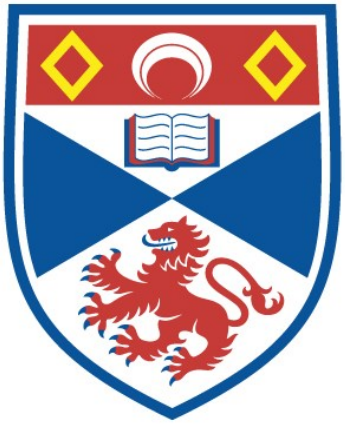
Automating inventory composition management for bulk purchasing cloud brokerage strategy

Chalee Boonprasop

A thesis submitted for the degree of PhD
at the
University of St Andrews



2024

Identifier to use to cite or link to this thesis:
DOI: https://doi.org/10.17630/sta/899

Abstract

Cloud providers offer end-users various pricing schemes to allow them to tailor VMs to their needs, e.g., a pay-as-you-go billing scheme, called *on-demand*, and a discounted contract scheme, called *reserved instances*. This work presents a cloud broker that offers users both the flexibility of on-demand instances and some discounts found in reserved instances. The broker employs a buy-low-and-sell-high strategy that places user requests into a resource pool of pre-purchased discounted cloud resources.

A key challenge to buy-in-bulk-sell-individually cloud broker business models is to estimate user requests accurately and then optimise the stock level accordingly. Given the complexity and variety of the cloud computing market space, the number of the regression model and inherently optimisation search space can be intricate.

In this thesis, we propose two solutions to the problem. The first solution is a risk-based decision model. The broker takes a risk-oriented approach to dynamically adjust the resource pool by analysing user request time series data. This approach does not require a training process which is useful at processing the large data stream. The broker is evaluated with high-frequency real cloud datasets from Alibaba. The results show that the overall profit of the broker is closely related to the optimal case. Additionally, the risk factors work as intended. The system hires more reserved instances when it can afford while leaning to the on-demand otherwise. We can also conclude that there is a correlation between the risk factors and the profit. On the other hand, the risk factor possesses some limitations, i.e. manual risk configuration, limited broker setting.

Secondly, we propose a broker system that utilises the concept of causal discovery. From the risk-based solution, we can see that if there are parameters correlated with the profit, then by adjusting those parameters, we can manipulate the profit. We infer a function mapping from the extracted key entities of broker data to an objective of a broker, e.g. profit. The technique is similar to the additive noise model, causal discovery method. These functions are assumed to describe an actual underlying behaviour of the profit with respect to the parameters. Similar to the risk-based, we use the Alibaba trace data to simulate long term user requests. Our results show that the system can infer the underlying interaction model between variables unlock the profit model behaviour of the broker system.

Acknowledgements

I would like to thank my supervisors (Professor Adam Barker and Yuhui Lin) for helping and guiding me with this project. Along with that, I would also like to thank my school (School of Computer Science, University of St Andrews) wholeheartedly.

I would also like to thank my sponsor the Royal Thai Government for the funding during my PhD study.

Lastly, I would like to thank my parents and friends who helped me finalise this project within a limited time frame.

Declaration

Candidate's Declarations

I, Chalee Boonprasop, do hereby certify that this thesis, submitted for the degree of PhD, which is approximately 50,000 words in length, has been written by me, and that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for any degree. I confirm that any appendices included in my thesis contain only material permitted by the 'Assessment of Postgraduate Research Students' policy.

I was admitted as a research student at the University of St Andrews in May 2017.

I received funding from an organisation or institution and have acknowledged the funder(s) in the full text of my thesis.

Date: May 5, 2024

Signature of candidate:

Supervisor's Declaration

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Doctor of Philosophy in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date:

Signature of supervisor:

Permission for publication

In submitting this thesis to the University of St Andrews we understand that we are giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. We also understand, unless exempt by an award of an embargo as requested below, that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that this thesis will be electronically accessible for personal or research use and that the library has the right to migrate this thesis into new electronic forms as required to ensure continued access to the thesis. I, Chalee Boonprasop, confirm that my thesis does not contain any third-party material that requires copyright clearance. The following is an agreed request by candidate and supervisor regarding the publication of this thesis:

Printed copy

No embargo on print copy.

Electronic copy

No embargo on electronic copy.

