Master Thesis

Optimizing configurations to get the best of the cloud

Author
Pepijn Gramberg
ICA - 3818489

Supervisors
dr. J.A. Hoogeveen (UU)
dr. S.L.R Jansen (UU)
M. Overeem, MSc. (AFAS)

December 4, 2016

Abstract

Hosting a robust and high-performance Software as a Service solution requires resources and efficient usage of those resources. This relies on the infrastructure where the application is running on. If the usage of a SaaS solution gradually scales overtime the infrastructure can be managed on-the-fly. However if it is a new product and it is known that there will be a lot of users from the start, the infrastructure needs to be prepared for the load. The effectiveness of a configuration for the infrastructure can be measured in multiple objectives such as: costs, performance and robustness.

In this research we create insight into how the configuration of the infrastructure influences the objectives and present the decision maker with the best options to make a well informed trade-off, this is achieved with the use of a Pareto front.

We have created a configuration for hardware and orchestrator with 13 parameters. The focus of these parameters are the objectives. To create the Pareto front we use the non-dominating sorting genetic algorithm, this requires a fast evaluation of the configurations. We use heuristics to speed up the evaluation. Training these heuristics requires training data, to make sure the training data covers the possibilities evenly and with few samples we use a sampling strategy. The nearly orthogonal Latin hypercube sampling design is created to fulfil these properties, following this design we got 65 sample points.

The sample points are evaluated with a load test to obtain the measurements of the output variables. With this training data set the heuristics were trained, five additional samples were taken and used as a validation set. The heuristic that approximates the reality most accurately is used in the non-dominated sorting genetic algorithm. This algorithm creates the Pareto front, on the front analysis can be performed and a trade-off can be made.

Keywords: *Software as a Service, infrastructure, cloud, orchestrator, microservices, multi objective optimization, Pareto, heuristics, sampling, genetic algorithm*

Acknowledgements

I would like to thank my supervisors from Utrecht University Han Hoogeveen and Slinger Jansen, for giving guidance and feedback on my research. I want to thank Michiel Overeem for the day-to-day supervision at AFAS. This brings me to thank Rolf and Machiel who made it possible for me to do my research at AFAS. I would like to thank Mark for connecting me with AFAS.

I want to thank Marten, Sander, and Bart for the support on my research as well as distracting me with their own challenges and of course for the many games of bartlett. Finally I want to thank my family for supporting me.

Contents

1 Introduction **1**
1.1 Problem Statement . . . 3
1.2 Solution Structure . . . 4
1.3 Thesis Structure . . . 5

2 Research Context **6**
2.1 AFAS Software . . . 6
2.2 Next . . . 6
2.3 Service Fabric . . . 8
2.4 AMUSE . . . 10

3 Configurations **11**
3.1 Hardware Level . . . 11
3.2 Orchestrator Level . . . 12

4 Optimization **15**
4.1 Reducing to Single Objective . . . 15
4.2 Pareto Optimality . . . 16
4.3 Non-dominated Sorting Genetic Algorithm . . . 18

5 Sampling **24**
5.1 Sampling Strategy . . . 24
5.2 Orthogonal Design . . . 25

6 Heuristics **27**
6.1 Linear Regression . . . 27
6.2 Non-linear Regression . . . 28
6.3 Kriging . . . 28
6.4 Neural Network . . . 28
6.5 Support Vector Regression . . . 30
6.6 Accuracy . . . 30

7 Presentation **32**
7.1 Visual Presentation . . . 32
7.2 Reduce Dimensions . . . 35
7.3 Smart Presenting . . . 35

8 Next Implementation **38**
8.1 Configuration . . . 38
8.2 Sampling . . . 40
8.3 Sampling Execution . . . 41
8.4 Analysing Samples . . . 44
8.5 Heuristics . . . 47

9 Results 49
9.1 Configuration Analysis 49
9.2 Objectives Analysis 53

10 Discussion 59
10.1 Limitations 60
10.2 Future Work 61

11 Conclusion 62

References 63

A Sample Configurations 67

B Results 71

1 Introduction

As a user, you have certain expectations of how software behaves. For instance you expect that you are able to use the software whenever you want to. You want it to work fast, and not wait for data to load. If you store new data in the system it should not disappear unless you explicitly delete it.

As a provider of the software, you want the users to be happy with the product. If they are happy they keep using it and are willing to continue paying for it, that way the company can stay in business. To keep the business running, you also need to keep the costs low. For a company that is providing Software as a Service (SaaS), the costs of keeping the service running are an important portion of the costs. It is pointed out by Warfield [1] and Key [2] that it is important to reduce the cost of the service, as each month a user can be supported with fewer costs it is more profitable for the company.

The operational costs can be reduced by writing better performing software, unoptimized code can increase the running time of the software, use more memory or make it unable to scale to more users. Operational costs also depend on the infrastructure on which the software is deployed. You can have the best driver in the world, but if you put him in a golf cart he is not going to win the race. With infrastructure we mean hardware, server parts like processors or disks, but also the orchestrator, which is a layer between the hardware and the software that manages the applications that are deployed. It takes care of balancing the applications over multiple machines, it handles failures and helps with the process of upgrading the software. We need to balance the performance of the software and the price of the infrastructure. There are multiple ways to adapt the infrastructure but you always have to watch out for underprovisioning, lacking the capacity to run the software up to your standard. Handling a request utilizes resources from the infrastructure, if all of these resources are already being utilized by requests then any new requests that come in cannot be handled or will be slowly handled. On the other hand there is the chance of over-provisioning, meaning the resources that are available are not used enough. For a user this is good, as every request can be handled in the fastest way possible. For the company however it means that there is infrastructure being paid for, that is not used intensively. Under- and over-provisioning is described by Lim et al. [3] and Armbrust et al. [4].

There are three important concepts when providing a SaaS solution. These concepts have resources in common, the hardware provides resources, software uses resources based on where the orchestrator allocates it.

Software Software requires resources to run. The performance of the software depends on the available resources and how efficient the software

uses them.

Hardware The hardware provides resources to the system on which the software is hosted. The cost of hosting software comes from the hardware.

Orchestrator The orchestrator handles the allocation of software on the hardware. Trying to give the software the resources it needs.

A common practise for creating an infrastructure is adding servers on the fly, using the current performance metrics and some insight into projected growth. Such an advice is given by Intel in a white paper by Cahal and Mailman [5], which is using Compound Annual Growth Rate to size the servers for the coming four years. There are cases however, when a new product is launched of which the performance is not yet known. This is not a problem when the load gradually scales up in usage. But when it is a new version with a completely new architecture (so not an update of the software but a replacement), there will be a large number of clients instantly on this new platform. The architectural changes come with a new kind of performance, it will behave in a different way than the other architecture to the usage of the system. This makes the old data no longer relevant, as it does not give us information about how the new architecture will behave.

Good examples of misjudgements in new software are found in the online gaming industry, as on launch day there can be real rushes for big games. The problems become visible fast and to a lot of people, as it is bothering a lot of individual customers. There have been numerous launches where the servers were not up to the task at hand, resulting in slow connections, large queues, random disconnects or even no availability at all. Examples of this are the launch of the original World of Warcraft game, where players were placed into a queue to join a server. The time you had to wait to play could run up to multiple hours, an experience of this is described by Lopez [6]. The start of Simcity not only made it a long wait to play the game, but also had impact on other games by the same company. The whole game library of EA Games (Origin) was not responsive, influencing the experience of players that wanted to play another EA game than Simcity, this failure is described on Ars Technica [7]. As a final example, Pokémon Go, an application for on a mobile device that uses the environment to play, there was such a large interest that the first couple of days that the servers could not handle the load, giving players the message *to come back later* as explained on the news-site Polygon [8]. Obviously customers are put off by these kind of events, and are likely to think twice before purchasing another game or even start discouraging other potential buyers. To avoid getting bad reviews and a decline in business, it is paramount to have the infrastructure ready.

Creating a stable and high performance configuration while keeping the costs of the infrastructure low, is a Multi Objective Optimization problem.

These types of problems are explained in multiple forms by Ehrgott [9], each form has one thing in common which is the multi objective nature of the problem. Examples of objectives in a multi objective optimization could be durability of a product versus the production costs, the speed of a plane versus the fuel consumption of the flight or in the software case, the cost of the hardware and the duration a user needs to wait for a response. The objectives are working against each other, if we go for cheaper hardware the user needs to wait longer as there are less resources available in the system. There is no impartial way to combine them into a single objective, we can only give a subjective priority. Because of this it is impossible to give an objective single optimal configuration without requiring a precise weighing of objectives by the decision maker or turning all but one of the objectives in to a constraint. A solution to this problem is called a Pareto front, instead of giving one answer, the decision maker is presented with multiple answers all laying on the Pareto front. A solution lays on the Pareto front if and only if it can not improve in any of the objectives while not worsening in any of the other objectives. According to Goel et al. [10] as well as Loghmani and Ghoddosian [11] this concept is used to make it possible for the decision maker to come in as the last step in a multi objective optimization, pushing the introduction of subjective priority to the end.

The research was conducted at AFAS Software, a Dutch software company. They are developing a new ERP application to replace their current product. They would like insight into the behaviour of the application based on the infrastructure on which it is deployed. This is one of the aspects which is being researched at AFAS in connection with Utrecht University and Vrije Universiteit Amsterdam. These two universities together with AFAS are collaborating in the AMUSE research project, which looks at software composition, configuration, deployment and monitoring in combination with cloud and generated software.

1.1 Problem Statement

We have an unproven system in terms of performance and scaling. We want to be prepared with an infrastructure that can handle the load of the application. We do not have insight in how the configuration of the infrastructure will impact the performance of the application. There is no documented way of getting such an insight on which we can base decisions. This problem leads to the following research question.

How to guide a company to create a fitting configuration of hardware and orchestrator for hosting new cloud-based software?

From the research question, the sub questions below followed to make it

possible to help a company with such a decision.

1. What is configurable on hardware and orchestrator level?

Before we can start searching for candidate configurations we need to look at what is actually possible or useful. We need to explore what parts of the hardware and orchestrator can be adapted to our needs. But also what would impact our performance, you could configure your car in a different colour, however that would not make it faster.

2. How to traverse the search space to get candidate configurations?

We need a process to search through all possible candidate configurations. As we are guiding the company we need to present options that are interesting, these would be the configurations that are Pareto optimal. As these would make up a set of solutions that are in the mathematical sense the best. These could then be used in a trade-off.

3. How to mitigate the cost of an evaluation?

As we are searching through the configurations we need to evaluate how the configuration behaves. In the case of software this would mean creating the infrastructure and simulating users to obtain measurements. This can take a lot of time and money, which is something we would like to avoid. This is why we look at other ways to gain this information besides just doing the test.

4. How to present the decision maker with Pareto optimal solutions?

The Pareto front contains multiple options that require a manual trade off, we need to make the possibilities clear to the person making the decision. This is not necessarily a technical person, meaning we need to present the data in an understandable format.

1.2 Solution Structure

First we determined what should be part of the configuration, in the Next case we used 13 dimensions. Each dimension covers a parameter in either hardware or orchestrator. With the help of a sampling design we determined 65 different combinations, which cover the configuration space evenly. These 65 samples are tested in a testing environment, by building a cluster of virtual machines with the settings provided by the sample. We then ran four different test scenarios (Duration, type of input, failures) on each sample, this was done by running a load on the application. In two scenarios we test the performance of the configuration. In the other two scenarios we tested the robustness of the application by stress testing, this was done by inducing failures in the infrastructure.

We used the output from these test samples to create heuristics that will approximate the output variables. We tried several types of heuristics, they were validated by testing additional samples and comparing the output from reality with the output from the heuristic. The heuristic that approximated the reality the best was used in the algorithm and is static in the rest of the process. The heuristics made it possible to quickly evaluate the output of a new (untested) configuration without really testing it. This enabled the use of a genetic algorithm to generate an approximation of the Pareto front.

1.3 Thesis Structure

In Section 2 we provide background information and context to the problem. Sections 3 through 7 contain the minimal information about the case, to keep the approach and case separated. In Section 3 we look into the possibilities with configurations, what is of influence on the performance and costs. In Section 4 we look at how to traverse the search space of all possible configurations. After this we look at how to keep the evaluation of a configuration manageable and how to complete it in a reasonable time with the help of approximations in Sections 5 and 6. Then in Section 7 we look at ways to present the resulting data in a way that the decision maker can actually use it. In Section 8 we present the execution of the approach with the case study company AFAS. We present the results of this case in Section 9. We close off with the discussion and the conclusion.

2 Research Context

This section gives context to the research, what and who the key components and parties of this research are.

2.1 AFAS Software

AFAS[1] is a Dutch vendor of ERP software. The privately held company currently employs over 350 people and annually generates €100 million of revenue. AFAS currently delivers a fully integrated ERP suite which is used daily by more than 1.000.000 professional users of more than 10.000 customers.

2.2 Next

The ERP software that will succeed the current product is currently in development and is called Next. The software is generated based on a model of the business. In this model only business terms need to be used and no software specific definitions. The application is fully derived from this model and is tailored to the business. For this research the whole generation step is not really relevant, as we are interested in the deployment and operational side of it. What is relevant is the structure or architecture of the application. There are two key concepts used in the application, microservices and CQRS. The concept of microservices means the application is divided in smaller parts that work together as a whole. The division can be done in different ways, the important part is that each microservice is responsible for his own task. For example if you have a video streaming website, the searching can be done in a different service than the streaming of a video itself. CQRS is an acronym for Command Query Responsibility Separation, this pattern is based on the notion that the data should not change when someone is looking at it as described by Dahan [12]. If you would look at the traditional CRUD (Create, Read, Update, Delete) data model it can be divided in two types, data altering operations which are the create, update and delete. The other type is read, which only retrieves the data but cannot change it. In CQRS the read operation is done by the query side and the data altering operations, CUD, are handled on the command side.

In the CQRS framework there are three types of messages that convey information in the system.

Query Queries are used to view the data that is in the application, it is a request for data. They are used to get the information that will be displayed for the user. This can be in the form of an instance, information of one person, or an entire set, a table of all persons. But it is not limited

[1] http://afas.nl

to database tables, it can also be another type of file such as an invoice in pdf format. One important thing a query can not do in a CQRS framework is alter the data, queries are read-only.

Command A command is issued when the data of the application needs to be altered. For example adding a person because there is a new employee. After entering the data in the browser it is sent to the server as a command. The command handler verifies whether all required data is in the command and whether any constraints are violated. If it is accepted the browser/user gets confirmation, otherwise an error is returned. It does however not return data itself, as that is the job of the query side, commands can be seen as write-only. The command handler also needs to notify the rest of the system, this is done by sending an event.

Event Events are the internal messages keeping everything updated. The moment that the command handler accepts the command, some data in the system has changed. This event needs to propagate through the system to make sure the query side also knows the latest update of the data. The event can be sent to multiple event handlers, as a change can be relevant for multiple components. The event handlers then update the data store on which the queries are executed. This can be a simple database or as stated before a file, if the address of a company changes it should also change on the invoices that are mailed to that company.

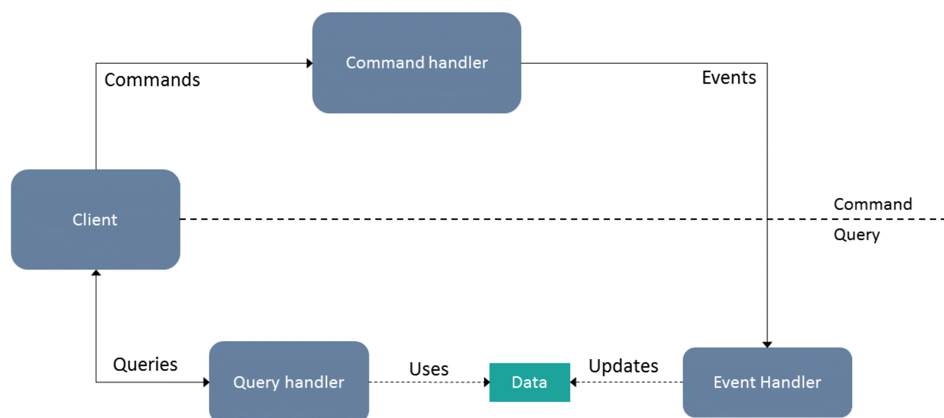

Figure 1: Simplified version of the CQRS framework with the flow of the messages.

Figure 1 shows a simple version of the CQRS framework. Because of the separation it does mean there can be a delay in accepting a change on the command side and being able to show that change to the user, on the

query side. With this architecture we have three locations in which we can measure how the system is performing, commands, events and queries. For each command, event and query we will log the time it was sent and when it was completed, giving us the processing time of each of them. The logging of events contains more information as they have a waiting time as well. There is a queue between the command handler and the event handler, which makes it possible for the command handler to keep accepting commands at a fast pace. The queue dispatcher makes sure the events actually get to the event handlers that need the new information.

2.3 Service Fabric

Service Fabric is a product from Microsoft, a product that is available on Azure, the cloud platform of Microsoft, but is also available to be hosted elsewhere. This technology was being used internally at Microsoft for some time and at this point they are making it a public technology. Examples of services that run on the internal version are, Bing, Cortana, Azure SQL and Halo. AFAS is one of the early adopters of the program. Service fabric is an Infrastructure as a Service (IaaS) solution with abstractions and intelligence, it also contains an orchestrator. Orchestrators make use of machines called nodes, these can have different characteristics like number of cores and size of RAM. The concept of orchestrator relies on having multiple nodes(machines) in a cluster. On these nodes multiple smaller applications or services are being run.

Services are a small part of a bigger application with a public API (Application Program Interface, a set of subroutine definitions) for communication. A service can be partitioned to make load balancing possible, meaning requests within a certain key range get mapped to a partition. Partitions each do the same work but for different data. Each partition is also replicated to make high availability and no data loss possible. So of each partition there are multiple replicas in the cluster, the primary replica handles the requests and updates the secondaries. The secondaries exist to make sure no data gets lost in the case of a failure and to quickly fail over in case the primary goes down. In Figure 2 on the next page an overview of a service is depicted. An important thing to remember is that even though it is now shown as one big component, in reality each replica can be on a different machine. Each service is distributed through the whole cluster.

