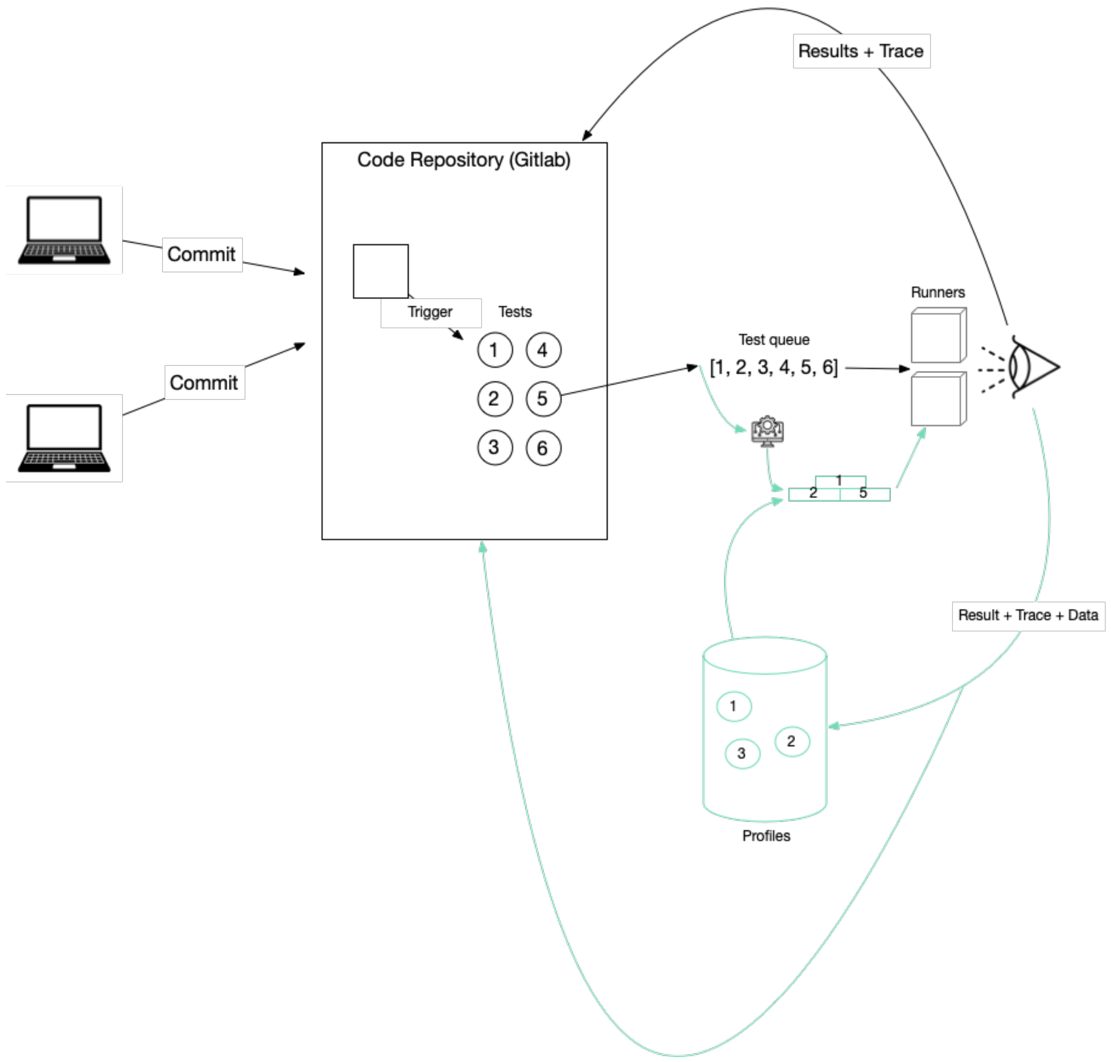


Figure 18: *Illustration of the same repository as figure 17 with added functionality from my algorithm shown in green. This figure shows how the algorithm may be implemented into an existing test environment such as Gitlab. The algorithm intervenes before the tests are scheduled and chooses an optimal ordering where some of the tests can be executed in parallel, before sending the new schedule of tests to the runners. A monitoring system keeps track of resource usage and trace logs, and sends the results back to the profiles so that they can be used to improve their quality. It also sends the results back to the repository.*

There were some goals mentioned in the Approach section of this paper, four to be exact. The first goal was to find software tests to use as examples, the second to make a script to run tests and record resource use, the third was to use recorded data to define profiles and fourth was to design an algorithm that uses the profiles in intelligent scheduling. The following table outlines the degree of completion for each goal along with some information of what happened to each of them.

Goal #	**Degree of Achievement**	**Reasoning / Details**
1	De-prioritized	It was found that it was more important to start developing the algorithm
2	Partially Achieved	A script that records resource usage was developed and used to make some examples, but the recorded data did not come from software tests and was mostly synthetic.
3	Achieved	Profiles were made and defined by CPU load in the examples, with support for memory usage as well
4	Achieved	The algorithm investigates pairwise compatibility of tests and states how long they have to wait before starting in order to avoid high resource use.

Table 3: *The four goals that were outlined in the Approach chapter, along with their degree of achievement and reasoning behind it.*

5 Discussion

The exploratory nature of this project meant that the result would be more knowledge-based rather than a product. There were many ideas in play at the beginning of the project, with the expectation that there would be a higher degree of technical implementation. This ended up not being the case, as the project quickly became an exploration of the idea surrounding the planned implementations rather than an advanced implementation. A long thesis may have had a higher degree of technical work, or just further exploration of even more ideas. The latter seems more likely, as there was a constant influx of ideas that ended up not being implemented during this short thesis. A technical implementation of the ideas presented in this thesis may be a good basis for a future thesis, most likely in a long thesis.

This chapter will discuss the project itself, the ideas and results that were found and the experiences had when working on the project. Starting with the algorithm itself, then the profiles and what happened to them, before moving on to how the project fits in the broader context of green cloud computing and the feasibility of a future implementation in a test framework.

5.1 The Problem Statement

The problem statement explains the motivation behind this thesis, namely improving test scheduling in the hopes of facilitating a greener cloud. The problem statement was: *Explore the usefulness of workload profiling in order to facilitate green resource provisioning for software testing in the cloud.*. The background chapter showed that there was some difficulty in finding literature on this exact subject. There was however a substantial amount of literature on the broader subjects of green cloud computing and resource allocation. The lack of relevant research forced the project in a more exploratory direction than expected, and cemented the technical aspects as a proof of concept rather than parts of a finished product.