Date: May 5, 2024

Signature of candidate:

Date:

Signature of supervisor:

Underpinning Research Data or Digital Outputs

Candidate's declaration

I, Chalee Boonprasop, hereby certify that no requirements to deposit original research data or digital outputs apply to this thesis and that, where appropriate, secondary data used have been referenced in the full text of my thesis.

Date: May 5, 2024

Signature of candidate:

Permission for Electronic Publication

In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that my thesis will be electronically accessible for personal or research use unless exempt by award of an embargo as requested below, and that the library has the right to migrate my thesis into new electronic forms as required to ensure continued access to the thesis. I have obtained any third-party copyright permissions that may be required in order to allow such access and migration, or have requested the appropriate embargo below.

The following is an agreed request by candidate and supervisor regarding the electronic publication of this thesis:

> *Access to printed copy and electronic publication of thesis through the University of St Andrews.*

Date: May 5, 2024

Signature of candidate:

Signature of supervisor:

I want to thank everyone who takes part in helping me finish this project. This study has been particularly challenging due to the unique situation during my time at St Andrews. And special thanks to my family, who are always with me through all ups and downs.
Chalee

CONTENTS

List of Figures

List of Tables

CHAPTER ONE

INTRODUCTION

1.1 Cloud Computing

Cloud computing is currently one of the most talked-about technologies. Up until now, many of the businesses have been moved or planned to move their infrastructure to the cloud (48; 30). The move is to improve and enhance the capability of the business by taking advantage of the strengths of cloud computing. Not only businesses, regular computer enthusiasts and small businesses are now interested in the adoption of the cloud as well (65). Cloud computing as an entity has changed the computing paradigm as we know it.

Cloud computing as a concept is defined as "cloud computer is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" by the National Institute of Standards and Technology (90). From the definition, cloud computing is a convenient method of accessing computing resources over the network. Remote access benefits the pooling resources giving access to higher performing computers rather than the usage of local machines. However, more often than not, the process of selecting and configure cloud computers create the first barrier of complication for cloud customers. Without access to the physical hardware, cloud users are at the mercy of network connection from the local machine to the cloud. Offloading computing workloads from the local computer to the cloud need the connection to be good enough to ensure the experience of using the cloud. Thus, cloud computing only started to take off when the computing power can be accessed over a network quickly, i.e. via broadband internet. Even then, computer users come up with the first cloud concept within the institution whose mainframe access time has to be shared (16; 72). After the idea become commercialised, cloud computing surged in popularity in recent years due to the popularity of off-loading computing

power from personal devices (smartphone, laptops) to the data centre taking advantage of higher computing power (89).

In the market space, there is a considerable number of cloud providers. Each of them tries to offer the best cloud experience to the customers. The selection of cloud instances, performance, stability, geo-location coverage, and price are among many features that the providers try to offer to the customers to compete with each other. On the pricing offering, cloud providers try to provide lower prices and various pricing schemes. On the surface level, the pricing schemes from cloud providers are simple to understand and cater well to customers' need. For example, on-demand instances are for customers who need to use cloud computing for a short time. It offers good flexibility in exchange for a higher price compared to other pricing schemes. On the other hand, the reserved instances offer a substantially lower price than the on-demand for the same performance. The condition of the reserved instance is a lengthy contract time or, in some cases, high upfront cost. For a customer, these pricing schemes create choices and constraints at the same time. Customers with low time usage have no alternative but to pay higher prices. Together with diverse instance specifications, cloud computing is a complex entity which new cloud customers might not be entirely familiar (114).

1.2 Cloud Brokerage System

The origin of a brokerage service starts from the complexity and choices of the services in the market. A broker job is to streamline the complexity, make the process easier for the customers, and earn a profit. The behaviour is not strictly unique to the cloud market. In a cloud environment, the responsibility of the broker is more than just streamlining the market. Some brokerage services offer assistance such as a common market space, value-added merchandise (cloud instance) and more. This work focuses on the broker that offers lower price cloud instances to the customers, making it more affordable for ordinary customers. The challenge of a broker is to balance its inventory composition such that it can turn more profit.

Cloud brokerage service is to connect and bridge the gap between cloud providers and cloud customers (33). This gap is the difficulty in the adoption of the cloud, especially for relatively new cloud customers. The first barrier is the complexity of the cloud instance. For example, cloud compute units description from many providers are not entirely explicit. Providers usually display the specification as "vCPU". Therefore, customers have a difficult time choosing the best instance for their applications.