Actors are an even smaller entity, they are isolated units of data and computing. An actor handles all the computation and data for one specific object/task. For example it handles all the data regarding one product type in the system, then for each individual product type an actor is spawned. Multiple actors are inside a partition of a service and are spawned when necessary. To summarize, these concepts are described below.

Service In the microservice architecture, an application consists of multiple

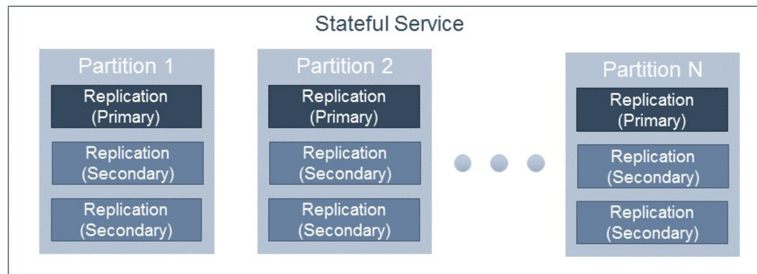

Figure 2: Schematic overview of a service in Service Fabric.

smaller services. They can be stateless or stateful. Stateful services require replication, to make sure the state is never lost. Stateless services, do not use state for their computation, an example could be a service that multiplies two numbers. It gets the numbers and returns the answer but it does not retain any information. Stateless services can still be replicated to make it possible to load balance over multiple machines.

Partition Stateful services are divided into partitions, either by name or by a range of identifiers. This helps keeping the state smaller per partition, therefore a partition is easier to move.

Replica Partitions have multiple copies called replicas, one of these copies is the primary and handles the computation. The secondaries keep their state the same as the primary. So in the case that the primary becomes unavailable the secondaries can take over immediately.

Actor Actors are independent objects that do the computation for only one identifier. They are created on the fly in a stateful service when required.

Service Fabric has its own services running on the cluster, the most relevant to this research are the failover manager and the resource balancer. The failover manager tries to ensure availability, if a service fails it will bring up another copy. Often this is done by having secondary replicas already running and in-sync with the primary on different nodes. The resource balancer tries to balance all services that are run on the cluster. This system takes multiple aspects into consideration, standard things like capacity and load. But also placement constraints like node type or geographical placement.

The final thing the resource balancer considers are fault and upgrade domains, it tries to place replicas of the same service in different domains. The fault domains indicate possible places of a coordinated failure, if you can place replicas in different domains the chance all replicas will go down at the same time is reduced. Fault domains are determined by the infrastructure

and are defined in a hierarchy, from data center to blades in a rack. The upgrade domains are determined by a policy. It is used for rolling out updates without losing availability, it updates all the nodes in an upgrade domain at the same time. By default a policy is chosen which spreads upgrade domains over multiple fault domains. The resource balancer tries to get good service placement using a local search algorithm based on simulated annealing. This algorithm is executed when the cluster is out of balance, the load ratio exceeds a threshold. The load ratio is the load of the node with the heaviest load divided by the load of the node with the lowest load [13,14].

2.4 AMUSE

The AMUSE research project[2] is an academic collaboration between Utrecht University, Vrije Universiteit Amsterdam, and AFAS Software to address software composition, configuration, deployment and monitoring on heterogeneous cloud ecosystems through ontological enterprise modelling.

[2] http://amuse-project.org

3 Configurations

We want to optimize the infrastructure that will run the software in such a way that the company is content. To be able to optimize the infrastructure we need to look at what can we change to make an impact on the infrastructure. The configuration will contain parameters that are relevant to the infrastructure and the performance of the software. Parameters that have an impact, can be on one of two levels. They can be on the hardware level, the physical resources provisioned to the system. Or they can be on the orchestrator level, these parameters influence how the orchestrator behaves. All these parameters come together in the infrastructure and make it possible to run the application. They influence the performance of the application and how well it can deal with failures in the system. We want to configure these parameters to get the performance we want for a price we can afford. In every environment you will encounter different possible parameters to tweak. It is important to carefully choose which of these parameters will make up your configuration. If you pick too many to configure it becomes difficult to find the impact of different factors, this is due to the Curse of Dimensionality described by Bellman [15]. In order to get a statistically reliable result, the amount of data needed to support the result grows rapidly (often exponentially) with the dimensionality. So you will want to choose those parameters which are relevant to what you want to achieve and not make every parameter part of the configuration as it will diminish the quality of the optimization. This choice can be made based on domain expert knowledge or results of other experiments.

3.1 Hardware Level

The hardware level has a lot of different parts which could impact the overall performance of the system. Examples of parts that influence the performance are, the processor, RAM and disks, as these are being the main influences on processing time and the access time of the system. However there are a lot more choices which could impact performance, such as graphical cards for heavy parallelization, network interfaces for more throughput or connectivity. Another thing that can vary is the cluster size, how many servers are we using. Even outside of the main servers you can change parts of the systems configuration, think of things like hardware firewalls, routers and cables.

The bare machines approach could be a problem, either all the components need to be swapped for each test that needs to be done or have all the configurations completely in the first place. As components cost money and if you do not have a use for them after the test it is a costly way to do your testing. Especially if you need multiple machines for a configuration. To test multiple configurations you would need to update the machines in between

each test, requiring manual attention. Therefore it is useful to make this level a bit more abstract, moving away from these properties that come with physical machines. This can be done by using a virtual machine approach, Virtual Machines (VMs) can be given certain properties to simulate different types of real machines, while using the same hardware to host those VMs. This also enables us to not have the hardware ourselves at all. We can now move towards the cloud, in the cloud VMs are the way to rent server space. The cloud provider has all kinds of hardware available as there are a lot users. This makes it easy to support different types of VMs and to create them quickly. Instead of having to rebuild the hardware for each test you just need to wait 15 minutes and all VMs are ready to go. If you pick the types in a way such that they represent differences in the components you are interested in, you can deduce what your hardware configuration should look like.

This does require a footnote, with virtual machines you always have a little bit of overhead. The (physical) hosting machine uses some of the resources to make the virtual machine run. Another thing that could impact the performance is noisy neighbour, it is possible to run multiple VMs on one physical machine. If they are all running full power they can affect each others performance.

3.2 Orchestrator Level

The orchestrator level is very specific, it fully depends on what the orchestrator exposes to the user. This differs between available and home-made orchestrators. *To orchestrate* means to arrange, organize or build up for special or maximum effect. In the context of software deployment this means, it arranges placement of the microservices on the servers, replicating the data, updating of the application, etc.

We will look at Service Fabric from Microsoft as this is used at AFAS. Other orchestrators might have different but in concept similar options, as in concept they want to achieve the same things. Service Fabric uses metrics to indicate how much load there is on a node. The default metrics that are tracked are.

Count The number of replicas on a node.

Primary count The number of primary replicas on a node.

Replica count The number of stateful replicas on a node

The reason for tracking the stateful services is because these require storage space, while stateless replicas do not and are therefore less interesting to track. The primaries are tracked to make the system more robust, by spreading the primaries over multiple nodes the fail over can be managed

easier as fewer primaries will be hit at the same time. These default metrics can be extended with custom metrics, which can also be dynamically reported. The microservices that run on Service Fabric can report load on their own. This can be based on how many items there are in the queue or how many events have been handled in a certain time period. We can report different types of resources, however performance counters such as CPU-time or memory used is not available [16]. This is due to the structure of Service Fabric, it aggregates multiple partitions and replica's in the same process and all performance counters work on the level of process.

Service Fabric offers a lot of parameters that can be tweaked. We are interested in the resource balancing part of these parameters. The parameters are extracted from the online Microsoft documentation written by Snider [13, 14, 17]. There are numerous more settings but they do not have an impact on the resource balancing and that is why they are left out. In the following list the most options have an interaction with the metrics that were just explained above, as these indicate how much work is being done in a microservice.

Replication count How many replications of the microservice are spawned. With more replica's you are able to handle more failure. It takes more space and computation to keep them all up.

Partition count How many partitions the microservice is split into. Each partition handles a part of the input. This is based on names or hash ranges.

Balancing Weight How important is a resource metric for a microservice. A service might have a lot of state and rates metrics regarding to memory/space higher, while another service is computationally intensive but does not use space and is therefore not interested in metrics regarding to space. These weights are given on an ordinal scale of; Zero, Low, Medium, High. The resource balancer will use these weights to determine where a service will be placed, paying the most attention to the resource labelled with High for that service.

Balancing Thresholds If a metric ratio, the highest load on a node divided by the lowest load on a node for a specific metric, exceeds the balancing threshold for that metric, Service Fabric will start rebalancing the microservices over the cluster.

Activity Thresholds This is an extra threshold requirement which can be set. In this case the highest load should exceed the activity threshold otherwise the rebalancing will not trigger. This is to make sure a high ratio will not trigger rebalancing when the load is still low.

Capacity How much load of a resource metric can a node hold. For example a node can handle up to X events per second. The balancer will respect these constraints and will roll-back application upgrades if necessary.

Buffer ratio This can specify a buffer in the capacity, this buffer can only be used by the balancer if there are node failures or maintenance.

The first two parameters are per microservice. The balancing weights are defined for combinations of microservice and resource metric. These three parameters are all based inside one application, so if there are multiple application instances on the same cluster these parameters can differ. The rest of the settings are per resource metric and are therefore not bound to an application, as multiple applications can use the same resource metrics. So these are Service Fabric cluster wide parameters.

If you would use all these parameters at the same time the number of dimensions in a configuration gets very large. This negatively affects the results of the optimization, as it becomes harder to extract which parameter influences the results. The choice for using particular parameters as a dimension in the configuration should follow the idea of impact, which of the parameters are expected to influence the infrastructure in a way it affects the objectives. In our case we use virtual machines, balancing weights and balancing thresholds. In total the configuration consists of 13 dimensions, the specification will be handled in Section 8 where the full case is described.

4 Optimization

The problem we are trying to optimize has multiple objectives. In the SaaS industry this often means cost versus performance, but there are more possible objectives such as security, scalability or stability. In our case we will have four objectives: costs, command performance, event performance and event robustness. A configuration consists of 13 dimensions, each value represents a parameter for either hardware or orchestrator the precise formulation of the configuration and objectives will be given in Section 8. To make this more abstract, we say F is a set of objective functions. For the sake of simplicity we assume that all objective functions need to be minimized, you can easily swap between maximizing and minimizing by adding a minus sign in front of the function. We can now formulate the optimization problem.

$$\begin{gathered} \min_x \quad f_j(x) \in F \\ s.t. \\ \forall i \quad l_i \leq x_i \leq u_i \\ \text{optionally } x_i \in \mathbb{Z} \end{gathered}$$

Minimizing all objective functions in F at the same time by finding the optimal configuration x, while the values of x_i are bound with lower l_i and upper u_i bounds. Some parts of the configuration are categorical, meaning they can not be represented in rational numbers but require to be integer($x_i \in \mathbb{Z}$). There are no other constraints on the configuration. Minimizing all objective functions at the same time is not always possible, if one of the objectives gets better another can get worse. The example in software is the costs of hosting the software versus the performance, it is not possible to have both at their respective minimals with the same configuration.

4.1 Reducing to Single Objective

A possible solution to multi objective optimization is reducing it to a single objective optimization. This can be done by scalar or weighted sum of all the objectives. This combines all the objectives into a new objective function. This gives a weight to each of the objectives, meaning the trade-off between objectives is now explicitly known. This makes it possible to end up with a single optimal configuration. The optimization formulation then becomes.

$$\begin{gathered} \min_x \quad \sum_{j=1}^{|F|} c_j * f_j(x) \\ s.t. \\ \forall i \quad l_i \leq x_i \leq u_i \\ \text{optionally } x_i \in \mathbb{Z} \end{gathered}$$

	x	x'	x''
f_1	2	3	1
f_2	2	2	3

Table 1: x and x'' are non-dominated, x' is dominated by x.

The objective is now a weighted sum of all original objectives, the constraints did not change. As we are now minimizing for just one value there also exists one minimum value. Even though multiple configurations could have the minimum value, returning only one of them is sufficient as they are all equal in the optimal sense.

The only problem that remains in this approach is how to weigh each of the objectives. It is really difficult to determine these weights, being abstract weights it is hard to imagine the impact on the solution. Furthermore the moment you start weighing the objectives is the moment the optimization will return a *subjective* optimal solution. If someone else has other priorities for the objectives, there is another optimal solution. This would mean you need to re-run the optimization.

Another solution that also suffers from the subjectivity argument is the (epsilon)ϵ-constraint method. This method makes all objectives except one into a constraint, which results that those objectives can no longer be larger than a certain epsilon. This again means you need to give these restrictions early in the process, and re-run if you want to relax or tighten up one of the constraints. Also if you are able to do this, you should have started with a single objective optimization as apparently the other objectives can be represented as a constraint. This notion of ϵ-constraints can be used in another way which is described below.

4.2 Pareto Optimality

There exists a solution that enables us to make the subjective decision after the optimizing step. This is achieved by generating all possible "optimal" solutions. All these solutions together are called the Pareto-front because they are Pareto optimal or efficient, named after an economist Vilfredo Pareto who defined the concept as explained by Ehrgott [18]. A solution is Pareto optimal if it is non-dominated. To know what non-dominated means we first need to know what dominated means.

Solution x is dominating x' if and only if x is equal or better than x' in all objectives and there is at least one objective where x is strictly better. The mathematical notation of this concept is:

$$x \succ x' \Leftrightarrow \forall f \in F \quad f(x) \leq f(x') \quad \wedge \quad \exists f \in F \quad f(x) < f(x')$$

Table 1 shows examples of (non-)dominance. x is better than or equal to x' in both objectives and strictly better in objective 1, so x dominates x'. x''

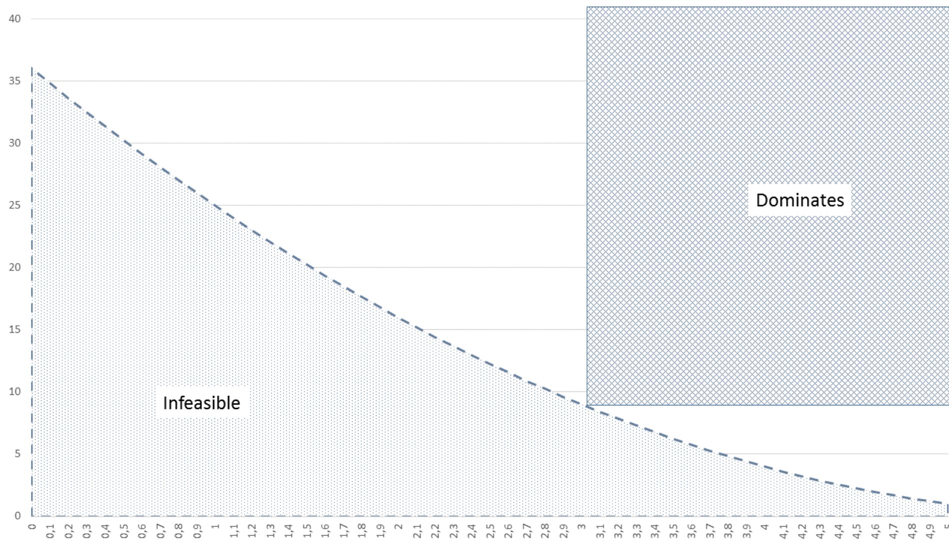

Figure 3: Pareto front example, (3,9) dominates the rectangular area.

does not dominate and does not get dominated because it is best in objective 1 and worst in objective 2. Now we know what dominance is we can define the Pareto front mathematically.

$$P(X) = \{x' \in X : \{x'' \in X : x'' \succ x'\} = \emptyset\}$$

The equation above states that a solution is only part of the Pareto front if there does not exist a solution that dominates it, by showing that the set of dominators for x' is empty. If we take Table 1 as the solution space, $P(X)$ would contain x and x'', as both these solutions stay non-dominated. Another example is given in Figure 3, all points that lay on the dashed line between feasible and infeasible would be part of the Pareto front. It also shows how one solution dominates a part of the solution space. If a point would lay on the vertical or horizontal line of the dominated square it would mean that it is equal in one of the objectives but worse in the other objective. In reality the boundary between infeasible and feasible does not always follow such a clean curve, it is entirely possible to have a jagged Pareto front.

Creating the Pareto front by adaptations of (Integer) Linear Programs has been studied extensively. Such as ϵ-constraint method by Haimes [19] which is capable of generating the entire Pareto front. This is done by iterating through each combination of epsilons, solving to optimality and save that optimal point to the Pareto front. Generating the entire front is expensive so more research has been done into generating an approximation or subset of the Pareto front. Examples of this are scalarization described by Veerapen et al. [20], multi-level programming described by Bialas and Karwan [21] or

goal programming described by Charnes and Cooper [22]. These methods create a subset of the front by smoothing out the jaggedness of the front, they can only find the peak values of the jagged teeth. Methods to generate an approximation of the Pareto front are evolutionary algorithms. Examples of this are Pareto Archived Evolution Strategy (PAES) by Knowles and Corne [23] or Non-dominated Sorting Genetic Algorithm II (NSGA-II) by Deb et al. [24, 25].

4.3 Non-dominated Sorting Genetic Algorithm

We have chosen to use NSGA-II as the optimizing algorithm. In the choice between evolutionary algorithms and linear programs, we have chosen for an evolutionary algorithm firstly because not all objectives can be expressed in a linear function. As the relation between the input and the output does not have to be linear, in the case of SaaS performance this relation is more complex. This removes the possibility of using linear programs to solve the problem. Besides linear programs evolutionary algorithms are used often in multi objective optimization problems. A comparison among multi objective evolutionary algorithms has been done by Kunkle [26], this showed that NSGA-II is one of the most used algorithms. NSGA-II is capable to deal with noisy objectives and is able to find the solutions on the outlying edges of the Pareto front. For making a trade-off we want a wide spread of solutions, NSGA-II provides this by finding the outlying solutions. The sorting of solutions makes this algorithm capable of finding a broad range of solutions. It uses the same domination property as the Pareto front, to rank solutions but also thinks about what evolutionary algorithms need, namely variation in the population. How this works is explained in the rest of this section, it follows the structure of NSGA-II that is described in Algorithm 1.

```
Seed Population;
while Stop criterion not reached do
    Select Parents;
    Create Children;            /* Crossover and Mutation */
    Evaluate Children;
    Rank Population;
    Reduce Population;          /* Survival of the fittest */
end
Return best solution;
```

Algorithm 1: General structure of Non-dominated Sorting Genetic Algorithm.