Green resource provisioning refers to practices that seek to reduce the carbon impact of data centers by provisioning as few servers as possible, thereby reducing electricity usage. This thesis tries to facilitate greener provisioning by letting more tests run on the same hardware in parallel, reducing the number of active servers and potentially shortening the runtime of the tests. I therefore believe that the project was successful in this respect, due to the fact that the new ideas presented here can most likely be used in greener scheduling systems.

5.2 PAST and the Implementation of the Idea

The algorithm ended up being the main focus of the thesis, as the focus gradually changed from the profiles themselves over to the theory surrounding the algorithm and its applications. The work around the algorithm was still exploratory in nature, leading to the implementation being relatively uncomplicated and serving as a a proof of concept that can be expanded into a complete product. The implementation itself ended up being more bare-bones than what was originally intended, in the sense that it was not connected to an existing test framework such as Jenkins or Gitlab and it was not used in a scheduling program for cloud environments. With that being said, I still believe that the algorithm is a meaningful contribution to the field, as it offered new knowledge and a new approach to test scheduling in cloud environments. The usual way of looking at workload profiling is through static resource demands, which is not necessarily representative of the actual resource use of the test during its execution. A test has varying resource use depending on what it is doing at a given time and the algorithm in this paper tries to look at the pattern of resource use instead of representing it as a single number. This approach enables a more fine-tuned resource scheduling which could potentially allow data center administrators to run more software tests on the same server than what would otherwise be possible.

The actual data being fed to the algorithm in my tests does not come from real software tests, but that is not important to the point being made. Data can be from any application or test so long as it is in the correct format, the algorithm can thus be used for any application and not just software tests. The source of the data did not matter in this case because most of the examples are only meant to show the functionality of the algorithm, testing in a real cloud environment with a test environment like Gitlab or Jenkins could be done in a future thesis.

PAST is mostly directed at scheduling tests and not managing the resources used by the test environments. Provisioning resources and managing their usage is a suitable task for future work, especially when combined with the algorithm from this thesis. The benefits should become apparent when intelligent scheduling is used in conjunction with resource allocation, where the algorithm can hopefully enable a greater degree of consolidation by scheduling the tests in a way that lets them be executed in tandem without using excessive amounts of CPU and memory.

There are many ways to further develop the algorithm and the idea behind it could be applied to other applications than what is presented in this thesis. The first thing that comes to mind is the fact that PAST does not consider the entire pipeline, it only looks at tests pairwise and does not consider how the schedule of the entire pipeline would change after the algorithm has been applied. The first step to improving the algorithm could be to add functionality for looking at more than two tests at the same time, or at least looking at how the new schedule affects the other tests in the pipeline. An example of this would be if we have three tests, t1, t2 and t3, we find out that t1 and t2 can run at the same time. It is now necessary to check if t3 can run together with t1 and t2, or if the combined CPU load from all three tests would be too much.

5.3 What Happened To The Profiles?

The initial plan of the project was to have a greater focus on the profiles themselves and use them to redefine how we look at resource scheduling. Focus rapidly changed to the algorithm itself and the ideas surrounding its implementation in a test framework instead, with the profiles becoming less important. Each profile contains a list of resource usage over time for several executions of the same program as well as an average of all the executions. The profile can be defined by the maximum or average of the recorded values, this gives the profiles some flexibility. The conservative estimate of the maximum values may be suited to cases where there are frequent spikes in CPU load. The average values may be more suited for cases where the CPU load generally follows the same pattern across executions. The source of the data is not that important to the definition of the profiles, so long as it is in the correct format. This means that the algorithm could be used for other applications and not just software tests, opening up even more opportunities for further research.

There were many ideas surrounding what the profiles should be and how they could be used in the future, but I quickly discovered that the profiles themselves were not the most interesting part of the project and thus did not devote more time to the exploration of those ideas. The shift of focus over to the algorithm came when I experimented with simple profiles that were initially meant to be placeholders, but through some discussion with my supervisor, we realized that placeholder profiles could be used to sufficiently illustrate the concept being presented. The concept being the novel way of thinking of workload profiles with resource use that changes over time, meaning that as long as the profiles contained resource usage patterns instead of static numbers it did not really matter if the implementation was a bit more basic than what we initially planned.

The profiles are still important on some level, they were just not the focus of this thesis as the concept could be illustrated without further development of the profiles. More sophisticated models might be needed if the algorithm was to be implemented in a real test environment where the resource usage patterns need to be accurate if we want to make sure that the servers don't crash due to low memory or CPU capacity.

Access to relevant data was also a problem for the development of the profiles, some data was generated but this was mostly just meant to illustrate the concept of the profiles and not representative of real-world data. More data could have been used to develop statistical models that may predict the resource usage of a given test pipeline, leading to sophisticated profiles with far more accurate resource usage patterns than the ones used in this project. By statistical models, I am referring to methods such as regression and other supervised learning models. Given a set of training data consisting of resource usage patterns of software tests, one could make a model that predicts future resource usage patterns. Basing the profiles on these predicted patterns could yield a scheduling algorithm that produces fewer errors than the one presented in this thesis, depending on the quality of the statistical models.

5.4 Reflections on the Initial Plan and the Final Result

The exploratory nature of the project and focus on ideas rather than implementation can be a frustrating experience depending on the expectations of the researcher. The expectations regarding this project were that there would be more focus on the technical aspects and thereby a result consisting of something that looks more like a finished product or implementation. This changed during the development of the project as it became apparent that an iterative approach was more suitable. An iterative approach meant that there were many new ideas that had to be explored technically, before going back and revising the idea. The lack of existing research and solutions on this topic meant that ideas had to be explored to check if they were dead ends or not, which is a time-consuming process. This being a short project meant that there was not a lot of time left for the technical implementation of those ideas, as most of the time was spent on exploration.

An example of one such idea that ended up being changed over the course of the project was the idea of testing the algorithm with real software tests in a real test environment. There are many open-source projects on the Internet that could be used for testing purposes by downloading the repository and running their tests. The problem was that many of the tests found were intrinsically tied to a testing environment like Github Actions, Jenkins or something similar. This meant that in order to schedule the tests according to the result of the PAST algorithm we would have to add a plugin or change the code of the test environment being used. The next iteration of the idea was to use such open-source projects to generate data by monitoring the CPU load of the runner executing the tests and basing profiles on the resulting data. But this also proved to be outside of the scope of this project as it would require setting up dedicated runners and a system to monitor them. Setting up a local test environment and having the runners send a signal to the monitoring script to indicate when a test was finished would entail a substantial amount of development, this being a short project meant that there was not enough time to do this.