Another barrier that deter cloud usage is the price of cloud services. Although, cloud providers advertise that in many cases, using the cloud is more convenient and cheaper. However, in reality,

cloud instances usage per unit of time is a lot more expensive than using the computers on-site, particularly in a small scale deployment and short term usage (129). Therefore, this work focuses on the cloud inventory data estimation and optimisation strategy of the broker. The solution tries to generate more profit through the price differences of multiple cloud pricing schemes. The profit is passed to cloud customers making the overall cost of the cloud cheaper. A cheaper cloud will bring more customers to accelerate cloud adoption and help develop cloud computing in general. The growth in cloud customers should benefit both the industry and the development of cloud technology (77).

There are numerous service brokerages, and each of them aims to add features or ease the process of using cloud computing. As mentioned earlier, cost is an issue. A broker system that utilises cheaper instances such as reserved instances as their primary commodity in its inventory and resells the cloud instances to the customers as an on-demand type can benefit from the price difference and pass these profits to the customers. By doing so, customers can still enjoy the benefit of on-demand flexibility while can control many of the quality of services (QoS), such as performance and latency.

1.3 Broker Inventory Optimisation Problem

Cloud computing space is a lucrative market. Therefore, the providers are trying to cater to as many customers as possible. This leads to a sizeable number of choices, whether the instance types or pricing schemes. A broker is a cloud customer as well. Thus, it too has to deal with the complexity of cloud computing. Examples of pricing schemes are on-demand and reserved instances. An on-demand instance is the most straightforward pay per usage type by the unit time, i.e. hour. A reserved instance is a long-term instance where there is a fixed contract of usage.

The value proposition of each pricing scheme varies from customer to customer. If the customer values performance per time above anything else, then the best value would be the reserved instance that offers the highest performance for the money. However, if a customer values flexibility, then an on-demand type of instance would be better suited. For a broker, the concept of value cannot be measured by the price per performance or price per duration alone. Nevertheless, the objective of a broker when renting cloud instances from cloud providers is to have the best cloud instances that generate the best profit in its inventory. Hence, deciding the pricing scheme of cloud instances added to the inventory when needed plays a vital role in the system's profitability.

Given that cloud customers demand cannot be controlled by the broker, finding the right

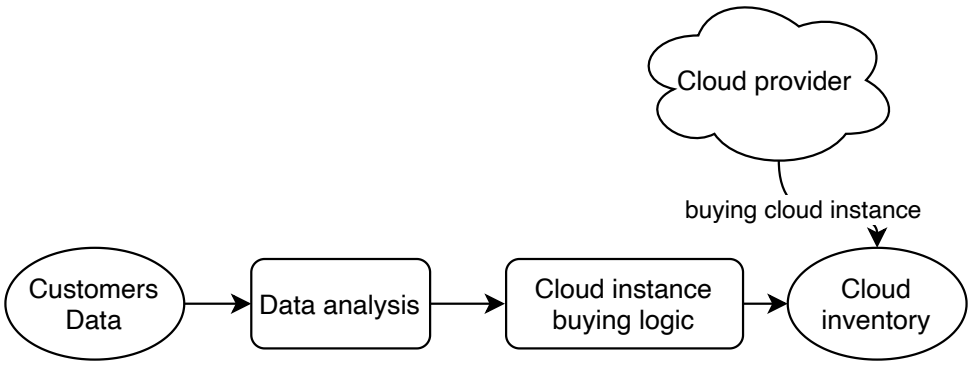


Figure 1.1: Figure shows a cloud broker process starting from analysing customer data to cloud inventory adjustment. Customers data, e.g. VM types, usage time, are passed through the data analysis process, such as the demand estimation. After that, data analysis is used to build cloud instance buying logic or aid the broker's decision-making, forming the broker inventory.

combination of cloud instances in its inventory is not trivial. One solution would be that if a broker can predict customers' demand and match them with the right set of cloud instances, then the broker can achieve an optimal profit level. Generally speaking, there are two main aspects in an attempt to achieve the goal of having an optimal profit, data analysis (data estimation) and inventory buying decision logic (optimisation).