4.3.1 Seeding population

In genetic algorithms we use names from biological origin, the set of solutions in the algorithm is called a population. A solution consists of n values, one for each of the dimensions in the configuration. Each iteration of the algorithm is called a generation, the old solutions become parents and create new solutions, children. To start the algorithm we need an initial population. The solutions in this population are required to be diverse, because of the inheritance of the generations. If all starting solutions are equal the children will resemble the parents greatly, except for mutations. The starting solutions can be chosen in various ways as long as the solutions are not identical, per default they are randomly generated.

4.3.2 Creating Children

In NSGA-II we use tournament selection for the parents, which means we pick two solutions at random and the better one becomes a parent. We will define shortly what a solution makes better than another, in 4.3.3. This is twice which results in two parents that have been selected, these are used to create two children. These children are created by crossover, in our case one-point crossover. This crossover picks a random spot in the solutions and the children each get the start of the solution from one parent and the last part from the other parent. The child solutions can have a mutation, which manifests as a change in one small part of the solution. This process is repeated until there are equally many old solutions as new solutions. The new/child solutions need to be evaluated for each of the objectives.

4.3.3 Ranking the population

After creating and evaluating the child solutions we get a NSGA specific step, determining which solutions are better than the other. The first distinction is the dominance rank, the Pareto front to the solutions or non-dominated set of solutions gets the best dominance rank. Then excluding those solutions we look for the next non-dominated set, these get the next dominance rank. Repeating this until all solutions have a rank, this way we have an ordering which follows straight from the property of dominance. This is however a long running process with a running time of $O(MN^2)$, where M is the number of objectives and N the number of solutions. An improvement on this running time was made by D'Souza et al. [27], their approach reduced the running time to $O(MN \log(N))$. This is done by sorting each objective independently. Than taking the sum of the positions in each sorted list for a solution. Each solution with the same sum is on the same dominance rank. An example of dominance ranking is given in Figure 4 on the following page.

So we now can calculate different ranks, however we have yet to make a distinction between solutions that are in the same rank. Since we can not

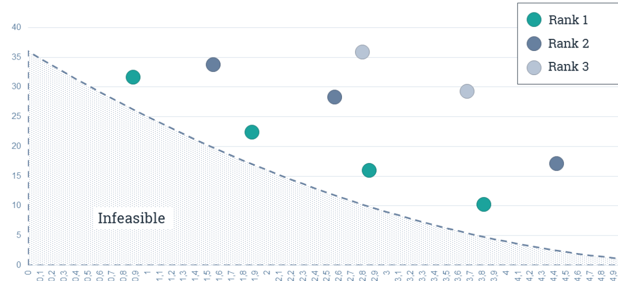

Figure 4: Example of dominance ranks.

differentiate in optimality as we do not have priorities or weights, we will sort the solutions in a way that is suitable for genetic algorithms. An important part in genetic algorithms is variation in the population, if the solutions in the population are all nearly the same it is hard to get new solutions we have not yet seen before. But if we have some variation in the population it is easy to find new solutions by combining two old solutions. This means that more of the possible configurations are being considered during the search. The distinction between solutions that are in the same rank will be uniqueness, this is measured by taking the crowding distance. Similar solutions are close together, but when a solution is unique there will be less solutions crowding around him. So we want to prefer solutions that have a large crowding distance. This distance is calculated by taking the sum of the normalized Euclidean distances between itself and all neighbouring solutions in the same dominance rank. It is a neighbour if it is the next or previous best solution in one dimension. If either of those do not exists it, the solution gets a very high distance because it is an extreme value of that dimension. So we can now sort the solutions within a rank and get an overall ordering in solutions. Using this ordering we can determine which solution is better during the selection of parents and when the population is reduced in size.

To give an example of the crowding distance calculation we use an optimization problem with two objectives. The lowest and highest found values in the population determine the range which will be used to normalize the dimensions.

$$\text{range } r(f_1) = 8$$
$$\text{range } r(f_2) = 16$$

For each dimension there are two neighbours on the dominance rank or one if the solution is on the extrema of that dimension, in rare occasions there is a rank with only one solution which leaves no neighbours. In this example we use two neighbours per dimension, meaning we have four neighbouring

Solution	$f_1(x)$	$f_2(x)$	$\frac{\|\|f_1(x_1)-f_1(x_i)\|\|}{r(f_1)}$	$\frac{\|\|f_2(x_1)-f_2(x_i)\|\|}{r(f_2)}$
x_1	4	6	-	-
x_2	3	16	0.125	0.625
x_3	5	2	0.125	0.125
x_4	1	5	0.375	0.0625
x_5	7	7	0.375	0.0625

Table 2: Crowding distance example data.

solutions. The solution for which the crowding distance is being calculated is x_1, this solution and its neighbours and their respective objective values are shown in Table 2. In the last two columns the normalized distance per dimension is shown. As example this is how the distance in dimension $f_1(x)$ is calculated between solution x_1 and solution x_2, $\frac{||f_1(x_1)-f_1(x_2)||}{r(f_1)} = \frac{||4-3||}{8} = \frac{1}{8} = 0.125$. The total crowding distance for x_1 is the sum of all the values in both columns, which results in a crowding distance of 1.875. In the case that a neighbour is missing we add n to the crowding distance, where n is the number of dimensions. This ensures that the crowding distance of extreme points is always larger than points in the middle of the population.

4.3.4 Reducing the population

For reducing the population NSGA-II uses an elitist scheme as explained by Deb [25], which means it will pick the best half of the population (old and new solutions combined) and toss the rest away. With this approach the fronts are preserved as the first sorting criteria is the dominance rank. Only one dominance rank can be chopped up, some of the solutions from this rank will continue in the population and some will be removed. This is done based on the crowding distance, preserving variation in the population.

4.3.5 Stopping the algorithm

As a stopping criterion we have multiple options. We can set a fixed amount of time, when this is elapsed we stop the algorithm. Another option is running the algorithm for a fixed number of generations. The last option is a number of generations without improvement in fitness of the optimal solution, there is an adoption of this criterion for multi objective which follows from an additional step in the algorithm explained below. Obviously it is also possible to set multiple stopping criterion, so when either of them is met the algorithm finishes.

4.3.6 Archive extension

The size of the Pareto front can exceed the population size of the genetic algorithm. This means it is possible to throw away candidate Pareto optimal solutions when reducing the population. That is why there have been extensions of NSGA-II which include archiving, this has been described by Goel et al. [10] and Hassan et al. [28]. Each generation we archive the non-dominated set of the population, which are the solutions in the first dominance rank. This way we can be sure we do not lose any Pareto optimal solutions. The only thing is that newly added solutions could be dominating old solutions in the archive, so these old solutions will need to be removed. The stopping criterion of not improving the fitness in a certain amount of generations is based on this archive. Instead of improving the fitness we look at improvement of the archive, when a solution is added to the archive we see this as an improvement of the fitness of the Pareto front. When the stopping criterion is reached the algorithm stops and returns the full archive. Figure 5 shows the flow of the Non-dominated Sorting Genetic Algorithm with the archive extension.

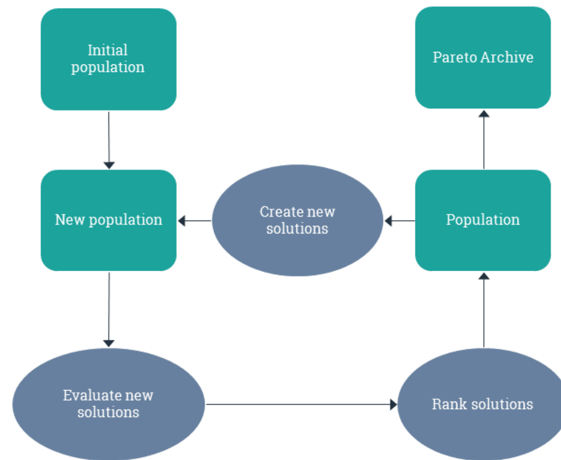

Figure 5: Schematic flow of the optimization.

4.3.7 Evaluation of the solutions

A problem with genetic algorithms is the number of evaluations. In each generation all the new solutions need to be evaluated, which means hundreds of thousands of evaluations during the entire optimization. Calculating the price of a given configuration is easy, there is a price per machine and you know how many machines there are in the configuration. There is no problem with doing these calculations for every single configuration. But evaluating performance of a configuration is much more time consuming,

you would need to run the entire program with a workload and collect the data to analyse. Even if this could be done in a couple of minutes, which realistically we can not, this would result in a running time in the orders of years for the optimization. To mitigate the evaluation time we will use approximation functions or heuristics, more on this in Section 6. Using these heuristics we are able to evaluate at a much quicker rate and making it feasible to run the genetic algorithm. Heuristics require training data, you need examples to be able to approximate. The training should be done with a good representation of the entire solution space, this will be explored in the next section.

5 Sampling

We need to reduce the evaluation time of a configuration to make the genetic algorithm run in a feasible time. This will be done with the help of heuristics. To make these heuristics accurate we need an accurate view of the entire configuration space i.e. all of the possible combinations of hardware and orchestrator settings. This is done by sampling the possible configurations, each sample will be evaluated with the original method, simulating user activity on the system, so we can use its outcome to train the heuristic. To get the most accurate heuristic we should sample each individual configuration, but that would defeat the purpose of sampling as it would become a brute force calculation. We want to save time and money by this approach, each sample takes just that, time and money. We want to take few samples but these samples should accurately represent all possible configurations.

5.1 Sampling Strategy

To make sure the configuration space is sampled correctly we need a sampling strategy, a way to pick the configurations to use. The most basic strategy is random sampling, for each dimension of the configuration taking a random value between a lower and upper bound. Each sample is taken without knowledge of each other meaning there is no guarantee on the spread of the samples. This strategy can give you all kinds of results but there is no certainty whether the samples represent the entire configuration space, statistically the samples will avoid the edges of the configuration space, as it is unlikely to randomize all dimensions in their respective extremes.

To get a more representative view of a sampling space Neyman [29] created a strategy called stratified sampling. With this strategy the sample space is split up, creating mutually exclusive subspaces called strata. Then you can use random sampling within such a strata. By splitting up the sample space, it is certain that all possible subspaces are reached and represented in the data which is used for the heuristic. In this case you have to sample each subspace, which means the subspaces become large or the number of samples increases. To decrease the number of samples, McKay et al. [30] thought of a new strategy called Latin hypercube sampling. This takes the same stratification of the configuration space, but now for each sample the subregion is remembered and the future samples can not be in the same subregion that shares the same range in one of its dimensions. So each sample restricts in which subregions the following samples can be taken.

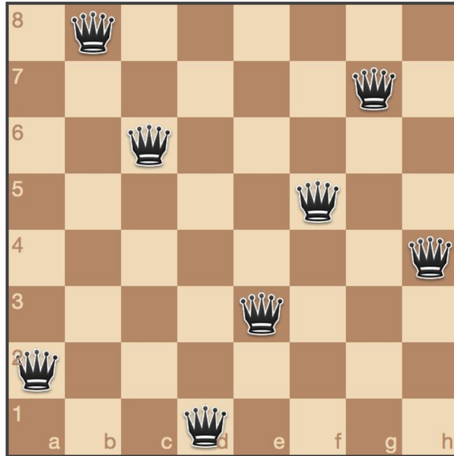

Figure 6: Solution to the queens problem.

5.2 Orthogonal Design

Orthogonal Latin hypercube (OLH) is the next step in this direction which was developed by Ye [31]. This way of creating a sampling design requires to determine all samples at once. Because of the orthogonal property, each pair of samples should be uncorrelated, with the previous method you could end up with correlating samples as not every sample was determined beforehand. To illustrate how orthogonality works, we give a two dimensional example in the form of a game. There is a challenge with a chess board (8x8), by placing eight queens on the board. However those eight queens may not be capable of capturing each other. This means they can not be on the same row, column or diagonal. A solution to the problem, there are multiple, is shown in Figure 6. If you would use this placement as a basis for sampling, the samples are orthogonal.

The problem with OLHs is that they are hard to construct for higher dimensions (larger than 7) and do not have a guarantee to fill the configuration space equally, it is possible to have large spaces without samples. That is why Cioppa and Lucas [32] relaxed the orthogonality property and added a space filling property. Creating the so called space filling nearly orthogonal Latin hypercube(NOHL), this is done using two space filling measurements Euclidean maximin distance and modified L_2 discrepancy. To keep the Latin hypercube nearly orthogonal they set a maximum on the pairwise correlation between samples, which may not be exceeded. Based on this research Sanchez [33] created a template for the best found designs for configuration spaces up to 29 dimensions. We only need to fill in the lower and upper bound and the number of decimals of each dimension into the template. It then gives us the best space filling nearly orthogonal Latin hypercube for

our problem. The number of samples that need to be taken are dependent on the number of dimensions. The number of dimensions are rounded up to the nearest k where:

$$k = m + \binom{m-1}{2}$$

Then the number of samples generated by the NOHL design is equal to:

$$n = 2^m + 1$$

As example, for $m = 6$ we can cover up to $k = 16$ dimensions with $n = 65$ samples.

These sample points will be our initial samples, if necessary we can permute the columns to get more sample points. This will retain the same near orthogonality and space filling properties while giving more sample points, according to Cioppa and Lucas [32]. Only the center sample point should be excluded because that will be the center no matter how the columns are permuted.

The near orthogonal Latin hypercube design fulfils the sampling properties we are looking for. It uses a low number of samples while guaranteeing an unbiased evenly spread sampling design through the entire configuration space. The NOHL sampling design is therefore chosen as our design to sample for the heuristic training data. These sample points will be evaluated using simulated user activity to get the output data required to train the heuristics.

6 Heuristics

As explained in Section 4, we need to speed up the evaluation of a configuration to make the genetic algorithm run in feasible time. To do this there are two options, heuristics and simulation. In a simulation the real-life situation is modelled, each step of the situation is calculated. To run a simulation a great understanding of the process is required, as each step needs to be modelled. Therefore simulation is not always possible, the inability in our case comes from the complexity of the software and closed box nature of Service Fabric. We are unable to model each step in the process, so simulation is not an option for our case. Heuristics also model the real-life situation, but it ignores each step in between. A heuristic is a model from input to output and an approximation of what happens in the middle. As we are interested in the output and do not have the information of all the steps of the process we will use heuristics. In this section we take a look at a couple of different heuristics. When using heuristics it is useful to compare multiple types as each heuristic has its strengths and models certain types of functions. This means that it is possible that for each output dimension you have, a different heuristic is approximating reality the best.

6.1 Linear Regression

Linear models are the simplest models. These models assume there is a linear correlation between the input variables (predictors) and the output variables (responses). Meaning if you would plot it you would get a straight line without curves or gaps. In the simplest form with only one predictor and one response variable you get the basic formula of $y = ax + b$ where a determines the slope of the line and b the offset. We however have more than one predictor in our problem, each dimension in the configuration is a predictor. We can extend this formula to n-dimensions.

$$y = b + \sum_{i=1}^{n} a_i * x_i$$

Training this model means we need to get the line as close as possible to all data points at the same time. To measure how close the data points are we use the squared sum of residuals. The residual is the difference between the real response value and the result of the linear equation, which results in the following formula where k is the number of samples.

$$SSR = \sum_{i=1}^{k} (y_i - AX_i + b)^2$$

To get a line that fits the data best, the formula needs to be minimized by changing A and b. With this model you can now, given a new point X' predict y' by entering the point into the model.

6.2 Non-linear Regression

Not every relation between the predictor and response variables is linear. This also presents the difficult portion of creating a non-linear model, determining the shape of the formula. There are infinitely many different formulas that possibly fit the data the best. With two or three dimensions you could make an educated guess by looking at the plots. However when we have more than three input dimensions, it becomes hard to create a meaningful plot to deduce a type of function.

For our test case we use a squared function for the non-linear approach, this includes pairwise interaction of the dimensions. As a lot of the dimensions we use functionally affect each other. For example we have the number of virtual machines and the type, the balancing weights for a microservice or the combination of the balancing weights with the ratio threshold. By incorporating this information in the heuristic it might give better results.

In the non-linear case, minimizing the squared sum of residuals is still the method that is used. So we want to have results of the heuristic that resemble the reality by changing the constants and scaling factors in the model.

6.3 Kriging

Kriging uses the sample points themselves as the predictor and is described by Cressie [34]. This means there is no real training involved, the data points themselves are everything that is needed. This also means the heuristic can be easily updated with new data points without re-training. When you want to evaluate an unknown solution it uses a weighted average of nearby sample points to determine the value of the unknown solution. The method comes from geostatistics where it is used to get an estimation of the land by sampling, originally it was used in finding the best place to mine for gold, this was part of a Master thesis of a student called Krige after whom the method is eventually called. The method has since then spread to other disciplines as the approach is applicable to approximations in general. It is a linear interpolator, as the computation is done with the neighbouring samples. This does mean this heuristic is lacking when trying to extrapolate solutions, which is described by Santana-Quintero et al. [35]

6.4 Neural Network

Neural networks are inspired by how our brains and neurons work, it is described by Broomhead and Lowe [36]. There are nodes (neurons) and arcs (synapses), the arcs connect the nodes to transfer information just like how synapses connect neurons in our brains. Each arc has a weight which is multiplied with the value of the source node, the target node gets all the weighted information and puts this through an activation function. The

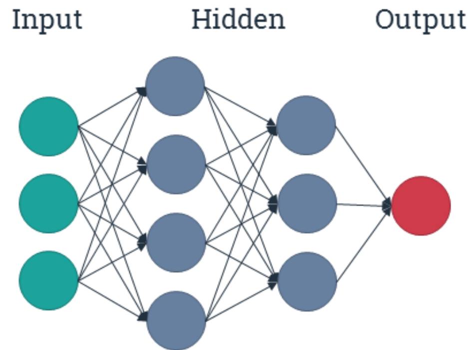

Figure 7: Example of a multilayer perceptron (Neural Network).

output is then sent through the outgoing arcs to the next layer of nodes. There are three types of nodes, the first type are the hidden nodes which are explained above the other two are different as they deal with the input and output of the system. The nodes are ordered in layers, there are only arcs between two consecutive layers in the multilayer perceptron as seen in Figure 7, in other types of neural networks this constraint is not necessary. The first layer is not hidden but contains the input nodes, each input node takes one of the predictor variables and sends this into the network. The last layer contains the output nodes which give the final response variable, so in our case the estimation of an objective. An example of this structure is given in Figure 7, the number of nodes in each layer and the number of hidden layers can vary. In our case we use a RBF network, which is a single hidden layer network with radial basis functions as activation function. Radial basis function depends on the distance from the input to a basis point or center. The function can vary, in our case we use the most common function which is a Gaussian function.

$$\phi(r) = e^{-(\epsilon r)^2}$$
$$\text{where } r = ||x - x_i||$$

During training each of the samples is fed through the network. The result is then used to give feedback and update the weights on the arcs and the epsilon in the activation function. After doing this the network is ready to use and predict new samples. For new samples the input variables are entered into the input nodes and the result will roll out of the output node.