This kind of iterative project might have been better suited for a long thesis, as that would have left more time to do the technical implementation. An iterative project needs time to arrive at the most feasible answer to the problem statement, and then even more time to implement the solution. A short, iterative exploratory thesis is destined to touch on a lot of ideas without going into depth on many of them. This can be a frustrating experience for the researcher, as they can get a feeling of not having accomplished what they set out to do. The feeling can stem from a misunderstanding of what such a project entails, especially when the chosen subject does not have a lot of existing research to look into beforehand.

5.5 Related Research

The most similar topic within the field of green computing and VM consolidation is probably bin-packing, although the approach in this thesis differs in a lot of ways. There are many papers regarding green cloud computing and the various methods involved in making data centers more energy efficient through both hardware and software, but very little in the way of bin-packing algorithms that look at items that can change size over time.

Energy can be reused in creative ways, and many papers try to find novel ways of utilizing the heat generated by computer equipment in data centers. Some do this by for example utilizing the water that has been used to cool down the computers, this water becomes very hot and could be transported to nearby homes and used in heating equipment. This train of thought is being explored by Amazon through one of their data centers located in Dublin [1]. They are planning to use the heat generated by the data center to heat up nearby community buildings, thereby reducing the carbon impact of the data center. Others suggest that the warm water could be sent to a nearby hydropower plant and used to create more electricity. Exploring such strategies could be a master thesis in itself and the field has many interesting and novel ideas that could be explored further.

Consolidation algorithms is also a highly researched field with room for more improvements and new ideas, such as the concept presented in this thesis. Many papers focus on faster and smarter algorithms for consolidation, but most of the ones found during the literature review don't look too closely at the workload profiles themselves. The reasoning behind this may be the same as the reason why this project also shifted focus away from the profiles, namely the fact that the profiles proved to be not as important as the algorithm in that the algorithm could be tested and developed with mock profiles. Mock profiles are in this case basic versions of the ones originally envisioned, developed with the purpose of showing the functionality of the algorithm without spending too much time on the profiles. This was done because of the fact that this is a short thesis and there was not enough time to develop a more complete technical implementation.

The relation to bin-packing is mostly restricted to the concept of trying to fit as many tests on the same server as possible. The similarity is apparent if we view a test as an item and the servers as bins, then the problem becomes fitting the items into as few bins as possible. This thesis does not deal with the algorithmic aspect of finding an optimal way of fitting items into bins, but it is still relevant to the problem. The PAST algorithm can be viewed as a facilitating agent to a bin-packing solution in that it lets us fit more tests on a server than what would otherwise be possible. Thus it could be used to improve a solution to an existing bin-packing problem in cloud environments. This opens up yet another avenue for future work, where one could attempt to combine the concepts from this thesis with a bin-packing algorithm.

The algorithm in its current form only looks at one metric, which it assumes to be either CPU or memory, it can however be extended to look at several metrics at the same time. The implementation of handling more metrics was thus left out due to it not being important to this thesis, using CPU load as an example was sufficient to show this proof of concept. Moreover, it became increasingly apparent when working on this thesis that the end result would be less of a complete product and more of a demonstration of an idea, the idea being a kind of temporal bin-packing. Temporal bin-packing has been referred to as a generalized bin-packing problem where the items have a lifespan [7]. The temporal bin-packing in this thesis is a bit different, in that time is included by letting the resource demand of the items change over time, also the way the problem is approached was quite different from traditional bin-packing. It is similar to bin-packing in that the goal is to fit as many items (virtual machines) in as few bins (servers) as possible, but the algorithm is not an implementation of a solution to the bin-packing problem. The algorithm could be used in a bin-packing algorithm, in that it may enable the program to fit more VMs on a machine. In this way it may be included in a problem similar to the temporal bin-packing introduced by De Cauwer et al. in the paper *The Temporal Bin Packing Problem: An Application to Workload Management in Data Centers.* [7].

5.6 Towards Greener Software Testing

Saving resources is the main goal of this thesis, but which part of the project is meant to reduce the resource use changed during the development of the algorithm. The initial thought was that resources would be saved by reducing the time it takes to complete a pipeline of tests, but this is not necessarily the case. Tests that are not dependent on each other are often executed in parallel to save time, but they might be running on separate servers. The location of the test runners is usually not considered when running the tests, as it is left up to the scheduling algorithm of the cloud provider to find a spot to run the tests.

The algorithm in this thesis wants to run tests in parallel but sometimes has to delay one of the tests in order to avoid simultaneous spikes in resource usage. This means that if one of the tests is delayed, then the time it takes to complete the pipeline might be longer than if the tests were executed in parallel without the use of the algorithm during the scheduling process. Does this mean that the algorithm does not help in reducing resource usage? No, not necessarily. Although it might sound counter-intuitive, there could still be a reduction in resource use even though it takes longer to complete the pipeline. The reduction comes from reducing the number of servers being used by consolidating the virtual machines or how long the servers are kept running if the tests being run are compatible. Data centers already have consolidation and scheduling algorithms in place, but the lack of literature surrounding workload profiles that look at the pattern of resource usage through the lifetime of the application instead of looking at it as a static number suggests that this is a novel idea that could potentially improve VM consolidation.

The assumption here is that resources are saved if the algorithm improves consolidation by letting more tests be placed on the same server even if it does not shorten the runtime in some cases. For this reason, I think that this project fits under the umbrella of green cloud computing, and may be included in future work on the subject. A combination of more intelligent VM consolidation and scheduling with the existing resource reduction techniques in data centers may help curve the ever-increasing energy demand of cloud computing. Such measures need to be taken due to the extreme increase in the popularity of cloud services to prevent the energy demand from growing completely out of control. Researchers and cloud providers have been working hard in reducing the increase in energy usage and have made significant strides toward a greener cloud.

Moving more and more computation to the cloud instead of having it distributed in personal computers opens the door for far more efficient power-saving strategies than what would otherwise be possible. It would be very difficult to measure the environmental impact of computing if we did not have centralized computing hubs such as cloud data centers, and we should exploit this as much as we can to reduce the environmental impact of computers. Given how widespread and important computers and the Internet are in modern society, it is critical that measures are taken to reduce their environmental impact and ensure stable access to electricity for data centers.