From Figure 1.1, data analysis is a process of figuring out useful information within the data. Typically, the information is used for making a decision or keeping the operation running efficiently. Data analysis by a broker is the first step after receiving customers queries to find what aspect of the data is valuable to the broker system. The preparation step is the transformation of an individual query into usable data. Since the broker system processes individual queries chronologically, time series is the main data type that our broker uses. The next step in the broker system is to predict or estimate the demand. It is arguably one of the most important steps toward the high efficient broker. An accurate estimation would lead to good inventory preparation and thus best fit the customers' data. The demand prediction starts from deciding which aspects of the data need to be estimated, such as the overall demand, short term, or long term queries. The main issue with the prediction is the accuracy of the result. High accuracy is the ultimate goal and is challenging to achieve, especially when the data is volatile.

Buying logic is a decision-making process used in conjunction with the prediction to decide which cloud instances are to be in the inventory. Once the prediction results are obtained, a system builds a buying logic or a function based on the outcome. If the aim is to make an optimal decision for the broker, then a popular approach is to consider it an optimisation process, i.e. choosing the best cloud instance composition for the customers' data.

An optimisation is a process of selecting the best values of target variables within constraints of the system. It is the ideal method for determining the combination of cloud instances in the broker inventory on paper. However, an optimisation requires a well-defined domain which means unless the prediction is accurate, the broker system will not have the best result. Given that reserved instance duration typically ranges from one to three years, an accurate forecast of the data is less likely.

Since making decisions far into the future is difficult, a just-in-time (JIT) decision model is better suited for the system. A just-in-time function is also called a decision function. For example, a broker can set up a function that uses more reserved instances when the demand goes up. Alternatively, a broker can set the number of reserved instances in the inventory to be precisely the value of an average number of customer queries per unit time. All of these functions are valid decision methods. Nevertheless, to create a practical decision function, the system requires a more elaborate method. The decision function can created by multiple methods. Most notably, a form of regression can, in theory, find a model which a broker can use as a decision function. For example, a regression method can find a model of broker composition given historical data of optimal inventory composition to the customers' data. With the constructed model, a broker can follow the decision made by the optimal data and adjust the inventory according to the model and new customers data.

However, applying a regression method also has a non-trivial pre-processing, such as the selection of independent variables (128). Also, there could be variables other than the chosen independent one that are not in the model and influence the dependent variable. Additionally, a strong correlation from the regression does not simply imply a cause and effect relationship (83). Therefore, the decision model might make a wrong decision causing a lower yield in profit. Hence, a model of a system that infers the cause and effect as much as possible should be able to unlock the insight needed to make a better decision.

1.4 Our Approaches

Our system takes all given individual cloud customers queries and allocates them to respective cloud instances in its inventory as shown in Figure 1.2. The orders from customers are analysed, then the broker decides which cloud instances are to be put in the inventory. The decision model is the heart of the system, where the results made by it must fulfil many of the broker's conditions such as the guaranteed resource allocation or minimum computing power given to the customer.

In our system, customers' queries data consisted of starting time, performance level, and termination time. Starting time and performance level is, as the name suggests, a timestamp