6.5 Support Vector Regression

The support vector regression is based on support vector machines. The idea starts the same as with most heuristics, we need to find a function such that all response variables are at most ϵ away from the value of the function. In this method we use the dot product of the input variables and a vector w, which the is capturing the model. This can be formulated as an optimization problem like this:

$$\text{minimize} \quad \frac{1}{2}||w||^2$$

$$s.t.$$

$$y_i - \langle w, x_i \rangle - b \leq \epsilon$$

$$\langle w, x_i \rangle + b - y_i \leq \epsilon$$

Where y_i is the output value, x_i is the input vector. The idea is that the training samples all lay within the high-dimensional tube which has a radius of ϵ. Later this hard constraint has been softened to allow for outliers and not create anomalies in the heuristic just to sustain those constraints. This is done by introducing slack variables, that are also a part of the objective function. However the method now takes some steps transforming the problem. The result is actually what gives this method the name Support Vector, after the transformations there are a number of input data points still relevant, the supporting vectors. These points lay on the boundary of the ϵ-tube and define the form of the tube. For new points we can make a prediction by taking a combination of these support vectors, using a kernel function and weights that have been trained. [37, 38]

6.6 Accuracy

To train the heuristics we use samples which have their input variables and output variables known, all samples together are called the training set. The heuristics are trained using this training set. We would like to know how accurate a heuristic is. This is done by validating the heuristic with other sample points, the validation set. If the data points to validate the heuristics are equal to the original training set, the validation will favour the heuristic that is fitting tightly to the original dataset. However this might be too tight, resulting in erratic approximations for data points that are not close to the original set or the training set might have included an outlying sample which dominates the heuristic. This is called over fitting or over training and should be avoided.

To measure how accurate the heuristics are, we use the root mean squared deviation or RMSD. This is calculated by taking the difference between the predicted value and the observed value. This is then squared to make sure that negative and positive deviations do not cancel each other out

when taking the average as well as punishing large deviations even more as they are not acceptable. The root mean squared deviation is mathematically defined as:

$$RMSD = \sqrt{\frac{\sum_{j=1}^{n}(\hat{y}_j - y_j)^2}{n}}$$

All the algorithms have parameters which can be set, you could optimize these parameters with the help of the RMSD. However this can also lead to over fitting, this time you are actually fitting towards the validating data set. In this research these parameters are not further investigated and the default settings are used.

At this moment we have not yet chosen which heuristic to use. As for each problem the type of function is different we cannot choose the best heuristic. We will pick the heuristic which came closest to reality, the heuristic with the best accuracy. We can only determine this after training and testing all of the heuristics, which will be done in Section 8. After comparing the heuristics we will use the most accurate heuristic in the evaluation step of the optimization.

7 Presentation

Given that we now have the tools to run the genetic algorithm, it will result in an approximate Pareto front which can contain a lot of configurations. So the next problem we face, the final sub research question, is to present this data in a clear way. We want to inform the decision maker with the different possibilities. But we need to make sure we feed the information in such a way that he does not feel overwhelmed. The first thing we can do is, only show the output variables and not the input variables. The trade-off should be made based on the merits of the configuration and not on which setting it uses to get there.

If however you are looking for more insight in the configuration and are less interested in a decision it is more useful to analyse the configurations themselves. For instance, are there settings converging on a specific value, meaning that there is an optimal way to use that setting? Looking at the configuration is more about analysing than it is about presenting the data. In this section we look how to convey the information by presenting the results.

7.1 Visual Presentation

Originally our data is just a table with rows for each configuration and columns for the different output values. Humans can read tables easily, we can find values we want to know. However it is hard to reason about the data in the table, in the sense of patterns, trends or exceptions. It is also hard to see the relative position of configurations, the differences between configurations are less noticeable. Furthermore the Pareto front can contain thousands of configurations, making a table unusable. If however the same data is presented graphically our brains can do these things with ease, this has been researched by Friendly [39] and Few [40]. As we have a multi objective optimization problem we have at least two dimensions to represent in our visualization, for instance in the case with Next we have four dimensions. The first option to visually enhance the table is by using colour-coding, by simply adding a gradient to the columns we are able to spot the differences. This serves good as a first indication but is limited to what it can achieve. The following parts will show graphical representations of high dimensional data.

7.1.1 Scatter Plot

Scatter plots are used to display the values of the configuration in a coordinate system. Where each axis represents one of the variables, the most often used scatter plot only uses two axes, but a three dimensional scatter plot is also possible. In the case of a three dimensional scatter plot it is useful

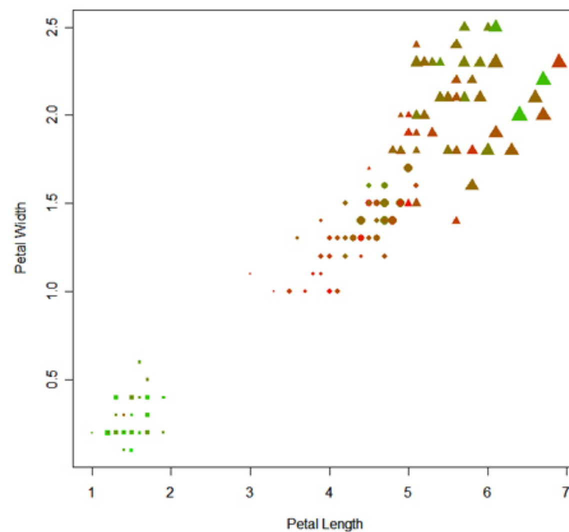

Figure 8: Higher dimensional scatter plot example (Iris data set).

to have it interactive, so you can rotate around the plot and get a better grasp of the location of each dot. Even though we only can visualize three dimensions on the axes, we can add more information in different ways to the scatter plot.

Colour You can add a colour-coding for a dimension. Either categorical giving each category its own distinctive colour. Or you can use a gradient to show a continuous variable.

Size You can change the size of each point in the plot. Giving a larger point to higher values in one of the variables.

Shape You can also change the shape of the points. This is only possible for categorical variables. Each category can then be bound to a shape such as; circle, square, triangle or cross.

With these additions we can reach six dimensions in a scatter plot. However you should pick the representations carefully for each dimension. As it can make such a plot intuitively or fail to convey the information properly. An example with the iris data set is given in Figure 8, where two axes are used and the three other representations described above.

Another option using scatter plots is a pairwise scatter plot. This uses a scatter plot for each pairwise combination of variables. These are placed in a matrix, an example with the Iris data set in shown in Figure 9 on the following page. Even though all relations between paired variables is represented, it is hard to make a trade-off with this representation as the connection between more than two dimensions becomes unclear.

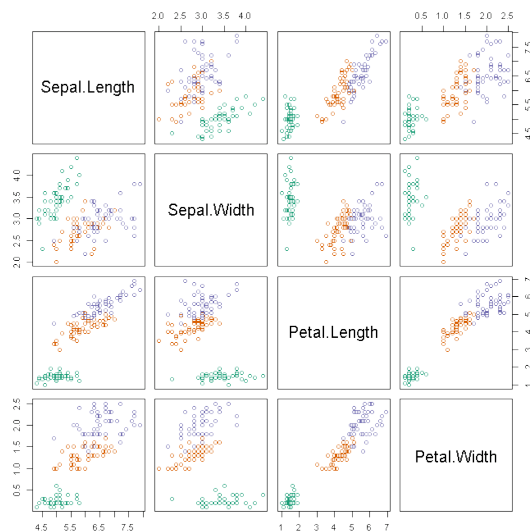

Figure 9: Example pairwise scatter plot matrix (Iris data set).

7.1.2 Radar Chart

Radar chart is sometimes called a star, web or spider chart, because it can resemble each one of these. Each dimension has an axis between each of them is an equal sized angle. The axes are normalized, they are all equal length. A configuration is plotted by connecting the points on each of these axes. This can result in an image which resembles a star, which is why it is sometimes called a star chart. Figure 10 on the next page shows an example of a radar chart with four data points. If we would try to plot all our data points on one radar chart, it would be hard to make any distinctions when the number of data points represented goes higher than twenty. So either we would need multiple charts to plot the points in or we need to reduce the number of points we want to present at the time. A drawback of radar charts is that the comparison between data points uses the area of the polygon a data point creates. During extensive experimentations back in 1984, it is shown that humans tend to misjudge relative areas much more than for example lengths [41]. This might influence the decision making process if radar charts are used.

7.1.3 Parallel Coordinates

Parallel coordinates uses a lot of the same principals as the radar chart. Instead of having the axes in a circle, the axes are now parallel to each other. The configurations are plotted as series of connected line segments. In Figure 11 on page 36 an example is given with the Iris data set, in this

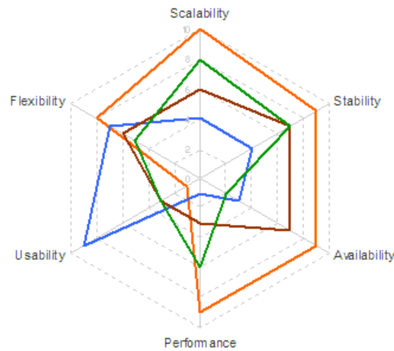

Figure 10: Example of a radar chart.

you can see relations between two objectives at the time. However this is influenced by the placement of the axes, you can swap them around and get a different picture. Only the relation with the neighbouring objective is visible. Because the placement of the objectives on the axes makes it hard to see the interaction between all objectives, in the same way a pairwise scatter plot does. Because of this, the method may not be the best for making a trade-off decision.

7.2 Reduce Dimensions

Besides trying to fit all dimensions into one graphical representation we can also try to reduce the number of dimensions. If we can successfully do this, we can have a simpler graphical representation, making it easier for the decision maker to understand the data.

We can show the ranges, minimum and maximum value, of the dimensions in the Pareto front. The decision maker could then give a smaller range for one of the output variables, in which the eventually chosen configuration must lay. This also indicates that for this smaller range the decision maker is indifferent to the exact value. This means the total decision set becomes smaller as some configurations do not fulfil the new range constraint. But more importantly we can remove the dimension from the presentation, as it does not matter any longer for the remaining configurations. This method should be used with care, as the full picture of the configuration is no longer shown to make the final decision.

7.3 Smart Presenting

Instead of presenting all configurations at the same time we can also use a procedure for showing the configurations. Thousands of configurations make us numb to the differences. But if we are presented with twenty

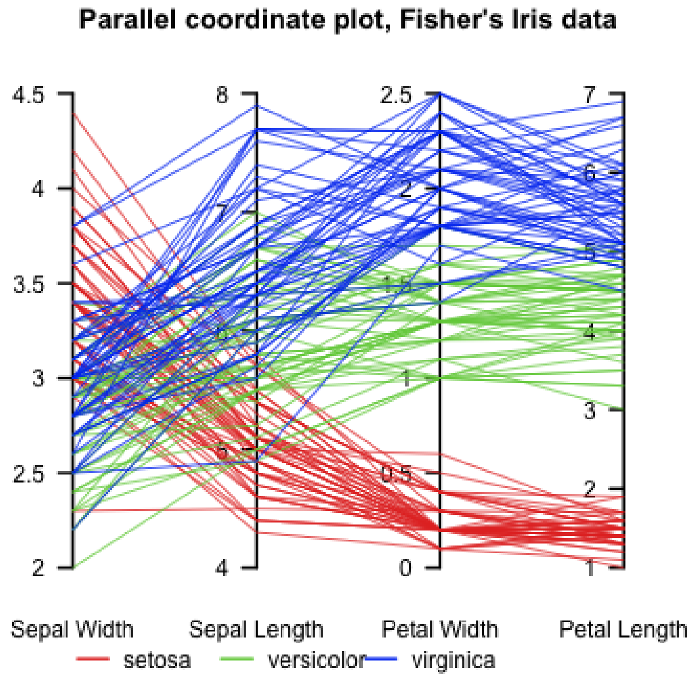

Figure 11: Parallel coordinate example chart (Iris data set).

configurations we are able to make distinctions and have a clear preference. It also benefits the presentation itself as less data needs to be represented we can use different methods easier, such as the radar chart. Using this we can present the information step-wise, creating an iterative/guided search. Each step we have a set of configurations, starting with the full Pareto front. From this set we pick a number of configurations that each represent a group of other configurations, how this is done will be explained shortly. The decision maker now selects the preferred configuration. This will then shrink the original set of configurations, by only including configurations that have comparable values. This process can than be repeated to eventually get to the most preferred configuration. The process is described step by step below.

1. Pick representative configurations from the given set.
2. Present these to the decision maker.
3. Decision maker picks the favourite of presented configurations.
4. Reduce the set by only using the configurations that are comparable with the chosen configuration.

5. Repeat from step 1 until only one configuration is left.

We now have to solve two parts: how to pick representative configurations and how to shrink the set of configurations. The latter is quite simple if we just shrink the set to the configurations that are represented in the first place. To get these representatives we can use clustering, by clustering similar performing configurations we can pick the mean of the cluster to represent that group of configurations. The clustering algorithm that really fits here is the k-means clustering, with this algorithm we can even specify the number of representatives (clusters) we end up with. On average each round of this approach you will divide the total set of configurations by the number of representatives, as the clusters tend to have similar size. However as it could bias the presentation, so having a comparison with the full Pareto front should be done.

8 Next Implementation

In this section we present the case study, the process to find the optimal infrastructure for AFAS Next. Next is an ERP application that will be SaaS solution, meaning the hosting of the software is done by AFAS, the service providing company. During this study we want to optimize three objectives, performance, robustness and costs. Performance is based on the time a user has to wait for the application to complete its tasks. This will be measured in the time a command takes to complete and how fast events are handled to make the data available to read. Query time is not a part of this objective as it solely depends on the database, which is not altered in our configuration. Robustness is based on how well the application can deal with disruptive behaviour, such as failing infrastructure. This is measured with the same metrics as the performance but with a different testing scenario. Costs are the operational costs of the hardware.

We go through specifying the configuration and the samples, the experiment set-up to evaluate the configurations with workload scenario. We analysed the output of the evaluations to reduce the output dimensions from the experiments to use in the optimization. With these samples and output variables we trained the five heuristics described in Section 6. The most accurate heuristic will be used in the optimization.

8.1 Configuration

In our experiment we do not use all of the parameters described in Section 3. The reason for this is the number of dimensions, if we make the configuration too big it becomes hard to tell which part of the configuration contributes to success. This has been researched by Bellman [15] and is part of the Curse of Dimensionality.

We use the virtual machine approach and have selected four different types in the Azure cloud. In Table 3 you can see the specifications of the virtual machine types. Secondly we will vary the size of the Service Fabric cluster, how many of these virtual machines there will be in a cluster.

VM Type	CPU (Cores)	Memory (GB)	Disk (IOPS)	Price p/h
A2	2 (2.1GHz)	3.5	4x500 (HDD)	€0.1518
A5	2 (2.1GHz)	14	4X500 (HDD)	€0.2867
D2V2	2 (2.4GHz)	7	4x1600 (SSD)	€0.2505
D3V2	4 (2.4GHz)	14	8x1600 (SSD)	€0.5001

Table 3: Specifications of the virtual machine types.

At the orchestrator level we have chosen to focus on the resource balancing. But even within the resource balancing there are multiple parameters

as described before, we start with the two most basic parameters. Balancing weights and balancing threshold, these parameters are indicated as the starting point to configure the resource balancer by the Microsoft Service Fabric team. To recap from Section 3, balancing weights determine how important a resource metric is for a service. Balancing thresholds determine when the resource balancer needs to re-balance the cluster, because the balance of a resource metric exceeds the threshold ratio. That is why we add the balancing weights and the balancing thresholds to the configuration.

For these parameters we see the built-in metrics of Count, Primary Count and Replica Count, as one. This only means the weights of the built-in metrics can not vary from each other, but the weight for them together can.

We define three resources that will be used in conjunction with the built-in metrics. These custom resource metrics are, unlike the static built-in metrics, dynamic, meaning they will change over time.

Actor Executions The number of executions done by actors in a specific time, for now this is mainly the handling of commands.

Event dispatch queue count The number of events that are waiting to be dispatched to the event handlers.

Events handled The number of events that are handled by an event handler during a period.

In total we define seven balancing weights, we have three distinct custom metrics, these and their relative defaults will have weights. This is done to see whether the original metrics or our custom metrics are more important to each service. As a final weight we add all the other microservices that do not fit with a custom metric yet, but still count towards the load in the default metrics. For the ratios, as it is defined once per metric, we use four thresholds, the three custom metrics and one for the default metrics.