5.7 How Feasible Is A Future Implementation?

The lack of relevant research found during the literary review phase of this project leads me to believe that this is a relatively unexplored avenue of workload profiling and scheduling. The feasibility of the algorithm needs to be decided through testing in an actual test environment such as Gitlab or Jenkins, which could be a good basis for another thesis in the future.

Implementation into an open-source testing service such as Jenkins seems to be the most obvious path towards a feasible implementation. Jenkins in particular has support for user-made plugins that can easily be downloaded and installed into a Jenkins server, with extensive documentation available on their website [9]. The scheduling algorithm could be used as such a plugin, connecting this to an allocation system in a cloud environment could turn the concept of this thesis into a real product. This requires a lot of work and has some prerequisites, but seems doable. It would require access to the internal systems of a cloud environment to be completely certain that the algorithm contributed to a higher degree of consolidation.

The feasibility of the implementation also depends on what the goal is. There are many ways to save time in software testing, but if the goal is to save time while also consolidating as much as possible then PAST might be a feasible solution. Virtual machines that are running on the same server can share resources and thereby reduce the total energy demand of their tasks, meaning that there are two benefits to consolidating VMs to the same server. They can save resources by keeping more servers idle and by reducing the power needed by the VMs by letting them share the resources of the server.

5.8 Testing The Profiles

The profiles are meant to facilitate more efficient usage of resources when performing software tests in cloud environments through VM consolidation, but testing the process in a cloud environment is not a simple task. Provisioning and resource allocation is usually done by the cloud provider themselves, which means that one would need access to their back-end services in order to make changes to that process. This falls outside of the scope of this project, as this is a short thesis, but there are ways to mimic VM consolidation for testing and demonstration purposes. There are two main methods that come to mind when simulating a cloud on a physical machine, namely multi-threading and containerization. A thread or a container can be viewed as a VM, and running several at the same time can be used as a cloud for demonstration purposes.

Containers are more similar to VMs than threads, so they are the most logical choice when simulating a cloud. Matching profiles found through the algorithm can be placed in Docker containers and executed simultaneously on the same machine, while monitoring the total resource usage of the machine. It is then possible to see if the total resource usage exceeds the chosen threshold at any time throughout the run-time of both tests. The experiment is considered a success if the resource usage does not exceed the threshold at any point.

Some of the simulations in this paper was done using containers, but not for the purpose of simulating a cloud. They were used to generate CPU load that might look similar to resource usage metrics from actual software tests.

5.9 Future Work

The previous paragraphs of this section have already mentioned some avenues for future work, but there were several more ideas that were discovered when working on this project. Implementation into a CI/CD platform such as Gitlab or Jenkins is the most logical continuation of the project. Combining this with a cloud environment as well would make it possible to accurately assess the usefulness of PAST by testing it with actual VMs and seeing if it is possible to use fewer VMs to do the tests.

The current version of the algorithm assumes that each execution of a given test takes an equal amount of time, but this is practically never the case in real life. This was done mostly because of the way the test data was generated, as using real tests would have given data of varying lengths. Alterations to the algorithm would need to be made to support data from real tests.

Measuring the time it takes to complete a pipeline before and after implementing PAST opens the way for more future work. I demonstrated the essence of this idea in the results section, where the percentage of improvement after applying PAST is shown (see table 2). It is necessary to run a test multiple times to get an idea of how long a typical run of the test takes, as basing the profile on a single run could be very misleading. There are several ways to represent the typical time, such as the average or the median. Estimating using the average is a bad idea if the data has significant outliers, as they could skew the data one way or the other. The median might be more suited in some cases but it also risks losing a lot of information. In the case of how long it takes to run a test, the average is a suitable estimate. This is due to the fact that most tests should have a somewhat consistent runtime without too many outliers.

Investigating the results of the algorithm is also an area that can be explored in future work, especially when comparing the speed of real pipelines before and after applying PAST. A possible way of measuring if PAST is faster or slower than traditional scheduling could be to take the shortest registered runtime of a test, lets call it t^*, and comparing it to the time used after implementing PAST. If the new time is less than t^* we can conclude that PAST has quickened the pipeline. PAST could still be beneficial if it does not beat the optimal time t^*, as it is possible that it was able to save resources through consolidation even if the new schedule was not faster than the old one.

Other metrics such as I/O statistics and network usage would be excellent additions to the algorithm, as they would open up for more sophisticated profiles and schedules. Adding them should be fairly straightforward and could be done in much the same way as CPU and memory. The challenge comes in gathering the data and processing it, the comparison between metrics from different profiles should be mostly the same as with CPU. Finding the optimal upper limit for CPU load and other metrics is also important, partly due to the fact that it may vary depending on the type of test being monitored as well as the conditions of the server.

Investigating the effect of consolidating two tests has on the rest of the pipeline is also a big improvement that can be made. Meaning that if two tests are scheduled to run at the same time, then the next test in the schedule can be compared to the combined resource usage of the two preceding tests. There should be limits to this however, as constantly stacking tests together and comparing the sum of CPU load to the next test would eventually result in none of the following tests being compatible with the consolidated ones. One solution to this could be to stop trying to stack tests when a crash is found, the algorithm could then try comparing the next two tests to each other.

Expanding and generalizing the algorithm to support other applications is something that has been mentioned earlier in the paper as well, and should be possible for certain types of software. The idea behind the algorithm is developed around scheduled short-lived tasks that are computationally demanding and would not be as applicable to long-lived software such as web servers. One type of software that comes to mind is high-performance computing and other scientific or mathematical software, as they are often scheduled and consist of various tasks that could be profiled in a similar way to the profiles from this project.

6 Conclusion

The goal of this project was to explore the usefulness of workload profiling in greener software test environments. This entailed a critical investigation where several ideas were explored, some ideas proved to have potential while others were not as important to the results.

The data-collection phase of the project ended up being fairly lightweight, as there was an eagerness to start testing and developing the algorithm itself. This development resulted in a Profile-based Algorithm for Scheduling Tests, referred to as PAST. PAST combines fine-tuned temporal workload profiles with an algorithm that compresses a test schedule so that the tests can be executed concurrently on the same server without exceeding limits on resource usage. This novel way of looking at workload profiles and scheduling differs from the traditional approach in several ways. The profiles are unique in that they look at how resource usage changes over the runtime of the software instead of representing the demand as a static number. Scheduling was also improved by opening the way for a more consolidated approach by letting the runtime of tests overlap while running on the same server.