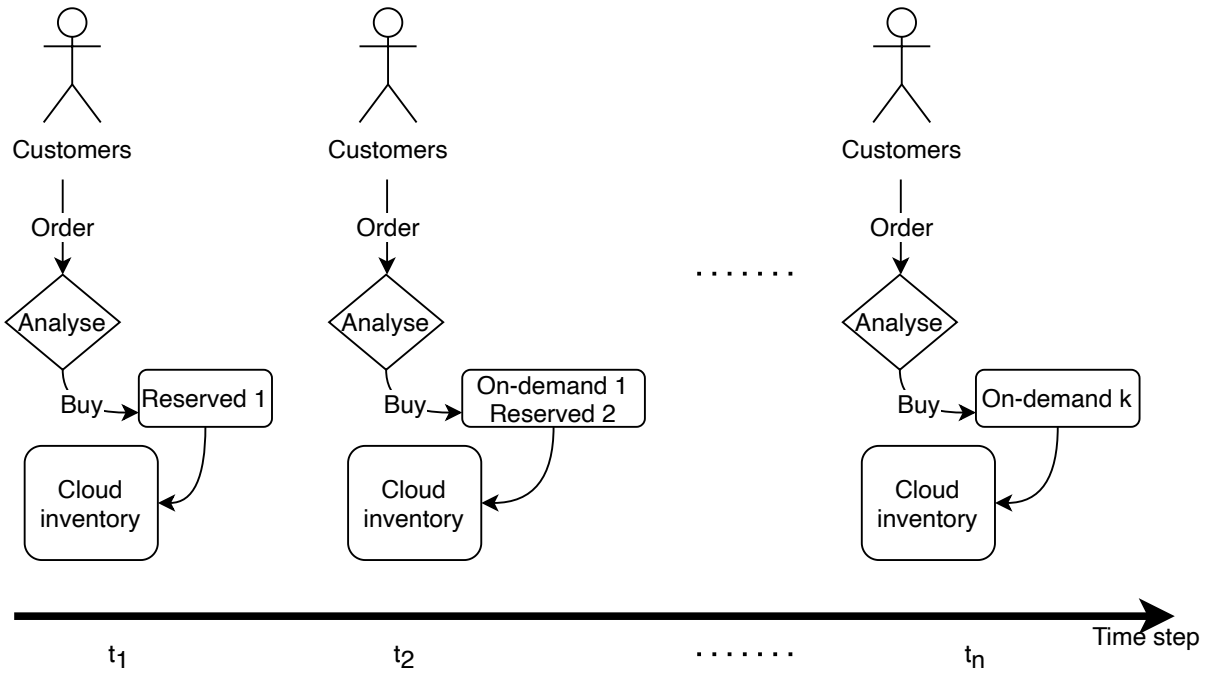


Figure 1.2: A broker system works by taking customers' queries, analyse them and then order cloud instances to be placed in the cloud inventory. The "analyse" step is what dictates the broker's inventory composition which directly influence the profit level of the broker.

of the starting query and the performance level requested by the customers. The termination time is not notified before hand by the user to the broker. The termination process triggers when the customer finishes the job ,making the cloud instances from brokers behave similarly to an on-demand instance.

The broker inventory is a set of cloud instances that consist of multiple pricing schemes and performance levels. For example, in a normal setup, an inventory with n reserved instances of the same performance level, m on-demand instances can host up to k customer queries of the same or multiple performance level.

Although, reserved instances are used as the primary instance in the inventory. If the situation demands, the broker can offload some of the queries to more expensive but less committed instances, i.e. on-demand instances, to reduce the chance of some reserved instances being underutilised. If the broker determine that the chance of newly created reserved instance is going to be underused, then the broker might order on-demand instances to be used in its inventory.

To build a decision function, we make an assumption that there are parameters associated or correlated with the profit of the broker. Suppose these parameters are manipulated or changed. Then profit of the broker system also changes. For example, hand-picked parameters such as risk factors from the solution in Chapter 5 can build a decision function to make the system

perform better profit-wise than a typical time series prediction method. Also, another approach is discovering the variables through an auto-model generator, a causal discovery-like method used to infer a relationship function between profit and other parameters in a broker system.

1.4.1 Risk Factors in a Brokerage System Inventory Management

The indicator method has been used in many financial applications where certainty and predictability are hard to come by (13). Indicators are parameters that try to predict the behaviour of the future values of a time series (108). The indicator indicates or predicts the directional movement of the future value in a time-series. The development of indicator usually involves the historical observation and manual data mapping. A similar approach is applied to the broker problem.

Adjusting broker inventory is done when the broker inventory cannot accommodate the incoming requests from cloud customers. The adjustment process is triggered by every exceeding query over the inventory capacity. Therefore, for every exceeding request, the broker has to choose a cloud instance. If the reserved instance is selected, then a broker could make a profit from the price difference. However, if the broker chooses an on-demand instance, the broker will make no profit.

The risk factors system is created with the risk concept in mind (25; 53). The principle of risk is that lower risk can bring lower profit (or deficit) and higher risk can bring higher profit (or deficit). The risk factor system aims to solve an unbalanced decision-making the broker inventory system. Assuming that there are some problems in the prediction, the underestimated solution will place more queries into more expensive on-demand instances. The chances of purchasing reserved instances into the inventory are lower, resulting in a potential loss of profit made from cheaper reserved instances. On the other hand, if the system overestimates the demand, the idle time of reserved instances is higher. Since the reserved instances usually have a significant upfront cost, it could lead to bad overall profit. We have set up a risk factors system so that they are to deal with a problem that might *happen* in the future. The risk factors are similar to the indicator where they try to help a broker make *correct* decisions when adjusting its inventory, since the risk factors are predetermined and shaped by situation rather than historical data-driven.