In Table 4 on the following page all the input variables are displayed with their lower and upper bound and whether the value should be an integer. The virtual machine types are mapped to the numbers 0,1,2,3. These are a representation of an ordinal scale, meaning we can not apply normal arithmetic. The number of virtual machines per cluster has a lower bound of five, this is given by Service Fabric. They can not give their guarantees with fewer machines in the cluster. For instance if there are three machines in a cluster, one goes down for maintenance/upgrade only two machines remain on which the replicas need to be placed. This results in replicas being on the same machine which means there is no reliable quorum, meaning it is unable to accept new data safely. That is why the Service Fabric team advises a minimum of five machines for a production cluster. The balancing weights are, like the VM types, ordinal, so these are also easily mapped to zero through three. The balancing ratio thresholds are not necessarily integer as

ID	Setting	Lower	Upper	Integer
V1	VM Type	0	3	Yes
V2	Number of VM's	5	15	Yes
V3	Actor Executions Weight	0	3	Yes
V4	Actor Defaults Weight	0	3	Yes
V5	Events Handled Weight	0	3	Yes
V6	Event handler Defaults Weight	0	3	Yes
V7	Queue Count Weight	0	3	Yes
V8	Queue Defaults Weight	0	3	Yes
V9	Defaults Weight	0	3	Yes
V10	Actor Executions Threshold	1	5	No
V11	Events Handled Threshold	1	5	No
V12	Queue Count Threshold	1	5	No
V13	Defaults Threshold	1	5	No

Table 4: Specification of the characteristics of the possible configurations.

ratios are in definition rational numbers. They have a fixed lower bound of one, just because of the specification of a ratio. Maximum divided by minimum, minimum is always equal to or smaller than maximum so the ratio is always greater or equal to one. To get a sense of scale, if we use one decimal for the none integer parameters and the bounds we just specified there are 1845493760000 or $1.8 * 10^{12}$ possible configurations. To compare there are approximately $3.1 * 10^{10}$ seconds in a year, if we evaluate one configuration each millisecond it would still take 50 years to evaluate all configurations.

8.2 Sampling

For sampling the configuration space we use the spreadsheets from Sanchez [33], as described in Section 5 this gives us the sample points according to a nearly orthogonal Latin hypercube design. In Table 4 we specified 13 dimensions for Next, this means we need to use the design for up to 16 factors, as per instruction we only use the first 13 columns of the design. For each column the boundaries are set. The number of decimals for the discrete dimensions is set to zero, for the continuous dimensions this is set to two decimals. The mapping for integers to (Azure) virtual machine types is:

0. Standard A2
1. Standard A5
2. Standard D2 V2
3. Standard D3 V2

	Sample point 1	Mapped to Next	Sample point 2	Mapped to Next
V1	1	D2_V2	2	D3_V2
V2	5	5	12	12
V3	1	Low	0	Zero
V4	1	Low	1	Low
V5	0	Zero	1	Low
V6	2	Medium	1	Low
V7	2	Medium	2	Medium
V8	1	Low	2	Medium
V9	3	High	2	Medium
V10	3.88	3.88	4.69	4.69
V11	3.19	3.19	4.06	4.06
V12	4.75	4.75	2.94	2.94
V13	1.75	1.75	1.94	1.94

Table 5: Example sample points with the mapping to the Next configuration.

The balancing weights are Zero, Low, Medium and High, these are mapped from zero to three. The number of virtual machines and the ratio thresholds do not require a mapping, the data from these columns is used as is. As an example the first two samples from this design are depicted in Table 5, with the mapping to a Next configuration besides them. The full sample design consists of 65 nearly orthogonal space filling configurations. All the samples are shown in Appendix A.

8.3 Sampling Execution

We have 65 sample configurations which require evaluation. We need to test the performance and robustness of each configuration. We evaluate the configurations in the cloud of Microsoft, Azure. The cloud use the virtual machine approach and makes it possible for us to create any kind and size of cluster. Using the cloud relieves us from the necessity to set-up an physical infrastructure to create multiple different clusters. The choice for Azure was made because at the time it was the only cloud that ran Service Fabric reliably.

8.3.1 Testing Environment

Besides the virtual machines for the Service Fabric cluster we create three more machines, each dedicated to a task. In Figure 12 on the following page the environment used for testing is depicted. The database machine is a Standard A3 (Azure VM type) machine running PostgreSQL 9.5. This SQL server is storing the data of the application and is cleared between each test. Another machine (Standard A2) is configured for logging, it contains the full

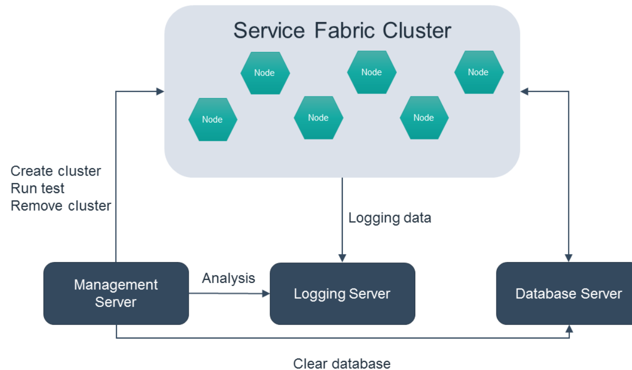

Figure 12: Overview of the test environment.

ELK-stack, Elastic Search, Logstash and Kibana. ELK is an open source software stack for collecting and analysing data[3]. This machine gets the data from all machines in the Service Fabric Cluster. This includes logging on commands, events, queries, warnings and errors. The management machine (Standard A2) is in control of all the experiments.

8.3.2 Deployment of the configuration

On this machine there is tooling to automatically create the resources in Azure, including installation of the required frameworks on the virtual machines. During this process the virtual machine type and number specified in the configuration is used. When Service Fabric is deployed on these machines the threshold ratio's are set from the configuration. The management machine now waits until the Service Fabric Cluster is operational. At that point first a logging service is deployed to make sure all of the test data is actually gathered and shipped to the logging machine. Now the Next application gets deployed, in this step the balancing weights for the microservices are being set.

8.3.3 Executing the Workload

When the application is correctly deployed a workload is played, following this scenario:

- Seeding data (set-up phase)
- Wait for events to complete
- Run insert commands

[3] www.elastic.co

- Wait for events to complete
- Run update commands
- Wait for events to complete

In the set-up phase the default data is inserted, this consists of data which is required by other commands and data that will be referenced in the update commands. The waiting periods make sure the collected measurements are not leaking into other phases of the test. During the first test scenario we only send insert commands, meaning new records are created in the system. This scenario is time bound by three minutes, in this time between 10.000 and 20.000 commands are processed by the system. This results roughly in tenfold events, these are handled in 15-25 minutes. In the second test scenario only update commands are used, meaning records are updated but not created. It is again time bound by three minutes. The records that are updated are inserted in the set-up phase, so we know for sure they are available for updating. The number of commands and events is in the same range as with the insert commands. The event handling time is also in the same time range as with the insert scenario. After this the test tooling aggregates the data on the logging machine and stores these results on the logging machine. The application is now removed to clear all state. This concludes the default load test.

8.3.4 Robustness Stress Test

Following the initial load test a stress test will be executed, to test how the system reacts to failures. This is done by deploying a chaos service on the cluster. This will keep inducing failures in the cluster. A couple of actions this service will do are; resetting a code package on one machine, moving partitions between machines, kill partitions and restart machines. Each iteration of failures contains one to five failures with a two second delay between each failure. After each iteration the chaos service waits until the application has recovered, when it has recovered there is a five second delay before the next iteration starts. The failures that occur are mostly low impacting such as moving partitions, but in one percent of the cases it will restart an entire machine which has a high impact on the performance. While this service is active the application will be deployed once more and the workload 8.3.3 will be executed again.

Once this is all done the resources are removed from Azure and the next configuration can be tested. To get an accurate view we should run each test multiple times to account for variability. These tests have a lot of variability due to the distributed nature of the application, the placement of the services, and the chaos service. However this is just a proof of concept on how to execute the process and not decision making material. We will

keep it at one run for each test, to keep the time more manageable and the experiments costs lower.

8.4 Analysing Samples

The tests give us a lot of measurement data. The metrics we use are events per second and the duration of a command as this determines our performance. The measurements are split in two ways, default or stress test and insert or update commands. This is based on the scenarios in the tests, to look at the performance with and without failures but also the type of operations that are performed. The stress test results are our measurements for the robustness objective. The inserts create new data while the update commands only change the data. These different types of commands might have a different performance. This means we end up with eight different output variables for each test next to the costs of the configuration. These eight are labelled as defined below.

O1 Events per second - Default - Inserts

O2 Events per second - Default - Updates

O3 Events per second - Stress - Inserts

O4 Events per second - Stress - Updates

O5 Command duration - Default - Inserts

O6 Command duration - Default - Updates

O7 Command duration - Stress - Inserts

O8 Command duration - Stress - Updates

These eight output variables and the cost of the configuration result in nine output values for each of the 65 samples. If we use all nine output variables, it would be too much to start the optimization. This follows again from the Curse of Dimensionality with so many objectives it becomes very hard to make the trade-off. Furthermore with each dimension the number of configurations in the Pareto front will increase significantly, which will make the decision only harder. This is why we need to reduce the dimensionality of the data. We can reduce the number of dimensions by removing dimensions without extra information, if a dimension does not contain new information it is not helping the optimization. To do this we need to analyse the data. The first step is looking at our data and see if there are any patterns, this is done by making pairwise scatter plots for each of the output variables, you can see this in Figure 13 on the next page. In each sub plot two output variables are plotted against each other containing the 65 data points. In

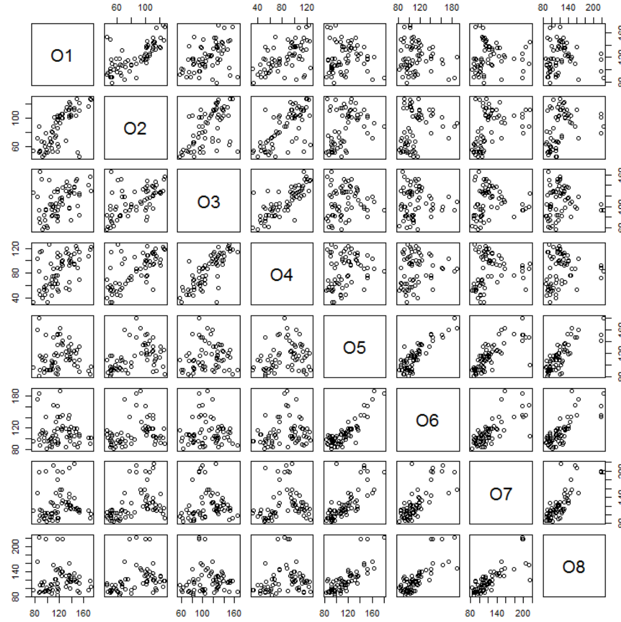

Figure 13: Pairwise scatter plot of the measurements.

the top-left and bottom-right you can see there are indications of a linear correlation.

However we can not make a decision based on those plots, it gives us an indication but now we need to confirm it with mathematics. This can be done with the sample correlation coefficients. This is estimating the population Pearson correlation [42], which is the correlation when all the data is available. As we only have samples of the entire configuration space and not the output data of every possible configuration, we can only estimate the correlation coefficients. All coefficients are between -1 and 1, where -1 is a perfect negative correlation. 1 is a perfect positive correlation and 0 means there is no correlation at all. From this follows that we are looking for numbers close to -1 or 1, so we can combine these variables and reduce the number of dimensions that way. To calculate the coefficient between output variables x and y we use the following formula.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The results of the correlation coefficients in Table 6 are confirming the graphical indication. The values in the top-left and bottom-right of the table are higher than the other quadrants. However we need to be sure the values are significant enough to actually use. To test whether a value is significantly different from 0 we can calculate the confidence of the coefficient. This is

	O1	O2	O3	O4	O5	O6	O7	O8
O1	1.000	0.714	0.431	0.544	-0.015	-0.118	0.058	-0.019
O2	0.714	1.000	0.574	0.678	0.395	0.272	0.417	0.356
O3	0.431	0.574	1.000	0.799	0.080	-0.026	0.040	0.005
O4	0.544	0.678	0.799	1.000	0.238	0.167	0.155	0.168
O5	-0.015	0.395	0.080	0.238	1.000	0.871	0.752	0.802
O6	-0.118	0.272	-0.026	0.167	0.871	1.000	0.743	0.786
O7	0.058	0.417	0.040	0.155	0.752	0.743	1.000	0.824
O8	-0.019	0.356	0.005	0.168	0.802	0.786	0.824	1.000

Table 6: Sample correlation coefficients.

relative to the number of samples n and the value of the coefficient r_{xy}. It is based on the Cumulative Probability Density function (CPD) of the t-distribution. The full formula is stated below. We are more confident of the correlation if this P_{xy} is closer to 0. The results of this calculation can be found in Table 7.

$$P_{xy} = 2 * \left(1 - \text{CPD}\left(|r_{xy}|\sqrt{\frac{n-2}{1-r_{xy}^2}}, \text{DF}{=}n-2\right)\right)$$

	O1	O2	O3	O4	O5	O6	O7	O8
O1		2.36e-11	3.42e-04	2.78e-06	0.9034	0.3498	0.6439	0.8818
O2	2.36e-11		5.79e-07	5.38e-10	0.0011	0.0284	0.0006	0.0036
O3	3.42e-04	5.79e-07		1.33e-15	0.5250	0.8371	0.7490	0.9711
O4	2.78e-06	5.38e-10	1.33e-15		0.0559	0.1838	0.2188	0.1815
O5	0.9034	0.0011	0.5250	0.0559		0	5.45e-13	8.88e-16
O6	0.3498	0.0284	0.8371	0.1838	0		1.35e-12	9.33e-15
O7	0.6439	0.0006	0.7490	0.2188	5.45e-13	1.35e-12		0
O8	0.8818	0.0036	0.9711	0.1815	8.88e-16	9.33e-15	0	

Table 7: P-values: Confidence of the correlation.

From the confidence values we can conclude that the bottom-right quadrant is correlated. So we can simplify O5-O8 to reduce the dimensionality of the problem. The range of O1-O4 has a difference between the default test (O1,O2) and the stress test (O3,O4). But this still means we can reduce the dimensionality. O1 and O2 will be combined into O9, O3 and O4 create O10 and O5-08 will make O11. To combine dimensions we will use the average, note that if the variables would have a different order of magnitude you need to normalize it in some way before combining.

O9 Events per second - Default - Combined

O10 Events per second - Stress - Combined

O11 Command duration - Combined

To test whether the new response variables actually capture the originals we perform the correlation and confidence calculations again. The results of those calculations are shown in Tables 8 and 9. We can see that all intended correlations are higher than 0.9, and the confidence is rounded to zero because it becomes too small. It once more confirms there is a difference between the default and stress test for the event handling.

	O9	O10		O11
O1	0.917	0.513	O5	0.914
O2	0.934	0.659	O6	0.909
O3	0.547	0.950	O7	0.919
O4	0.664	0.946	O8	0.935

Table 8: Correlation values of the combined output variables.

	O9	O10		O11
O1	0	1.257461e-05	O5	0
O2	0	2.392023e-09	O6	0
O3	2.449197e-06	0	O7	0
O4	1.627278e-09	0	O8	0

Table 9: P values of the combined output variables.

8.5 Heuristics

We try all the heuristics specified in Section 6, because there is not a universal heuristic that can be used for every problem. We need to make an informed decision, which of the heuristics is best for the objectives. All heuristics were tested with existing packages in the programming language R. R is developed as one of the frontrunners in statistical and predictive algorithms and has numerous packages that facilitate these operations.

To train the heuristics we use the samples generated via the approach explained above. To counter over fitting we require more samples that will not be used in the training. This is why five more sample tests were done, these samples were taken within the same boundaries as the training data but were randomly constructed. These configurations can be found in Appendix A. These were tested in the same way as the rest of the samples so the environment was equal except for the dimensions in the configuration. The measured output variables are found in Appendix B.

The root mean squared deviation of each combination of heuristic and output variable is calculated. The root mean squared deviation was explained in Section 6, it determines the deviation from reality for the approximation. The results can be seen in Table 10 on the next page, as we

	O9	O10	O11
Linear Model	26.4569	35.3014	27.5312
Non-linear	31.6856	41.2493	48.7933
Kriging	26.4569	35.3072	27.5106
Neural Network (RBF)	27.7464	32.2345	25.7079
Support Vector Regression	**25.6650**	**27.5827**	**22.6611**

Table 10: Accuracy approximation algorithms, root mean squared deviation of the validation samples.

are looking at deviations from reality a lower score means the heuristic represents reality better. The table shows that for each response variable the best heuristic is the support vector regression. The results also show that the heuristics do not need to have the same ordering for each variable. For example, the neural network is worse than Kriging and the linear model in the first column but better in the other two. Linear and Kriging are having near identical accuracy in each of the objectives. This table shows us that not every variable behaves like each other, making it important to try out several heuristics before picking one. There exists no ultimate heuristic that fits every problem. In this case, support vector regression is superior for all the output variables. So this will be the heuristic used in the evaluation step of the genetic algorithm to optimize the configurations.

9 Results

We have created a configuration for hardware and orchestrator with 13 parameters. The focus of these parameters was resource balancing to get better performance and robustness for the application. To create the Pareto front we have used the non-dominating sorting genetic algorithm, but this required a fast evaluation of the configurations. We were unable to use simulation to solve this problem, so we used heuristics. Training these heuristics required training data, to make sure the training data covers the possibilities evenly and with few samples we have chosen a sampling design. The nearly orthogonal Latin hypercube sampling design is created to fulfil these properties, following this design we got 65 sample points.

The sample points were evaluated with a load test to obtain the measurements of the output variables. With this training data set the heuristics were trained, five additional samples were taken and used as a validation set. During validation the support vector regression showed the best accuracy and is therefore used in the genetic algorithm.

The non-dominated sorting genetic algorithm was run. As parameters the population size was set to forty. The mutation chance was set to one mutation per generation of new solutions (children), so one in forty. As a stopping criterion the number of generations without improvement of the archive was used, this number was set to 300. The genetic algorithm ran for nearly two hours and iterated through 44861 generations. Each generation between ten and twenty configurations were on the first dominance rank within the population and therefore compared with the archive. In total 5993 configurations were part of the archive during the optimization, but as new configurations are added old configurations get dominated and need to be removed. After the algorithm completed there were 2873 configurations left in the archive. In the rest of this section we will present the results of this Pareto front.

Showing all Pareto optimal configurations in a table takes too much space and will not give a lot of information. So instead this section will present the data in other representations. A benefit of having all Pareto optimal configurations is the possibility to look at commonalities between them or how factors influence each other. We can analyse both sides of the configurations in the Pareto front, the thirteen dimensions that make up the input of a configuration and the side which is more important for decision making, the four output variables of the configurations.