PAST is most suited as an addition to services such as Jenkins or Gitlab and could improve them by opening the possibility for fast and resource-efficient pipelines. There is a possibility of generalizing the concept to other applications as well, with some constraints. The principle of this project revolves around applications that run in a schedule and would not be applicable to long-lived applications such as web servers. Software that does calculations might be another suitable field for this kind of scheduling algorithm, where there is a queue of computations that could be optimized to complete some of the tasks in parallel.

One of the most important goals of the project was to find a way to facilitate greener test environments, this was achieved in the form of new knowledge that can help improve a previously unexplored avenue of workload consolidation. The new approach is similar to bin-packing methods, but differs in that it lets the size of the items change over time and thereby enabling a more fine-tuned view. Additionally, it does not try to find an optimal method of placing items into bins but rather tries to show that items that were previously thought of as too big to fit in the same bin could in fact be placed together through a more dynamic approach.

There is room for more research in the data collection and workload profiles, as the profiles were based on synthetic data and their implementation was fairly simple. But the contribution the algorithm offers in the form of a new way of combining workload profiles with intelligent scheduling still stands.

The end result of this project is a scheduling algorithm that seems to be implementable in a CI/CD system, where it may help save resources and, when combined with a cloud environment, stall the ever-increasing need for electricity in data centers.

Bibliography

[1] Amazon. *Local community buildings in Ireland to be heated by Amazon data centre*. Blog. 2020. URL: https://www.aboutamazon.eu/news/amazon-web-services/local-community-buildings-in-ireland-to-be-heated-by-amazon-data-centre.

[2] Amazon. *Water Stewardship in Data Centers*. Blog. 2023. URL: https://sustainability.aboutamazon.co.uk/environment/the-cloud/water-stewardship#recycling-water-on-site-water-treatment.

[3] Qingwen Chen et al. "Profiling Energy Consumption of VMs for Green Cloud Computing". In: *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*. 2011, pp. 768–775. DOI: 10.1109/DASC.2011.131.

[4] Henrik I Christensen et al. "Approximation and online algorithms for multidimensional bin packing: A survey". In: *Computer Science Review* 24 (2017), pp. 63–79.

[5] European Commission et al. *Energy-efficient cloud computing technologies and policies for an eco-friendly cloud market : final study report*. Publications Office, 2020. DOI: doi/10.2759/3320.

[6] Peter Corcoran and Anders Andrae. "Emerging trends in electricity consumption for consumer ICT". In: *National University of Ireland, Galway, Connacht, Ireland, Tech. Rep* (2013).

[7] Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. "The Temporal Bin Packing Problem: An Application to Workload Management in Data Centres". In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. 2016, pp. 157–164. DOI: 10.1109/ICTAI.2016.0033.

[8] Anh Vu Do et al. "Profiling Applications for Virtual Machine Placement in Clouds". In: *2011 IEEE 4th International Conference on Cloud Computing*. 2011, pp. 660–667. DOI: 10.1109/CLOUD.2011.75.

[9] Jenkins. *Jenkins Developer Documentation*. Web Page. 2023. URL: https://www.jenkins.io/doc/developer/tutorial/prepare/.

[10] Toni Mastelic and Ivona Brandic. "Recent Trends in Energy-Efficient Cloud Computing". In: *IEEE Cloud Computing* 2.1 (2015), pp. 40–47. DOI: 10.1109/MCC.2015.15.

[11] Asit K. Mishra et al. "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters". In: *SIGMETRICS Perform. Eval. Rev.* 37.4 (Mar. 2010), pp. 34–41. ISSN: 0163-5999. DOI: 10.1145/1773394.1773400. URL: https://doi.org/10.1145/1773394.1773400.

[12] Ismael Solis Moreno et al. "An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Models". In: *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*. 2013, pp. 49–60. DOI: 10.1109/SOSE.2013.24.

[13] Alessandro Orso and Gregg Rothermel. "Software Testing: A Research Travelogue (2000–2014)". In: *Future of Software Engineering Proceedings*. FOSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 117–132. ISBN: 9781450328654. DOI: 10.1145/2593882.2593885. URL: https://doi.org/10.1145/2593882.2593885.

[14] Redhat. *What is CI/CD?* Blog. 2022. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[15] Statista. *Amazon, Microsoft and Google Dominate Cloud Market*. Web Page. 2021. URL: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.

[16] Statista. *Number of internet and social media users worldwide as of July 2022*. Web Page. 2022. URL: https://www.statista.com/statistics/617136/digital-population-worldwide/.

[17] Ingvild Stølen. "The CAST-algorithm bridging green energy with continuous testing." Thesis. 2021. URL: https://oda.oslomet.no/oda-xmlui/handle/11250/2774596.

[18] Kejiang Ye et al. “Profiling-Based Workload Consolidation and Migration in Virtualized Data Centers”. In: *IEEE Transactions on Parallel and Distributed Systems* 26.3 (2015), pp. 878–890. DOI: 10.1109/TPDS.2014.2313335.