1.4.2 Decision model Generator Broker System

While risk factor is one concept that deals with uncertainty, the risk assessment process, on the other hand, is a tedious process and relies on human input and expertise. The next natural step for the broker system is to try to reduce the human involvement and develop a methodology of

identifying the decision-making function as its accompany parameters.

We develop a broker process which aim to find broker inventory composition from a set of parameters. It starts with feature extraction of the data as candidates for the model similar to the risk factors of the previous system. The feature parameters are potential parameters that might influence the profit. A typical method of finding or filter out these features, whether or not they are associated with the profit, is to perform a correlation test. The correlation test is a suitable method of finding the relation of an already known connection. However, a strong assumption (known relationship) is not easy to conclude. Hence, in this work, we are adapting the method from the additive noise model into our system (55). The additive noise model has three main components: function approximation, added noise, and independent testing. The chosen features and their inference functions are used as the profit model of a broker. Then, the inventory configuration is adjusted following the model. As a result, we build a more flexible profit model and require fewer human inputs.

1.5 Hypothesis & Contribution

The main claim of this work is:

> *A cloud brokerage service that primarily manages reserved instances can offer cloud services to customers at lower costs by developing a profit model that thoughtfully considers various parameters and their interactions.*

We identify and solve the following components to evaluate our hypothesis.

- HYP 1 - A bulk buying cloud broker operates under some risks from the uncertainty of the user requirements. These risks can be managed by identification of the risk factors and functional risk management.
- HYP 2 - A random process in a structural system has hidden causal parameters and relations that we can infer.
- HYP 3 - In an optimal system, parameters behave under certain underlying relations.
- HYP 4 - We can use the behaviour of the parameters to build a decision system for the cloud broker inventory.

Our solutions can have extended usages outside of the cloud computing area. Our methods' criteria are the uncertainty of the incoming data while the cost of planning the data accommodation

is high. For example, we can use the risk-based solution with the financial portfolio adjustment, while the auto-model can be used in agricultural planning.

The main contributions in the area of cloud computing research are:

1. A risk-based cloud inventory optimisation approach that utilises risk analysis to dynamically adjust instances stocking level without assuming the underlying distribution of user requests.

2. A novel approach using automatic profit model generator based on the additive noise model combined with feature selection method and the model health check toward the broker inventory optimisation for a profit.

3. A systematic cloud broker framework that automatically extracts features from time series for given goals.

4. An evaluation of both approaches using the data from Alibaba.

1.6 Organisation of Dissertation

A short description of each chapter in this dissertation is given below.

Chapter 2 presents the literature survey about cloud computing, cloud brokerage service, time series data, causal discovery, and the additive noise model. A brokerage system enhances the capability of cloud computing or, in some cases, reduces the cost of using cloud computing. One key challenge is to optimise the inventory by analysing customer demand time series data. In addition to the time series analysis, a causal discovery method can also be used to find the connections between broker parameters and the profit value.

Chapter 3 is the background that explains the main technologies in this work. First, the details of profit maximising of a cloud broker were given. The model of using reserved instances as the primary instance of cloud inventory can be profitable under certain conditions. After that, the detailed Auto-ARIMA model is given as one of the main comparison models in this work. The ARIMA model is one of the most popular time series models and the Auto-ARIMA simplifies fine-tuning the parameters and thus produces the best possible result within the constraints. Next, the Gaussian process is explained. The Gaussian process is the model generator in our broker system. Lastly, the causal discovery method is used as a model generator and parameter filtering in the additive noise model.

Chapter 4 shows the problem formulation and research motivations. It starts from the cloud broker business model and its challenges. The challenges include optimisation, data

estimation, and the profit model. Together, they form a research motivation. Lastly, this chapter proposes solutions and a high-level explanation of the two approaches, a cloud broker system based on risk-based solution and auto-model generator.