9.1 Configuration Analysis

One thing all configurations have in common is they are all Pareto optimal, which means they excel in either one objective or a combination of objectives. It is interesting to know whether there exist commonalities be-

tween these configurations. If so, we can try to explain it and try to change other settings in the configuration that were not part of this test with this knowledge in mind.

Cluster size(V2)\VM type (V1)	A2	A5	D2V2	D3V2	Total
5	35	13	31	47	126
6	27	3	4	3	37
7	25	0	7	0	32
8	42	0	20	17	79
9	111	8	59	98	276
10	79	9	16	166	270
11	134	36	70	185	425
12	205	82	70	193	550
13	137	48	48	199	432
14	176	99	56	101	432
15	81	37	38	58	214
Total	1052	335	419	1067	2873

Table 11: Occurrences of cluster size and virtual machine type combinations in the Pareto front.

9.1.1 Hardware (V1, V2)

In Table 11, we take a look at the occurrences of cluster size and virtual machine type combinations. This is the hardware subset of the configuration and solely determines the output variable of costs. The first observation is that the A2 and D3V2, cheapest and most expensive in the test, both have more than thousand occurrences, while the other two combined make up less than thousand configurations. The cheap A2 machines can be explained as one of the objectives is cost of the cluster. A possible explanation for the large number of D3V2 machines could be, that they have the most resources at their disposal. With the most resources available it becomes easier to get better performing configurations. The reason that the two other types, A5 and D2V2, are in the Pareto front is because they are in between price wise but also performance, they bridge the trade-off gap between the cheap and expensive machines. The second observation is based on the number of machines in the cluster. To get a better idea of the distribution of the sizes we can take a look at Figure 14 on the next page. What we see is, a bell curve or Gaussian distribution with a peak at twelve machines. Clusters with a size from six to eight are under represented in the Pareto front. However there is a remarkable peak at five machines. For the A2 machines this could be explained as being the cheapest of all configurations, but this trend also shows with the other three types of machines. An explanation for the curve at a higher number of machines could be, firstly there are more resources

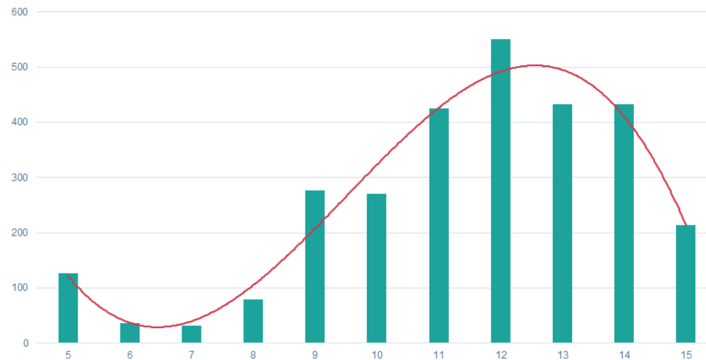

Figure 14: Number of virtual machines in a cluster (V2).

ID	Balancing weight	Zero	Low	Medium	High
V3	Actor executions	3	141	2719	10
V4	Actor defaults	0	747	1221	905
V5	Events handled	93	909	904	967
V6	Event handler defaults	1	872	1996	4
V7	Dispatch queue count	1	901	1967	4
V8	Dispatcher defaults	2	49	2793	29
V9	Defaults for remaining services	4	1823	1042	4

Table 12: Occurrences of balancing weights input variable.

available. Secondly if one machine has trouble, it affects less services at the same time as there are more machines to spread the services over. It also has a less likely chance to become imbalanced, there is more spread so the chance that one node becomes really overburdened declines. The downward slope towards 15 machine clusters means the extra machine does not add to the performance significantly, thus these configurations get overshadowed by cheaper alternatives.

9.1.2 Balancing Weights (V3-V9)

Now we look towards the settings of the orchestrator. First we take a look at the balancing weights, there are seven dimensions in the configuration using these weights. The weights are divided in four categories, Zero, Low, Medium and High. For the seven dimensions the number of occurrences is counted in the Pareto front, the results are shown in Table 12. In this table we see a large tendency towards the medium setting, with low as a great runner up. With the exception of events handled the zero weight does not see much use. Only two dimensions make use of the high setting in the Pareto front.

What might be more interesting is the interaction between the weights.

Micro-service	Specific	Default	Equal
Actors (V3&V4)	651	943	1279
Event handler (V5&V6)	1472	845	556
Event dispatcher (V7&V8)	35	905	1933

Table 13: Relation between the default and the specific balancing weight in a microservice. How often is the specific metric more important than the default.

For three services we have the specific metric and the default metrics, actors, event handler and event dispatcher. Table 13 shows how the weights are relative to each other in the Pareto optimal configurations. For the event dispatcher the specific metric is nearly always equal or lower rated than the default metrics. For actors and the event handler there is no clear distinction which should be weighted heavier.

9.1.3 Balancing Thresholds (V10-V13)

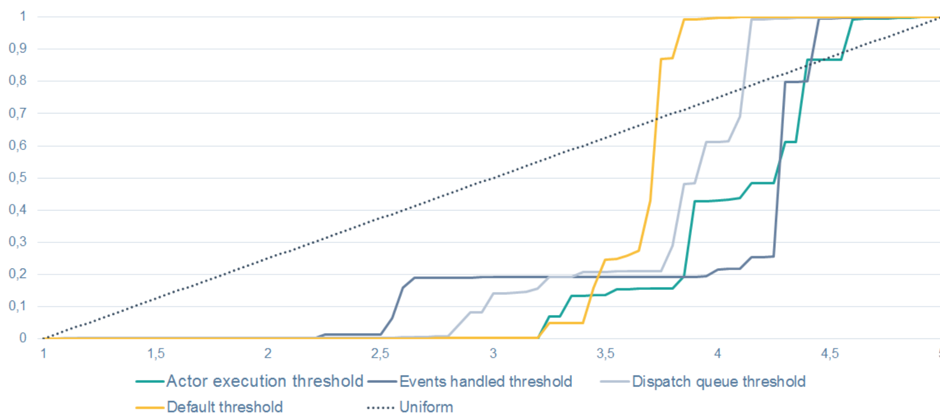

Figure 15: Cumulative of the threshold values found in the Pareto optimal configurations (V10-V13).

When looking at the balancing thresholds we notice that in the Pareto front the thresholds become high values. All the configurations are shown as a cumulative graph in Figure 15. In this graph, a base line is drawn for a uniform distribution. If all values would be used evenly between the boundaries the line should be close to this uniform line. However what we see is that all thresholds stay on zero percent until the value of the threshold exceed two or even three in some cases. After a threshold of 3.5 the cumulative really takes off, around eighty percent of the values are bigger than 3.5.

This favour for higher thresholds can be explained. The threshold deter-

mines when Service Fabric finds the cluster unbalanced. So if the value of the threshold is low, Service Fabric calls the cluster unbalanced faster. This results in more balancing, which in turn results in more microservices getting moved. During a move the service might temporarily respond slower, giving a lower fitness in the objective.

9.1.4 Conclusions

From the analysis of the configurations in the Pareto front we made some observations. The virtual machine type and number of virtual machines per cluster contain the most variability in the Pareto front. These variables have the most impact on the differences in the objectives, in the case of costs they solely determine the objective. The balancing weights do not have a preferred priority between the resource metrics. They favour the average values of Low and Medium. In a following experiment we could zoom in on these variables by locking the other variables down on a constant value. All of the balancing thresholds ended up with a high threshold, from this we can conclude that we always want a higher threshold value to make sure that Service Fabric does not start balancing for each tilt in resource usage.

9.2 Objectives Analysis

Now we have seen what makes up the input of the Pareto optimal configurations, it is time to look at the output. First we look at objectives on their own and we end with a full picture of the Pareto front.

9.2.1 Cost

First up is the cost of a cluster, this objective did not need an heuristic. The calculation was simply, the price of the machine type times the number of machines. The cheapest option was a five machine cluster of type A2, which costs €565 per month. The most expensive set up was fifteen D2V2 machines, is nearly tenfold of the cheapest, coming down at €5580 per month. In Figure 16 on the next page, we can see the range between these extreme points is covered quite evenly. It is a cumulative graph of the costs found in the Pareto front with the uniform distribution as reference. The large number of occurrences between €1000 and €1500 are the large A2 clusters. Because an A2 machine is relatively cheap all large clusters fit into just that range, the expensive D2V2 clusters are spread over a wider range and do not show one clear bump. From a decision making stand point this spread of choices is what we would like, there are no gaps or under represented price ranges.

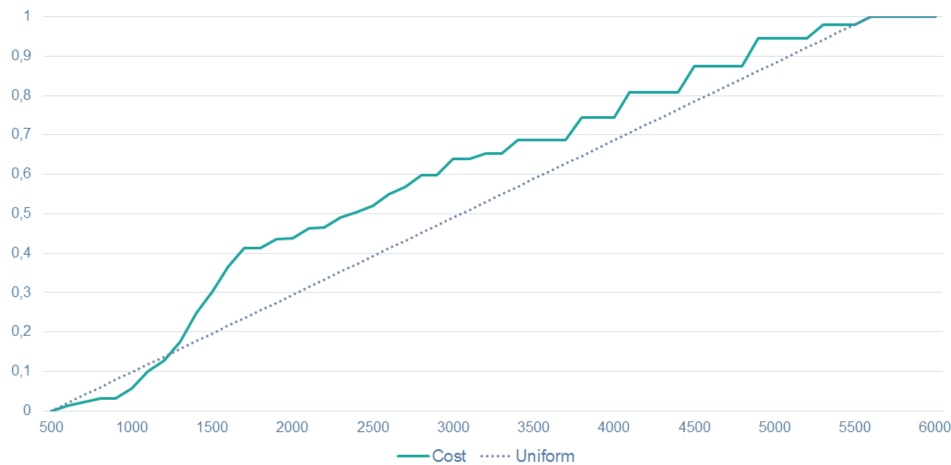

Figure 16: Cumulative of configuration price occurrences found in the Pareto front.

9.2.2 Performance and Robustness

The command durations (O11) in the Pareto front are between 75ms and 141ms. These can not be related in the same way to the input as the price of a cluster. For this objective a cumulative graph is also made, which can be seen in Figure 17 on the following page. This graph is nearly linear, meaning we have a near perfect spread between the minimum and maximum value of this objective. This means the Pareto front offers choice all across the range.

The events per second objectives, default (O9) and under stress (O10), are both between 75 per second and 144 per second and follow the same kind of distribution in the Pareto front. Just as the command duration this can not be related to the input in an explainable way, it comes from the heuristic. The range is also nearly the same as the command duration, so the cumulative of the events per second are plotted in the same graph, Figure 17. There are more configurations that have higher events per second, which is nice as we want to maximize the number of events handled. However there are no gaps or under representation, so the trade-off can be made here as well. One thing that stands out is the relation between the two objectives. In the graph the line of the stress scenario is below that of the default scenario, which means that there are less low values in the stress scenario. This implies that for a lot of configurations the stress scenario is performing better than the default scenario. This might be explained by two factors, the inaccuracy of the samples, the tests should be executed multiple times to make sure no artefacts end up in the results. Secondly extrapolation can be a cause for this phenomenon, the heuristic for the default scenario and the heuristic for the stress scenario can be formed in such a way that the

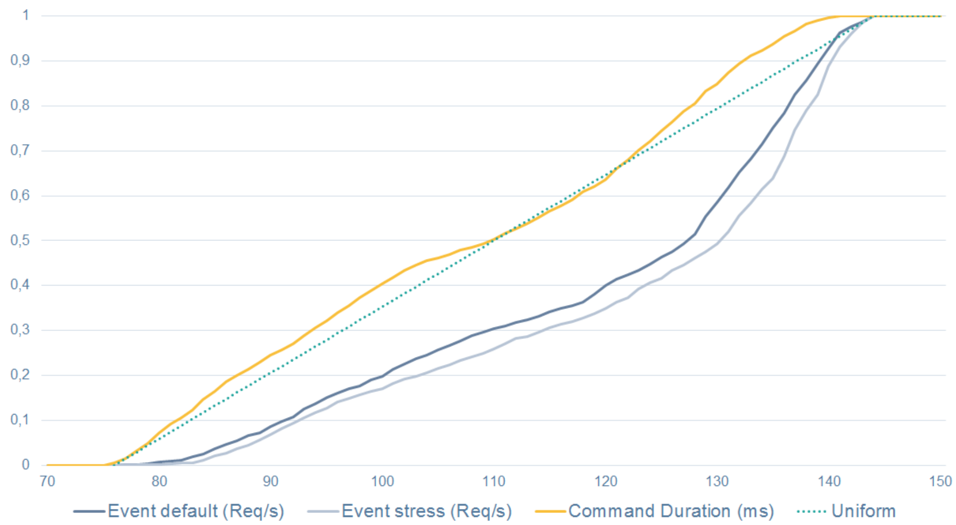

Figure 17: Cumulative of the threshold values found in the Pareto optimal configurations.

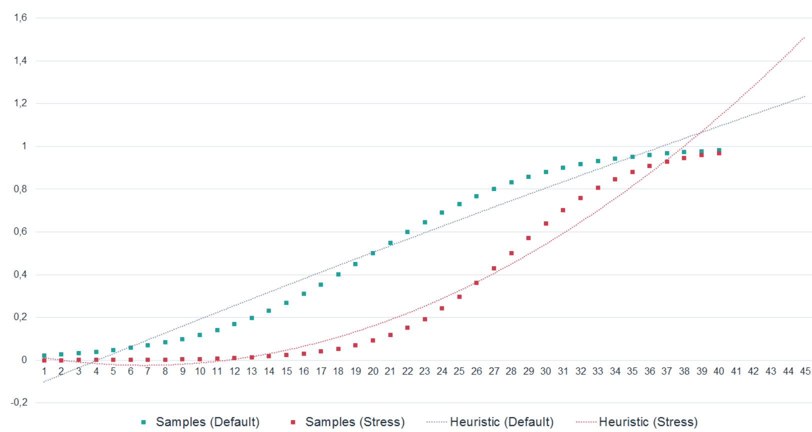

Figure 18: Example of extrapolation error in heuristics (Not real data).

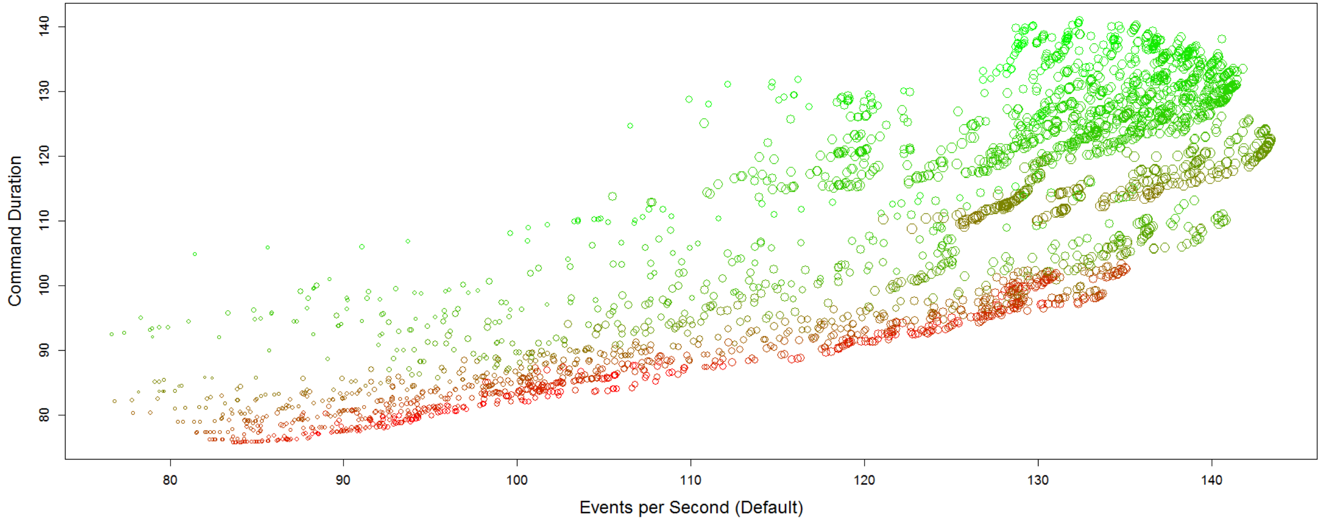

Figure 19: Full Pareto front of the Next case.

stress will overtake in performance. This would mean that according to the heuristics a configuration performs better while failures are occuring than when the system is stable. To illustrate this effect a simplified example, not results from the case, can be seen in Figure 18 on the previous page. In this graph the stress samples have lower scores than the default samples, as we are looking at the events we want to maximize so the stress is doing worse in the samples. Due to the way the samples lay the heuristics of the default and stress scenarios are different. In the range of the samples the relative order between default and stress is preserved but when looking at either tail ends the stress is doing better. This is because the default has a gradual ascend over the entire span of the samples, while the stress has a steep ascend in the samples resulting in a steeper heuristic. This could also have happened to the stress data from our study.