7 Appendix

```python
import profiles, numpy as np, matplotlib.pyplot as plt

def main(profiles):
    for i in range(len(profiles) - 1):
        res, index = PAST(profiles[i], profiles[i+1], 90)
        if res == False:
            print("Profiles are incompatible")
        else:
            print("Match found between profiles at index {}".format
    (index))
            profiles[i+1].start = index

def PAST(p1, p2, threshold):
    print("Comparing profiles {} and {}".format(p1.name, p2.name))
    i = 0
    j = 0
    start_index = 0
    compatible = False
    a = p1.get_avg_cpu()
    b = p2.get_avg_cpu()
    while (i < len(a) and j < len(b)):
        compatible = True
        sum = a[i] + b[j]
        if sum > threshold:
            compatible = False
            i += 1
            j = 0
            start_index = i
            continue
        else:
            i += 1
            j += 1
            continue
    return compatible, start_index

def test():
    a = [50, 40, 20, 30, 20, 10, 80]
    b = [50, 60, 10, 30, 5, 5, 5]
    res, index = PAST(a, b, 90)
    if res == False:
        print("Profiles are incompatible")
    else:
        print("Match found between profiles at index {}".format(
    index))

def plot_profiles(p1, p2):
    p1_cpu = p1.get_avg_cpu()
    p2_cpu = p2.get_avg_cpu()
    t = range(len(p1_cpu))

    plt.rcParams.update({'font.size': 10})
    plt.subplot(1, 2, 1)
    plt.yticks([i*10 for i in range(11)])
    plt.ylim(bottom=0, top=100)
    plt.xlabel('Time in seconds')
    plt.plot(t, p1_cpu, color='blue')
    plt.title('Profile "{}"'.format(p1.name))

    plt.subplot(1, 2, 2)
    plt.yticks([i*10 for i in range(11)])
    plt.ylim(bottom=0, top=100)
    plt.xlabel('Time in seconds')
    plt.plot(t, p2_cpu, color='orange')
    plt.title('Profile "{}"'.format(p2.name))

    plt.suptitle('CPU usage of two profiles')
    plt.savefig('figs/{}_{}_together.png'.format(p1.name, p2.name))
    plt.clf()

def plot_delayed_start(p1, p2, threshold, aggregate="average"):
    if aggregate == "average":
        p1_cpu = p1.get_avg_cpu()
        p2_cpu = p2.get_avg_cpu()
    elif aggregate == "max":
        p1_cpu = p1.get_max_cpu()
        p2_cpu = p2.get_max_cpu()

    t = range(len(p1_cpu + p2_cpu))
    p2_cpu_first = [None for i in range(p2.start)] + p2_cpu
    p2_cpu_ = p2_cpu_first + [None for i in range(len(t) - len(
    p2_cpu_first))]
    p1_cpu_ = p1_cpu + [None for i in range(len(p2_cpu))]
    sum_p1_p2 = p1_cpu[0:p2.start] + [p1_cpu_[i] + p2_cpu_[i] for i
     in range(p2.start, len(p1_cpu))] + p2_cpu_[len(p1_cpu):]


    plt.yticks([i*10 for i in range(11)])
    plt.ylim(bottom=0, top=100)
    plt.xlabel('Time in seconds')
    plt.ylabel('CPU usage percent at time t')
    plt.plot(t, p1_cpu_, color='blue')
    plt.plot(t, p2_cpu_, color='orange')
    plt.legend([p1.name, p2.name])
    plt.title('Two profiles with PAST schedule')
    plt.savefig('figs/{}_{}_delayed.png'.format(p1.name, p2.name))
    plt.clf()



if __name__ == "__main__":
    high_low = profiles.Profile('test_high_low', 'high_low',
    include_memory=False)
    low_high = profiles.Profile('test_low_high', 'low_high',
    include_memory=False)
    t1 = profiles.Profile('t1', 't1', False)
    t2 = profiles.Profile('t2', 't2', False)
    main([high_low, low_high, t1, t2])
    high_low.plot()
    low_high.plot()
    plot_profiles(t1, t2)
    plot_profiles(high_low, low_high)
    plot_delayed_start(high_low, low_high, 90)
    plot_delayed_start(t1, t2, 90)
```

```
import profiles, random, os
import numpy as np

def generate(n, t1, t2, t3, t4):
    cpu = np.zeros(n)
    for i in range(0, int(n / 2)):
        cpu[i] = (random.randint(t1, t2))
    for i in range(int(n / 2), n):
        cpu[i] = random.randint(t3, t4)
    return list(cpu)

def write_to_file(cpu, name, n):
    filename = 'logs/{}/{}.txt'.format(name, n)
    os.makedirs(os.path.dirname(filename), exist_ok=True)
    with open(filename, 'w') as outfile:
        for stat in cpu:
            outfile.write(str(stat) + '\n')


def main(name1, name2, t1, t2, t3, t4):
    for i in range(6):
        for j in range(5):
            write_to_file(generate(100, t1, t2, t3, t4), name1, j)
            write_to_file(generate(100, t3, t4, t1, t2), name2, j)


if __name__ == "__main__":
    dir1 = "logs/test_high_low"
    dir2 = "logs/test_low_high"
    main('t1', 't2', 0, 60, 20, 40)
```

```
def helper(a, b):
    threshold = 90
    a_avg_cpu = a.get_avg_cpu()
    b_avg_cpu = b.get_avg_cpu()
    G = (np.zeros(len(a_avg_cpu) + len(b_avg_cpu))).tolist()
    res = rec_alt(a_avg_cpu, b_avg_cpu, 0, threshold)
```

```python
    if res == -1:
        print("No match found")
    else:
        b.start = res
        a.matches.append((b,res))
        print("Match found between profiles {} and {} at index {}".
    format(a.name, b.name, res))

def rec_alt(a, b, index, threshold, match_found=False):
    if index >= len(a):
        return -1
    for i in range(len(a)):
        if (i + index >= len(a)):
            break
        if (a[i+index] + b[i]) > threshold:
            print(i+index, i)
            rec_alt(a, b, index+1, threshold)
            return -1
    print("match found, returning")
    return index
```

```python
import profiles as p, make_profiles as mp, PAST, matplotlib.pyplot