Chapter 5 shows a broker strategy that utilised risk factors. We build a broker inventory decision-making system that uses the risk factors and corresponding decision function. The combined functions form a linear decision model. The risk factors assessed from potential scenarios where the broker investment in reserved instances might cause a deficit in profit, such as the remaining reserved instances contract length in the inventory. The decision-making process decides between two types of cloud instances, on-demand and reserved. The decision making is triggered when there is a need to expand the broker inventory, i.e. customer requests exceed the size and availability of the broker. The logic behind risk factors is that if the risk is high, the system would balance itself and reduce the risk by adjusting the decision threshold. Therefore, the broker can avoid the unwanted idle reserved instance while lowering the opportunity cost. On the other hand, coming up with risk factors relies on human intuition and manually crafted associated functions, restricting the system's expansion.

Chapter 6 shows an extension of the broker profit system, rather than using the decision function from the risk factors. The broker uses an automatic profit model generator by utilising a method similar to the additive noise model (ANM). The ANM is one of the causal discovery method aiing to infer cause and effect relationship. Typically, the ANM is used to identify the causal pair of parameters. We are adapting the ANM to build a profit model from a selection of parameters. At the heart of the ANM, the Gaussian process is used to build an approximate function of the relation. Together with the independent testing between parameters, ANM produces a relation function that rejects or accepts the connection between parameters. This mechanism selects functional parameters concerning the profit. Utilising this technique, a broker can build a comprehensive map of the parameters that affect the system's ultimate target, profit. The identified model is used as a decision model when planing the broker inventory. The method can automatically filter the generated features and build a function around the chosen ones. As such, it can generate the decision model from the input data similar to the risk-based approach. The auto-model generator is more flexible than the risk-based due to the minimisation of human input.

Chapter 7 presents future directions of research and development in cloud brokerage system in general. This includes the topics that are not discussed in previous chapters. It starts with research directions of cloud brokers, such as cloud instance performance measurements and cloud instance matching. A good matching should make the decision on the suitable

cloud instance trivial for customers. Next, the broker inventory optimisation is discussed, followed by cloud services. Lastly, the conclusions of the dissertation are presented.

CHAPTER TWO

LITERATURE REVIEW

Cloud broker research is a multifaceted field that draws on various computer science disciplines. The rise of cloud computing was fueled by increasing demands for computing power and the advent of feasible remote access technologies. Since then, cloud computing has gained tremendous popularity in both private and public sectors. Consequently, numerous companies have begun investing in cloud infrastructure to secure an early market presence. As the market expands, cloud providers continue to diversify their offerings, presenting a range of cloud instance types and various pricing models for customers to choose from.

The complexity and fragmentation of the market often necessitate the introduction of third-party solutions. Similar to other types of brokers, a cloud broker facilitates cloud adoption for customers by providing services such as automatic selection of cloud instances and platforms for utilizing multiple clouds. Typically, cloud brokers charge an additional fee for these services. Some brokers capitalize on the price disparities across different providers' pricing schemes. By purchasing reserved instances at lower prices and reselling them to customers, brokers can generate profits. However, they must secure enough customers to cover the lengthy contract periods associated with reserved instances. Therefore, effective demand prediction methods are crucial to solve this challenge and potentially increase the broker's profitability.

In cloud brokerage systems, pricing schemes for cloud instances are typically based on usage time, requiring the broker's system to process individual customer cloud queries as time series data. This time series data is crucial as it also includes temporal elements like the expiration times of reserved instances. Consequently, our brokerage system primarily utilizes time series data.

Time series data is characterized by its vertical movements or changes in value at different time steps. These fluctuations, which can often appear random, present significant challenges for

analysis. The unpredictability in value changes—whether an increase or decrease—makes time series data complex to handle. Tools developed to analyze such data will be discussed, focusing on their application in a cloud brokerage context.

Furthermore, the task of selecting a suitable model and its parameters is a substantial challenge in brokerage systems and is often guided by the system administrator's experience, historical data, or a trial-and-error approach. The data generated by cloud brokerage systems, such as inventory composition, is a direct result of decision-making processes within the broker. Assuming these decisions yield optimal inventory composition, the resulting profit should also be optimal. However, establishing a clear link between customer demand data and broker profit is complex. One effective technique for exploring these relationships is causal discovery.

Causal discovery methods aim to identify causal relationships between variables without the need for re-experimentation. Among these methods, the additive noise model is particularly notable. This model introduces independent noise to determine if there is a cause-and-effect relationship between two variables with high precision. At its core, this model relies on function approximation, which is instrumental in developing the broker's profit model.