9.2.3 Pareto front

After seeing each objective independently, we take a look at the interaction between all of them in the approximate Pareto front. In Figure 19 the front is depicted in a scatter plot, the x-axis represents events per second in the default scenario, the y-axis shows the duration of a command. The size of the circles represents the events per second in the stress scenario where bigger equals more events. The colour scale represents the costs of infrastructure, green is cheap and red is expensive. To give a sense of direction, the optimization was going towards a large green circle in the bottom right

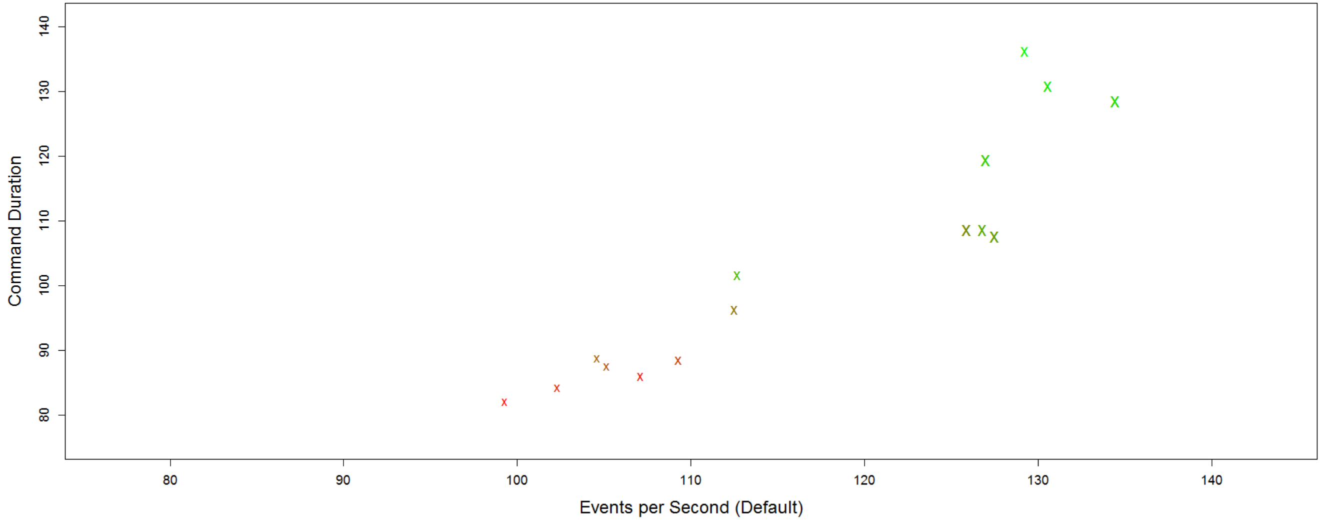

Figure 20: K-means clustered Pareto front.

as this would be the optimal configuration.

We observe a linear laying group which excels in the combination of low command duration and a high number of events per second, bottom layer in the graph, consisting of red points i.e. expensive configurations. These points have the best performance in the default scenario but are the most expensive in costs. This phenomenon is expected, costs and performance are the objectives which work against each other. Behind this red group there are similar parallel groups, which gradually lie higher in the chart. These parallel groups become cheaper when the configurations become worse in the command duration objective.

Another observation can be made about the cheaper solutions, the green circles. They occupy a large cloud to the top right. In these configurations the event throughput is high in both the default and the stress scenario, but they are slow on the command side. It appears that the cheap configurations are not able to compete for dominance with low command durations.

To make the trade-off easier, we can simplify the Pareto front. This is done by clustering the Pareto front and only presenting representative points from each cluster. We use k-means clustering, which is an iterative clustering algorithm. Each iteration every point is attached to the closest of the k centroids, then the mean value of each cluster is calculated which becomes the centroid of the next iteration. This stabilizes after a couple of iterations, at that point we can use the centroids of the last iteration as our representative points. However the starting centroids, which are randomly created, influence the final clustering. That is why the algorithm is executed

multiple times and the clustering with the least distance between the points and their representative is chosen. The clustering of our case can be seen in Figure 20 on the previous page, the trade-off between the objectives becomes more apparent. After picking one of the clusters this process can be repeated with the points from that cluster.

9.2.4 Conclusions

We have seen that for each independent objective there is a near uniform spread of configurations. There are no apparent gaps in the independent objectives, between the minimum value and maximum value there is plenty of choice. This translates to the full Pareto front, there are no big gaps in the front of combined objectives. The Pareto front that is created by our approach, gives the decision maker a set of configurations to make an informed trade-off. As the Pareto front is a set of configurations that are mathematically optimal we advise the company to use any of the given candidates, but are impartial to the choice of any configuration within the Pareto front.

10 Discussion

The problem we faced is configuring the infrastructure for a new SaaS solution. The client base will expand rapidly and that is why the infrastructure must be good from the start. It is not possible to define "good" as this is a opinion, therefore we look to present Pareto optimal configurations as trade-off candidates. This enables us to optimize the configurations without having knowledge upfront about how the configuration would influence the objectives.

We have created and executed an approach to find the trade-off candidates for the configuration of hardware and orchestrator in the case of new cloud software. The trade-off is based on the costs, performance and robustness of the configuration. Finding these candidates is done by a genetic algorithm that takes the mathematical property of domination in consideration while searching. To enable the use of this genetic algorithm we had to reduce the evaluation time of a configuration. Heuristics are the solution to reduce the evaluation time, by approximating reality with a function we can quickly evaluate each configuration. To train the heuristics real world examples are required, since these cost time and money this number should be kept to a minimum. Near orthogonal Latin hypercube sampling gives us the possibility to get an unbiased evenly spread sampling strategy, while using as few as possible samples.

The entire process makes it possible to search for trade-off candidates, which we show by using the approach at the case company AFAS. We successfully create a Pareto front within limited time for the new Software as a Service solution called AFAS Next. This approach that is used here for finding trade-off configurations for a new SaaS solution could also be used in different fields. For example designs that cannot be replaced or adjusted easily such as bridges (material type, thickness, type of suspension, etc.), satellites (material type, size, space debris shielding, etc.) or the design of a Formula 1 car spoiler (thickness, angle, width, etc.). Each of these problems can have multiple objectives for example the spoiler is responsible for the downward force of the car, making sure the wheels can get grip on the track but at the same time the spoiler should not add too much weight and the car needs to stay aerodynamic so the speed does not get restrained by the air friction. In these cases the evaluation the product is expensive, building multiple configurations may be impossible an example would be a bridge. The computer simulations are not the complete answer as they can take multiple hours. It could be possible to apply the same approach of sampling configurations, training heuristics and generating a Pareto front to reduce the costs of finding a fitting design.

10.1 Limitations

Problems might be complex, for example having additional constraints on the input of a configuration such as in the case of a bridge it might have as input the number of lanes and the width of a lane however the total width of the bridge can be limited. The non-dominated sorting genetic algorithm as most evolutionary algorithms has no built-in way of dealing with extra constraints on a configuration. It is only capable of lower and upper bounds as well as whether values need to be integer. This was not a factor with the configuration used for Next, but could limit other applications of this approach.

Another limitation of NSGA is that it is not possible to use preconceived knowledge of the problem to steer the search process. It will use, vary and mutate each of the input dimensions. There is no way to guide the search, by for example only changing a part of the configurations in the first half of the algorithm and when they have converged on an optimal value for those settings, the rest of the settings would be used in the search as well.

The experiment for Next could use some improvements. Firstly the workload that was used does not represent a realistic scenario. Not all areas of the application are hit during the test. The second part of the workload that might have influenced the results, during the stages we used a time based stage of sending commands instead of a fixed number of commands. This causes that more events need to be handled if the commands were handled quickly as this would make more commands possible in the same time period. The variable number of events might cause differences in the measurement results of the events. The use of one application is another simplification, in reality we want to host multiple applications/clients on one cluster. That way you can share the resources and keep the costs lower per client.

In the execution of the sampling we should have accounted for the stochastic nature of the process. There can be differences between two very similar configurations because the resource balancer decided to start moving critical processes in a certain time frame. To battle this each sample should be run multiple times, then taking the average of all the runs for a sample. This would reduce the chance of outliers that are used in the training of the heuristic. We chose not to do this in the current research because of time limitations. This should also be done for the validation set as they require the same treatment as the training set. The validation set could also be enlarged we currently used five samples in the validation set. We used 13 input dimensions, so five samples might not validate for the entire range of possibilities.

10.2 Future Work

The unstructured search method of non-dominated sorting genetic algorithm is a possible point to improve upon. When structure is introduced it could improve the results of the Pareto front or decrease the time required to find the results. The current search pattern uses crossover and mutation on the solutions in the population, applying this on every value in the configuration each iteration. The initial configurations are randomized in each dimension. It would be interesting to adjust the initial configurations in such a way that only a couple of the dimensions get randomized while the other will get a fixed value. Secondly the structure of the search could be changed so that it is possible to specify which dimension gets altered in an iteration, starting with a couple of dimensions and after X generations also start varying the other dimensions. This can also be done with conditional rules, for example when the progress of the Pareto front stagnates another dimension could be unlocked to start improving the Pareto front even further.

As future work for AFAS, there should be a follow up experiment. When the application gets near production-ready, the findings become more accurate and relevant. Besides improving the experiment by solving the limitations stated above, it is also possible to change the dimensions of the configuration. As we have seen in Section 9 the balancing threshold has a clear preference of a high value in all of the configurations in the Pareto front, which means it is not interesting to search those parameters again in a follow up experiment. By making the thresholds a constant, the number of dimensions is reduced and the remaining dimensions get a closer inspection. We expect with fewer dimensions that the running time will improve, as there is less variability in the configuration.

11 Conclusion

In this research we described an approach to help finding a fitting infrastructure configuration for hosting new cloud-based software. We use the definition of Pareto optimal to find configurations that are mathematically not worse than the other configurations, which enables the decision maker to choose while being fully informed of the possibilities. This approach with the steps of creating a configuration schema, sampling, heuristics, and finding the Pareto optimal configurations can be used for finding a fitting infrastructure for hosting a SaaS solution, but it can also be used in design other products as the notion of expensive evaluation is not limited to software deployment.

In our case a configuration is made up of 13 dimensions with a focus on performance on the command side and the information transfer to the query side by the means of events. By using the sampling design of near orthogonal Latin hypercube it is possible to evaluate a low number of samples that are evenly spread through out all possible configurations. These samples are used to train the heuristics, which make it possible to approximate the performance and robustness of a configuration of the infrastructure. With heuristics we have seen that the accuracy can differ and there does not exist an universal heuristic that fits all problems. In our case support vector regression approximated reality the most accurate. The heuristics enable the use of a genetic algorithm in reasonable time. The non-dominated sorting genetic algorithm uses the principle of dominance to search through the configuration space, this keeps the essence of a Pareto front in the search process and results in a well rounded Pareto front. By keeping an archive we made sure not to lose any of the configurations that are on the Pareto front. After creating the Pareto front we are able to gain insights in the behaviour of the various dimensions in the configuration and present a trade-off to the decision maker.

References

[1] B. Warfield. Minimizing the Cost of Saas Operation. https://www.enterpriseirregulars.com/15599/minimizing-the-cost-of-saas-operations/, 2010.

[2] D. Key. The Imperative to Reduce the Cost of SaaS Service. http://cloudstrategies.biz/imperative-reduce-cost-saas-service/, 2015.

[3] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. *Proceedings - International Symposium on Computer Architecture*, pages 315–326, 2008.

[4] M. Armbrust, A. Fox, R. Griffith, and A. D. Joseph. Above the clouds: A Berkeley view of cloud computing. *University of California, Berkeley, Tech. Rep. UCB*, pages 07–013, 2009.

[5] S. Chahal and K. Mailman. Sizing Server Platforms To Meet ERP Requirements. *IT@Intel White Paper*, (March), 2010.

[6] M. Lopez. World of Waitcraft. Get your queue on. http://www.gamespy.com/articles/581/581056p1.html, 2005.

[7] K. Orland. SimCity launch plagued by server problems. http://arstechnica.com/gaming/2013/03/clogged-streets-simcity-launch-plagued-by-server-problems/, 2013.

[8] A. Frank. Pokemon Go's server issues have been driving people wild all day. http://www.polygon.com/2016/7/7/12123750/pokemon-go-server-issues-ios-android-fix, 2016.

[9] M. Ehrgott. *Multicriteria Optimization*. Springer Science and Business Media, 2004.

[10] T. Goel, R. Vaidyanathan, R. T. Haftka, W. Shyy, N. V. Queipo, and K. Tucker. Response surface approximation of Pareto optimal front in multi-objective optimization. *Computer Methods in Applied Mechanics and Engineering*, 196(4-6):879–893, 2007.

[11] H. R. Loghmani and A. Ghoddosian. The use of the Corner Sorting method in the NSGA-II in comparison with SPEA-II in the simultaneous optimization , the size , Shape and topology of two-dimensional trusses. 13:321–335, 2014.

[12] U. Dahan. Clarified CQRS. http://www.udidahan.com/2009/12/0, 2009.

[13] M. Snider. Balancing your Service Fabric cluster. https://azure.microsoft.com/en-us/documentation/articles/service-fabric-cluster-resource-manager-balancing/, 2016.

[14] M. Snider. Describing a Service Fabric cluster. https://azure.microsoft.com/en-us/documentation/articles/service-fabric-cluster-resource-manager-cluster-description/, 2016.

[15] R. E. Bellman. *Dynamic Programming*. Dover Books on Computer Science Series. Dover Publications, 2003.

[16] V. Turecek. Measuring physical resources on partition level in Service Fabric. http://stackoverflow.com/questions/37112687/how-to-measure-resource-usage-on-partitionlevel-in-service-fabric, 2016.

[17] M. Snider. Managing resource consumption and load in Service Fabric with metrics. https://azure.microsoft.com/en-us/documentation/articles/service-fabric-cluster-resource-manager-metrics/, 2016.

[18] M. Ehrgott. Vilfredo Pareto and Multi-objective Optimization. *Documenta Mathematica*, I(Extra Volume: Optimization Stories):447–453, 2012.

[19] Y. Y. Haimes. Integrated system identification and optimization. *Control and dynamic systems: Advances in theory and applications*, 10:435–518, 1973.

[20] N. Veerapen, G. Ochoa, M. Harman, and E. K. Burke. An Integer Linear Programming approach to the single and bi-objective Next Release Problem. *Information and Software Technology*, 65:1–13, 2015.

[21] W. F. Bialas and M. H. Karwan. Two-level linear programming. *Management science*, 30(8):1004–1020, 1984.

[22] A. Charnes and W.W. Cooper. Goal programming and multiple objective optimizations. *European Journal of Operational Research*, 1(1):39–54, 1977.

[23] J. D. Knowles and D. W. Corne. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 98–105, 1999.

[24] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Parallel Problem Solving from Nature PPSN VI*, pages 849–858, 2000.

[25] K. Deb. Multi-objective optimization using evolutionary algorithms: an introduction. *Multi-objective evolutionary optimisation for product design and manufacturing*, pages 1–24, 2011.

[26] D. Kunkle. A summary and comparison of MOEA algorithms. *Northeastern University in Boston, Massachusetts*, 2(2003):1–19, 2005.

[27] R. G. L. D'Souza, K. C. Sekaran, and A. Kandasamy. Improved NSGA-II Based on a Novel Ranking Scheme. *Journal of Computing*, 2(2):91–95, 2010.

[28] M. H. Hassan, M. Palonen, and A. Hasan. Implementation of Pareto-archive NSGA-II Algorithms to a Nearly-zero-energy Building Optimisation Problem. In *Building Simulation and Optimization*, number September, 2012.

[29] J. Neyman. On the Two Different Aspects of the Representative Method : The Method of Stratified Sampling and the Method of Purposive Selection. *Journal of the Royal Statistical Society*, 97(4):558–625, 1934.

[30] M. D. McKay, R. J. Beckman, and W. J. Conover. Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239–245, 1979.

[31] K. Q. Ye. Orthogonal Column Latin Hypercubes and Their Application in Computer Experiments. *Journal of the American Statistical Association*, 93(444):1430–1439, 1998.

[32] T. M. Cioppa and T. W. Lucas. Efficient Nearly Orthogonal and Space-Filling Latin Hypercubes. *Technometrics*, 49(1):45–55, 2007.

[33] S.M. Sanchez. NOLHdesigns spreadsheet. Available online via harvest.nps.edu, 2011.

[34] N. A. C. Cressie. *Statistics for Spatial Data.* John Wiley adn Sons, INC., revised edition, 1993.

[35] L. V. Santana-Quintero, A. A. Montano, and C. A. C. Coello. A Review of Techniques for Handling Expensive Functions in Evolutionary Multi-Objective Optimization. *Computational Intelligence in Expensive Optimization Problems*, 2:29–59, 2010.

[36] D. S. Broomhead and D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2:321– 355, 1988.

[37] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. *Advances in Neural Information Processing Dystems*, 1:155–161, 1997.

[38] A. J. Smola and B. Schölkopf. A Tutorial on Support Vector Regression. *Statistics and Computing*, 14(3):199–222, 2004.

[39] M. Friendly. A Brief History of Data Visualization. In *Handbook of Computational Statistics: Data Visualization*. Springer-Verlag, iii edition, 2006.

[40] S. Few. Data visualization for human perception. https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/data-visualization-for-human-perception.

[41] W. S. Cleveland and R. McGill. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the Association*, 79(387):531–554, 1984.

[42] K. Pearson. Mathematical Contributions to the Theory of Evolution.– On a Form of Spurious Correlation Which May Arise When Indices Are Used in the Measurement of Organs. *Proc. R. Soc. Lond*, 1:489–498, 1896.

A Sample Configurations

Vm Type The Vm sizes used in the experiment, all types are prefixed with Standard_ in Azure.

Count The number of machines in the cluster.

Weight 1 Service Fabric balancing weight for Actor executions.

Weight 2 Service Fabric balancing weight for Actor default metrics.

Weight 3 Service Fabric balancing weight for Events handled.

Weight 4 Service Fabric balancing weight for Eventhandler default metrics.

Weight 5 Service Fabric balancing weight for Dispatch queue size.

Weight 6 Service Fabric balancing weight for Dispatcher default metrics.

Weight 7 Service Fabric balancing weight for default metrics of the other services.

Ratio 1 Service Fabric balancing threshold for Actor executions.

Ratio 2 Service Fabric balancing threshold for Events handled.

Ratio 3 Service Fabric balancing threshold for Dispatch queue size.

Ratio 4 Service Fabric balancing threshold for default metrics.

The first 65 samples are generated with NOHL design for up to 16 factors using only the first 13 columns and are used for training the heuristics. The final 5 samples are generated randomly and are used for testing the heuristics.