    as plt


def main():
    high_low = p.Profile('test_high_low', 'high_low', False)
    low_high = p.Profile('test_low_high', 'low_high', False)
    t1 = p.Profile('t1', 't1', False)
    t2 = p.Profile('t2', 't2', False)
    plot_sum(t1, t2)
    plt.clf()
    plot_sum(high_low, low_high)


def plot_profiles(p1, p2):
    p1_cpu = p1.get_max_cpu()
    p2_cpu = p2.get_max_cpu()
    t = range(len(p1_cpu))
    plt.yticks([i*10 for i in range(11)])
    plt.ylim(bottom=0, top=100)
    plt.xlabel('Time in seconds')
    plt.ylabel('CPU usage percent at time t')
    plt.plot(t, p1_cpu, color='blue')
    plt.plot(t, p2_cpu, color='orange')
    plt.legend([p1.name, p2.name])
    plt.title('CPU usage of two compatible profiles')
    plt.savefig('figs/high_low_low_high.png')
    plt.clf()

def plot_sum(p1, p2):
    p1_cpu = p1.get_avg_cpu()
    p2_cpu = p2.get_avg_cpu()
    p1_p2_cpu = [x + y for x, y in zip(p1_cpu, p2_cpu)]
    t = range(len(p1_p2_cpu))
    plt.yticks([i*10 for i in range(11)])
    plt.ylim(bottom=0, top=100)
    plt.xlabel('Time in seconds')
    plt.ylabel('CPU usage percent at time t')
    plt.plot(t, p1_p2_cpu, color='green')
    plt.title('Sum of "{}" and "{}"'.format(p1.name, p2.name))
    plt.savefig('figs/sum_of_{}_{}.png'.format(p1.name, p2.name))



if __name__ == "__main__":
    main()
```

```
import os
import matplotlib.pyplot as plt


# The following class is an example profile that takes in a
    directory of logs
# It processes all of the txt files in the chosen directory and
    assumes that each file is the log from one execution of a test
#
class Profile:
```

```python
    def __init__(self, dir, name, include_memory=True):
        self.name = name
        self.include_memory = include_memory
        self.cpu = []
        self.memory = []
        self.matches = []
        self.start = 0 # This indicates whether or not this instance of the profile should wait, and how long

        self.process_logs(dir)
        self.l = len(self.cpu)

    def process_logs(self, dir):
        if self.include_memory:
            for filename in os.listdir('./logs/' + dir):
                current_cpu = []
                current_memory = []
                with open('./logs/{}/{}'.format(dir, filename), 'r') as infile:
                    for line in infile.readlines():
                        current_cpu.append(float(line.split(' ')[0]))
                        current_memory.append(float(line.split(' ')[1]))
                    self.cpu.append(current_cpu)
                    self.memory.append(current_memory)
        else:
            for filename in os.listdir('./logs/' + dir):
                current_cpu = []
                with open('./logs/{}/{}'.format(dir, filename), 'r') as infile:
                    for line in infile.readlines():
                        current_cpu.append(float(line.split(' ')[0]))
                    self.cpu.append(current_cpu)



    def get_avg_cpu(self):
        return self.aggregate_avg(self.cpu)

    def get_avg_memory(self):
        return self.aggregate_avg(self.memory)

    def get_max_cpu(self):
        return self.aggregate_max(self.cpu)

    def aggregate_avg(self, metric): #Returns a list of averages,
    average resource use for each timestamp of given metric
        avg = metric[0]
        for i in range(1, len(metric)):
            for j in range(len(metric[i])):
                avg[j] += metric[i][j]
        for i in range(len(avg)):
            avg[i] = round(avg[i]/len(metric), 1)
        return avg

    def aggregate_max(self, metric): # Returns a list where each
    item is the highest value found for that timestamp
        m = metric[0]
        for i in range (1, len(metric)):
            for j in range(len(metric[i])):
                current = metric[i][j]
                m[j] = current if current > m[j] else m[j]
        return m

    def get_max(self, metric):# Returns the highest recorded value
    for the given metric
        return max(metric)

    def plot(self, aggregate="average"):
        if aggregate == "average":
            cpu = self.aggregate_avg(self.cpu)
            if self.include_memory:
                memory = self.aggregate_avg(self.memory)
        else:
```

```python
        cpu = self.aggregate_max(self.cpu)
        if self.include_memory:
            memory = self.aggregate_max(self.memory)


    t = range(0, len(cpu))
    plt.rcParams.update({'font.size': 15})
    plt.plot(t, cpu)
    plt.title('{} CPU usage of profile "{}"'.format(aggregate,
self.name))
    plt.yticks([i*10 for i in range(11)])
    plt.ylim(bottom=0, top=100)
    plt.xlabel('Time in seconds')
    plt.ylabel('CPU usage percent at time t')
    plt.savefig('./figs/{}_{}_cpu.png'.format( aggregate,self.
name))
    plt.clf()

    if self.include_memory:
        plt.plot(t, memory)
        plt.title('{} Memory usage of profile "{}"'.format(
aggregate, self.name))
        plt.yticks([i*10 for i in range(11)])
        plt.ylim(bottom=0, top=100)
        plt.xlabel('Time in seconds')
        plt.ylabel('Memory usage percentage at time t')
        plt.savefig('./figs/{}_{}_memory.png'.format(aggregate,
 self.name))
        plt.clf()


def __str__(self):
    return "Name: {}\nDimensions: l = {}".format(self.name,
self.l)