ID	Vm Type	Count	Weight 1	Weight 2	Weight 3	Weight 4	Weight 5	Weight 6	Weight 7	Ratio 1	Ratio 2	Ratio 3	Ratio 4
1	D2_V2	5	Low	Low	Zero	Medium	Medium	Low	High	3.88	3.19	4.75	1.75
2	D3_V2	12	Zero	Low	Low	Low	Medium	Medium	Medium	4.69	4.06	2.94	1.94
3	D3_V2	9	High	Low	Low	High	Zero	Low	Low	3.31	4.19	4.5	2.88
4	D2_V2	14	Medium	Low	Zero	Low	Low	Low	Zero	4.94	4.63	3.5	1.31
5	D3_V2	10	Low	Zero	Zero	Zero	Low	Medium	Medium	2.94	1.25	4.38	3.56
6	D2_V2	14	Low	Low	Zero	Medium	Zero	High	High	1.44	2.63	4.13	4.56
7	D2_V2	7	Medium	Zero	Low	Low	High	Low	Zero	2.75	2.38	4.63	4.81
8	D2_V2	13	High	Low	Low	High	Medium	Zero	Low	2.25	1.06	3.56	3.88
9	D2_V2	5	Zero	Medium	Low	Medium	Low	Zero	Medium	2.19	4.38	2.56	2.31
10	D3_V2	12	Low	Medium	Zero	Low	High	Low	Medium	1	3.69	1.13	2.69
11	D2_V2	5	High	Medium	Low	Low	Low	Medium	Zero	1.94	3.88	1.75	1.88
12	D3_V2	10	Medium	High	Low	Zero	Low	Medium	Low	2.5	3.13	2	2.81
13	D2_V2	7	Low	Medium	Low	Low	Low	Low	High	4.81	1.56	1.19	4.94
14	D2_V2	10	Low	High	Low	High	Zero	Zero	Medium	3.13	2	2.19	3.06
15	D2_V2	8	Medium	High	Low	Low	Low	High	Zero	4.31	1	2.75	4.88
16	D2_V2	11	Medium	Medium	Zero	High	High	Medium	Low	4.25	2.44	1.44	3.75
17	D3_V2	9	Low	Zero	Medium	High	Medium	High	Medium	3.38	2.25	2.13	2.63
18	D2_V2	14	Low	Low	Medium	Low	Medium	Medium	Low	3.94	1.44	1	1
19	D2_V2	9	Medium	Low	Medium	High	Zero	Low	Medium	4.19	2.94	1.31	1.81
20	D2_V2	12	High	Zero	Medium	Low	Low	Low	Medium	4.5	1.19	2.69	1.69
21	D2_V2	7	Low	Zero	High	Low	Zero	Medium	Low	1.38	2.69	1.69	3.44
22	D2_V2	13	Zero	Zero	Medium	Medium	Low	High	Low	2.38	4.69	2.25	3.94
23	D3_V2	9	Medium	Low	High	Zero	Medium	Zero	Medium	2.31	3.25	2.06	3.25
24	D2_V2	15	Medium	Low	Medium	Medium	Medium	Low	High	1.25	3.81	1.81	4.44
25	D2_V2	8	Zero	Medium	Medium	Medium	High	Zero	Zero	1.63	2.06	3.63	1.5

ID	Vm Type	Count	Weight 1	Weight 2	Weight 3	Weight 4	Weight 5	Weight 6	Weight 7	Ratio 1	Ratio 2	Ratio 3	Ratio 4
26	D2_V2	11	Zero	High	Medium	Low	Medium	Low	Low	2.81	1.13	4.44	2.38
27	D3_V2	6	Medium	Medium	Medium	High	Low	High	Medium	2	1.88	3.38	2.19
28	D3_V2	13	Medium	Medium	High	Low	Zero	Medium	Medium	2.44	2.5	4.94	1.25
29	D3_V2	8	Zero	Medium	Medium	Low	Low	Zero	Low	4.44	4.5	3.19	4.63
30	D2_V2	14	Low	Medium	High	High	Low	Medium	Zero	4.13	3.44	3.69	4.38
31	D2_V2	6	Medium	Medium	High	Low	Medium	Medium	High	4.88	4.25	3.13	4
32	D3_V2	12	High	High	Medium	Medium	High	Medium	Medium	3.44	4.31	4.06	3.5
33	D2_V2	10	Medium	Medium	Medium	Medium	Medium	Medium	Medium	3	3	3	3
34	A5	15	Medium	Medium	High	Low	Low	Medium	Zero	2.13	2.81	1.25	4.25
35	A2	8	High	Medium	Medium	Medium	Low	Low	Low	1.31	1.94	3.06	4.06
36	A2	11	Zero	Medium	Medium	Zero	High	Medium	Medium	2.69	1.81	1.5	3.13
37	A5	6	Low	Medium	High	Medium	Medium	Medium	High	1.06	1.38	2.5	4.69
38	A2	10	Medium	High	High	High	Medium	Low	Low	3.06	4.75	1.63	2.44
39	A5	6	Medium	Medium	High	Low	High	Zero	Zero	4.56	3.38	1.88	1.44
40	A5	13	Low	High	Medium	Medium	Zero	Medium	High	3.25	3.63	1.38	1.19
41	A5	7	Zero	Medium	Medium	Zero	Low	High	Medium	3.75	4.94	2.44	2.13
42	A5	15	High	Low	Medium	Low	Medium	High	Low	3.81	1.63	3.44	3.69
43	A2	8	Medium	Low	High	Medium	Zero	Medium	Low	5	2.31	4.88	3.31
44	A5	15	Zero	Low	Medium	Medium	Medium	Low	High	4.06	2.13	4.25	4.13
45	A2	10	Low	Zero	Medium	High	Medium	Low	Medium	3.5	2.88	4	3.19
46	A5	13	Medium	Low	Medium	Medium	Medium	Medium	Zero	1.19	4.44	4.81	1.06
47	A5	10	Medium	Zero	Medium	Zero	High	High	Low	2.88	4	3.81	2.94
48	A5	12	Low	Zero	Medium	Medium	Medium	Zero	High	1.69	5	3.25	1.13
49	A5	9	Low	Low	High	Zero	Zero	Low	Medium	1.75	3.56	4.56	2.25
50	A2	11	Medium	High	Low	Zero	Low	Zero	Low	2.63	3.75	3.88	3.38

ID	Vm Type	Count	Weight 1	Weight 2	Weight 3	Weight 4	Weight 5	Weight 6	Weight 7	Ratio 1	Ratio 2	Ratio 3	Ratio 4
51	A5	6	Medium	Medium	Low	Medium	Low	Low	Medium	2.06	4.56	5	5
52	A5	11	Low	Medium	Low	Zero	High	Medium	Low	1.81	3.06	4.69	4.19
53	A5	8	Zero	High	Low	Medium	Medium	Medium	Low	1.5	4.81	3.31	4.31
54	A5	13	Medium	High	Zero	Medium	High	Low	Medium	4.63	3.31	4.31	2.56
55	A5	7	High	High	Low	Low	Medium	Zero	Medium	3.63	1.31	3.75	2.06
56	A2	11	Low	Medium	Zero	High	Low	High	Low	3.69	2.75	3.94	2.75
57	A5	5	Low	Medium	Low	Low	Low	Medium	Zero	4.75	2.19	4.19	1.56
58	A5	13	High	Low	Low	Low	Zero	High	High	4.38	3.94	2.38	4.5
59	A5	9	High	Zero	Low	Medium	Low	Medium	Medium	3.19	4.88	1.56	3.63
60	A2	14	Low	Low	Low	Zero	Medium	Zero	Low	4	4.13	2.63	3.81
61	A2	7	Low	Low	Zero	Medium	High	Low	Low	3.56	3.5	1.06	4.75
62	A2	12	High	Low	Low	Medium	Medium	High	Medium	1.56	1.5	2.81	1.38
63	A5	6	Medium	Low	Zero	Zero	Medium	Low	High	1.88	2.56	2.31	1.63
64	A5	14	Low	Low	Zero	Medium	Low	Low	Zero	1.13	1.75	2.88	2
65	A2	8	Zero	Zero	Low	Low	Zero	Low	Low	2.56	1.69	1.94	2.5
66	D2_V2	8	Low	Zero	High	High	Medium	Zero	High	1.516	1.830	3.418	2.76
67	A2	10	Zero	Zero	Low	Low	High	Zero	High	4.612	3.512	4.789	3.35
68	A5	11	Low	Zero	Medium	High	Low	Medium	Low	2.029	4.196	1.778	1.03
69	D3_V2	13	Low	Low	Medium	High	Low	Medium	Low	4.550	1.911	3.165	1.75
70	A5	9	Zero	Zero	Zero	High	Zero	High	Zero	4.356	4.041	3.080	3.52

B Results

O1 Events per second - Default - Inserts

O2 Events per second - Default - Updates

O3 Events per second - Stress - Inserts

O4 Events per second - Stress - Updates

O5 Command duration - Default - Inserts

O6 Command duration - Default - Updates

O7 Command duration - Stress - Inserts

O8 Command duration - Stress - Updates

O9 Events per second - Default - Combined

O10 Events per second - Stress - Combined

O11 Command duration - Combined

ID	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11
1	114.7914	75.04885	81.72864	67.72253	86.03015	105.7126	104.0164	122.9916	94.92014	74.72559	104.6877
2	104.8297	71.37305	104.669	62.54412	90.93925	85.16875	94.20655	97.80301	88.10136	83.60658	92.02939
3	107.1134	80.6805	104.8074	70.15052	97.2697	101.0951	97.07876	87.73414	93.89697	87.47894	95.79441
4	102.6388	61.24717	149.7964	125.753	91.39976	96.76773	88.89392	92.99883	81.94296	137.7747	92.51506
5	117.5852	63.34212	81.44345	62.88863	96.02775	92.73115	87.5061	85.01622	90.46365	72.16604	90.3203
6	118.2917	59.47448	99.76095	50.34468	92.69785	92.33305	92.08368	97.99815	88.8831	75.05282	93.77818
7	131.2646	67.23177	129.8805	56.46274	109.1312	84.42628	111.5526	104.0471	99.24818	93.17162	102.2893
8	116.556	52.24108	72.7695	53.20525	84.70403	89.44483	92.38489	92.14206	84.39854	62.98738	89.66895
9	108.5525	79.93498	98.72849	75.17696	103.8281	90.89846	127.3888	96.53376	94.24375	86.95272	104.6623
10	99.40717	54.93946	63.15304	51.51925	91.0307	81.08698	84.00645	101.553	77.17331	57.33614	89.41928
11	106.7658	57.34851	95.46728	64.20824	100.3787	83.8531	104.1909	109.3687	82.05714	79.83776	99.44786
12	87.14591	51.33102	77.59856	52.01458	81.95092	98.29838	86.61238	89.46755	69.23846	64.80657	89.08231
13	109.2997	52.88912	82.86566	57.73872	92.68333	104.3042	87.16638	96.51415	81.0944	70.30219	95.16702
14	97.64324	51.26912	94.23897	60.08369	116.2258	111.0655	99.9616	95.12072	74.45618	77.16133	105.5934
15	87.84389	52.32875	165.9402	114.0694	91.69203	86.75904	98.27052	91.59425	70.08632	140.0048	92.07896
16	107.6837	65.12067	82.46149	43.88897	107.2337	93.65026	102.6417	88.64989	86.40217	63.17523	98.04387
17	150.1984	51.24725	100.2503	88.67947	97.41975	84.34456	108.724	95.3989	100.7228	94.4649	96.4718
18	92.0917	45.6314	56.66397	40.22381	108.2274	115.9779	96.93288	109.1312	68.86155	48.44389	107.5673
19	112.2852	72.22469	91.79188	63.3919	104.3491	95.16399	108.2843	122.8903	92.25496	77.59189	107.6719
20	96.09173	59.74266	74.67957	45.1835	106.6772	108.0503	102.9696	95.25614	77.91719	59.93153	103.2383
21	88.18253	53.57297	68.895	45.78346	114.9931	108.4925	111.1413	108.9903	70.87775	57.33923	110.9043
22	94.52305	47.70656	60.93093	58.15966	87.02178	101.821	91.82597	92.4955	71.11481	59.54529	93.29106
23	153.4245	45.73869	83.26189	69.07319	108.6128	107.2625	105.8178	112.4236	99.58162	76.16754	108.5292
24	101.0126	50.95554	60.1767	32.80113	92.12932	94.85953	102.8915	96.6593	75.98409	46.48891	96.6349
25	77.39404	53.06562	126.453	33.4142	95.98898	94.68365	109.444	96.34274	65.22983	79.9336	99.11485

ID	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11
26	94.64571	65.20402	83.33575	53.31193	94.24228	94.92664	90.25895	116.4206	79.92487	68.32384	98.96211
27	106.2928	66.89113	118.9176	95.19244	91.7128	99.47554	128.8462	110.0925	86.59196	107.055	107.5318
28	152.9694	110.5248	121.2135	114.3341	88.46227	104.3275	111.5062	90.32051	131.7471	117.7738	98.65411
29	168.377	115.4207	140.8317	106.7403	89.90456	101.085	108.6069	96.35355	141.8989	123.786	98.9875
30	170.5599	126.9359	155.5943	117.9635	84.6171	90.5472	91.99702	119.0204	148.7479	136.7789	96.54544
31	172.6334	124.8041	128.9604	122.5616	97.47433	100.9088	101.2112	104.4591	148.7188	125.761	101.0134
32	144.9713	113.5051	135.9599	111.2906	106.4936	88.85998	113.8449	124.9336	129.2382	123.6253	108.533
33	138.15	113.743	126.1961	103.0905	110.1962	95.60881	116.0887	100.5647	125.9465	114.6433	105.6146
34	147.4902	102.5964	147.9203	123.9445	119.8046	112.3748	113.6263	116.9623	125.0433	135.9324	115.692
35	136.7096	111.1947	96.77261	53.19638	146.7731	144.3161	206.1333	160.1825	123.9521	74.98449	164.3513
36	135.9014	121.9403	127.3609	92.47663	123.142	104.146	146.8089	129.8979	128.9209	109.9187	125.9987
37	125.6462	105.825	104.3553	92.20651	140.67	143.8062	196.9849	218.0397	115.7356	98.28088	174.8752
38	144.069	110.8301	137.5281	93.68846	124.0743	118.5031	138.8341	136.1585	127.4496	115.6083	129.3925
39	117.5732	102.5723	81.20394	76.02791	151.9315	141.5811	179.2472	146.2765	110.0727	78.61592	154.7591
40	144.9403	111.556	124.6242	95.12881	100.2227	104.8768	215.8028	121.364	128.2482	109.8765	135.5666
41	109.8127	87.53687	99.76253	86.58968	146.2534	163.0258	212.407	156.7678	98.6748	93.1761	169.6135
42	117.4694	84.79369	128.3333	106.4299	115.8956	98.18057	105.01	103.2417	101.1315	117.3816	105.582
43	124.0964	98.91671	112.2907	95.45712	136.291	116.1563	156.918	131.0222	111.5066	103.8739	135.0969
44	148.1187	126.2054	151.2857	119.2539	125.5158	118.1348	115.0444	118.8571	137.162	135.2698	119.388
45	113.853	94.09633	106.2982	75.81269	119.7018	131.9422	150.407	143.7973	103.9747	91.05545	136.4621
46	116.9985	96.8421	132.841	112.2293	134.4763	118.6639	119.7646	113.3929	106.9203	122.5352	121.5744
47	131.1963	110.1643	138.862	112.9244	122.7209	119.4187	129.2279	130.534	120.6803	125.8932	125.4754
48	136.4444	101.007	98.33291	65.4126	125.6471	117.6506	140.2428	132.547	118.7257	81.87276	129.0219
49	134.3153	110.9871	120.2895	110.7786	130.771	137.1177	128.5637	120.5466	122.6512	115.5341	129.2498
50	139.38	103.8461	128.7817	108.5831	134.9612	122.5581	113.707	129.1187	121.6131	118.6824	125.0863

ID	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11
51	86.68322	87.67976	93.76983	82.94228	180.1667	185.6121	197.3252	223.1696	87.18149	88.35606	196.5684
52	123.6591	105.7232	116.0989	98.6704	119.1036	119.1523	126.2799	140.7884	114.6912	107.3847	126.3311
53	125.5237	99.69458	135.7304	100.3006	134.1128	124.286	133.3063	140.3328	112.6092	118.0155	133.0094
54	132.7382	114.3143	146.1699	118.7265	115.572	103.9229	121.9889	114.0803	123.5263	132.4482	113.891
55	83.6872	71.35818	119.1511	102.6556	160.1157	174.4027	150.1557	164.2734	77.52269	110.9034	162.2369
56	137.3357	119.9337	126.9104	96.92902	112.8334	101.7307	121.2815	116.8521	128.6347	111.9197	113.1744
57	116.3609	100.1532	93.25331	88.25181	151.5299	161.6237	198.6542	217.4299	108.2571	90.75256	182.3094
58	152.3521	120.9451	125.1438	101.4333	131.9048	91.41113	108.6834	127.8104	136.6486	113.2885	114.9524
59	110.7745	101.1486	127.3286	74.78005	117.7871	114.3459	120.3233	110.6775	105.9615	101.0543	115.7834
60	90.12787	66.44067	151.217	115.9706	112.1947	110.3438	109.6502	113.7628	78.28427	133.5938	111.4878
61	121.5611	102.6283	115.8177	108.328	152.4635	141.3305	144.5332	137.103	112.0947	112.0728	143.8575
62	136.9147	102.8941	139.6809	100.5042	109.0514	111.9231	123.0081	117.7247	119.9044	120.0926	115.4268
63	97.11666	77.85012	93.38577	72.63215	151.7351	140.5692	199.7038	217.4563	87.48339	83.00896	177.3661
64	138.8374	101.4655	74.08447	99.58069	120.3213	107.0751	120.9244	116.8182	120.1514	86.83258	116.2847
65	121.0305	92.83283	108.6853	87.0793	163.1282	189.7853	157.7391	148.2838	106.9317	97.88231	164.7341
66	158.6555	119.6287	142.4997	118.951	90.35284	108.7717	94.96264	106.2701	139.1421	130.7254	100.0893
67	130.7692	106.5389	132.7667	105.2915	117.5455	130.4654	120.5246	111.6382	118.654	119.0291	120.0434
68	93.82453	83.09332	105.2493	89.58382	160.3952	139.1004	166.4842	176.7044	88.45892	97.41658	160.6711
69	143.1445	113.4444	118.1653	99.73639	102.5009	92.9506	109.0914	99.85329	128.2944	108.9508	101.0991
70	127.5995	113.0045	138.4465	105.1395	122.939	125.8401	142.458	128.2741	120.302	121.793	129.8778