```

```python
if __name__ == "__main__":
    p1 = Profile('test1', 'test1')
    p1.plot("average")
    p1.plot("max")

    p2 = Profile('test2', 'test2')
    p2.plot("average")
    p2.plot("max")
```

```python
import matplotlib.pyplot as plt
import numpy as np

def draw(test1, test2, threshold, name):
    s = np.add(test1, test2)
    x = np.linspace(0, 11, 10)

    plt.plot(x, test1, drawstyle='steps', color='g')
    plt.plot(x, s, drawstyle='steps', color='b')

    plt.fill_between(x, s, step='pre', color='b', alpha=0.4)
    plt.fill_between(x, test1, step='pre', color='g')

    plt.axhline(threshold, color='r')
    plt.xlabel('Time')
    plt.ylabel('Resource usage')
    plt.title(name)
    plt.ylim(top=10, bottom=0)
    plt.xticks([i for i in range(11)])
    plt.legend(['test 1', 'Sum of test 1 and test 2'], loc='upper
    right')
    plt.savefig('./figs/examples/{}'.format(name))
    plt.clf()

    plot_test(test1, 'Test_1_{}'.format(name), threshold, 'g')
    plot_test(test2, 'Test_2_{}'.format(name), threshold, 'b')




```

```python
def plot_test(test, name, threshold, c):
    x = np.linspace(0, 11, 10)
    plt.plot(x, test, drawstyle='steps', color=c)
    plt.fill_between(x, test, step='pre', color=c)
    plt.axhline(threshold, color='r')
    plt.xlabel('Time')
    plt.ylabel('Resource usage')
    plt.title(name)
    plt.ylim(top=10, bottom=0)
    plt.yticks([0,1,2,3,4,5,6,7,8,9,10])
    plt.xticks([0,1,2,3,4,5,6,7,8,9,10])
    plt.title('Resource usage of {} over time'.format(name))
    plt.savefig('./figs/examples/{}'.format(name))
    plt.clf()


def draw_special_case_1():
    x = np.linspace(0, 16, 15)
    T1 = [1,1,1,1,1,1,1,1,1,1,0,0,0,0,0]
    T2_T3 = [0,0,0,0,0,1,1,1,1,1,2,1,1,1,1]
    s = np.add(T1, T2_T3)
    plt.plot(x, T1, drawstyle='steps', color='b')
    plt.plot(x, s, drawstyle='steps', color='g')
    plt.fill_between(x, s, step='pre', color='g')
    plt.fill_between(x, T1, step='pre', color='b')
    plt.xlabel('Time')
    plt.ylabel('Resource usage')
    plt.title('Special case alternative 1')
    plt.ylim(top=10, bottom=0)
    plt.yticks([i for i in range(11)])
    plt.xticks([i for i in range(16)])
    plt.legend(['T1', 'T2 + T3'])
    plt.savefig('./figs/examples/T1_T2+T3.png')
    plt.clf()

def draw_special_case_2():
    x = np.linspace(0, 16, 15)
    T1 = [2,2,1,1,1,1,1,1,1,2,0,0,0,0,0]
    T2 = [0,0,0,2,2,2,2,2,0,0,0,0,0,0,0]
    T3 = [0,0,0,0,0,0,0,0,0,0,0,1,1,1,1]
    plt.plot(x, T1, drawstyle='steps', color='b')
    plt.plot(x, T2, drawstyle='steps', color='g')
    plt.plot(x, T3, drawstyle='steps', color='r')
    plt.fill_between(x, T2, step='pre', color='g')
    plt.fill_between(x, T1, step='pre', color='b')
    plt.fill_between(x, T3, step='pre', color='r')
    plt.xlabel('Time')
    plt.ylabel('Resource usage')
    plt.title('Special case alternative 2')
    plt.ylim(top=10, bottom=0)
    plt.yticks([i for i in range(11)])
    plt.xticks([i for i in range(16)])
    plt.legend(['T1', 'T2', 'T3'])
    plt.savefig('./figs/examples/T1_T2_T3.png')
    plt.clf()



if __name__ == "__main__":
    draw([7, 8, 5, 2, 3, 6, 2, 2, 1, 3], [6, 5, 3, 7, 6, 4, 5, 7,
    7, 6], 9, 'cpu_example_1.png')
    draw([2,4,1,5,3,6,7,8,6,7],[0,5,7,1,1,2,2,1,2,1], 9, '
    cpu_example_2')
    draw([7, 8, 5, 2, 3, 6, 2, 2, 1, 3],[0, 6, 5, 3, 7, 6, 4, 5, 7,
     7], 9, 'cpu_example_1_shifted_1')
    draw([7, 8, 5, 2, 3, 6, 2, 2, 1, 3],[0, 0, 6, 5, 3, 7, 6, 4, 5,
     7], 9, 'cpu_example_1_shifted_2')
    draw([7, 8, 5, 2, 3, 6, 2, 2, 1, 3],[0, 0, 0, 6, 5, 3, 7, 6, 4,
     5], 9, 'cpu_example_1_shifted_3')
    draw_special_case_1()
    draw_special_case_2()
```

```
import matplotlib.pyplot as plt
import os, datetime
from scipy.interpolate import make_interp_spline
import numpy as np

```

```python

# This script takes all logs (txt files) from a chosen directory
    and plots the contents in a graph.
# It assumes that the files contains CPU usage
def read_logs(filename):
    with open('./logs/{}'.format(filename), 'r') as infile:
        return list(map(float, infile.readlines()))


def read_files(dir):
    return [read_logs(filename) for filename in os.listdir(dir)]

def plot_stats(stats):
    for stat in stats:
        x = np.linspace(0, len(stat), num=len(stat))
        X_Y_spline = make_interp_spline(x, stat)

        X_ = np.linspace(x.min(), x.max(), 500)
        Y_ = X_Y_spline(X_)
        plt.plot(X_, Y_)

    plt.title('CPU usage percentages over time')
    plt.xlabel("Seconds")
    plt.ylabel("CPU usage %")
    plt.savefig('./figs/{}.png'.format(datetime.datetime.now()))

def main(directory):
    plot_stats(read_files('./'+directory))



if __name__ == "__main__":
    main('logs/experiment')
```

```python
import datetime, psutil, time, threading, os

def record(threshold, name):
    timestamp = str(datetime.datetime.now())
    filename = 'logs/{}/{}.txt'.format(name, timestamp.replace(' ',
    '-'))
    os.makedirs(os.path.dirname(filename), exist_ok=True)

    with open(filename, 'w') as outfile:
        print('Recording resource utilization...')
        start_time = time.time()
        while True:
            elapsed_time = time.time() - start_time
            if elapsed_time > threshold:
                print("Stopping recording")
                break
            outfile.write('{} {}\n'.format(psutil.cpu_percent(
    interval=1), psutil.virtual_memory().percent))

def simulate_docker(id, client, path_to_dockerfile):
    time.sleep(1)
    print("Start test {}...".format(id))
    tag = "test_{}".format(id)
    client.images.build(path=path_to_dockerfile, dockerfile='
    Dockerfile', rm=True, tag=tag)
    print("Finished test {}".format(id))

def simulate_cpu():
    os.system("sh simulate_cpu.sh")


def main(n_samples, name, t):
    for i in range(n_samples):
        t1 = threading.Thread(target=simulate_cpu)
        t1.start()
        record(t, name)
        t1.join()



if __name__ == "__main__":
    main(4, 'test1', 60)
    main(4, 'test2', 60)
```

```bash
#!bin/bash

for n in {1..10000};
do
    for m in {1..2000};
    do
       s=n+m
    done
done
```

```python
import docker, os, time, psutil, profiles, threading


def record_cpu(time_threshold, name):
    filename = 'logs/experiment/{}.txt'.format(name)
    os.makedirs(os.path.dirname(filename), exist_ok=True)

    with open(filename, 'w') as outfile:
        print('Recording resource utilization...')
        start_time = time.time()
        while True:
            elapsed_time = time.time() - start_time
            if elapsed_time > time_threshold:
                print("Stopping recording")
                break
            outfile.write('{} {}\n'.format(psutil.cpu_percent(
    interval=1), psutil.virtual_memory().percent))

def start_tests(a, b):
    client = docker.from_env()
    client.images.build(path='./', dockerfile='{}_Dockerfile'.
    format(a.name), tag=a.name, rm=True)
    time.sleep(b.start)
    client.images.build(path='./', dockerfile='{}_Dockerfile'.
    format(b.name), tag=b.name, rm=True)


def main(a, b):
    t = threading.Thread(target=start_tests, args=[a, b])
    t.start()
    record_cpu(60, 'experiment')
    t.join()




if __name__ == "__main__":
    profile_1 = profiles.Profile('test1', 'profile_1')
    profile_2 = profiles.Profile('test2', 'profile_2')
    main(profile_1, profile_2)
```