This chapter provides an overview of the foundational theories applied in this study, ranging from general cloud computing principles to specific methodologies like the additive noise model.

2.1 Cloud Computing

The computer generally referred to a machine that process operations automatically. Accessing the machine is done through input devices, which carry out the signal to the machine. The concept of accessing machines "remotely" was introduced as early as 1977 when the "internetting" was demonstrated (SATNET) (59). Remote access can arguably be the starting point of the "cloud", a symbol for networking since the input and output were off-site and was not at the same premise as the computer. Another concept of cloud computing might started from a time-sharing scheme on a mainframe computer (120). Scientific calculations are large scale workloads that usually are not possible on a personal computer. The workloads have to be moved to a better capable machine, such as a parallel mainframe. While highly utilised, it locked out other users when there are not enough computing capacity. Thus, it is necessary to arrange a time-sharing scheme or queuing system. Similar to the SATNET, users connect to computing resources remotely. Fast forward to the present day, cloud computing as we know it was popularised around the year 2000 when Amazon launched its Elastic Compute Cloud (EC2) platform (1). The service offers many computer services such as virtual machines, off-site data storage, machine learning models, etc. Many providers joined later, saturating and offering a competitive market for cloud users. Cloud

computing becomes one of the fastest-growing sectors in the digital market.

2.1.1 Definition of Cloud Computing

What is cloud computing? The National Institute of Standards and Technology's definition of cloud computing — National Institute of Standards and Technology (86) identifies "five essential characteristics":

- **On-demand self-service:** A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- **Broad network access:** Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
- **Resource pooling:** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
- **Rapid elasticity:** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear unlimited and can be appropriated in any quantity at any time.
- **Measured service:** Cloud systems automatically control and optimise resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilised service.

Although the definition can be varied from source to source, in general, cloud computing is viewed as a remote computing resource that customers have access to but cannot manipulate the infrastructure itself, which is similar to the actual cloud.

2.1.2 Cloud Computing Services

As remotely accessed computers, cloud services offer similar computing services as locally accessed computers if allowed by the latency and bandwidth. While limited, cloud computing

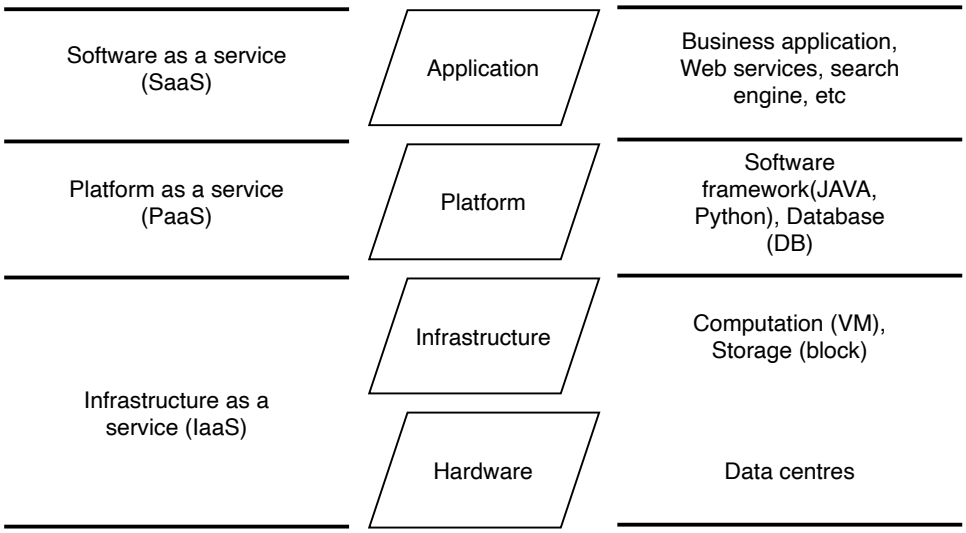


Figure 2.1: Cloud computing as a platform can be categorised into layers of operation. From the hardware level, such as the infrastructure, to the software level, such as the application layer, cloud computing offers customers solutions that suit all users.

also offers flexibility and a hassle-free experience of maintaining physical computers. The broker services can be categorised into three main groups.

- Infrastructure as a service (IaaS) is a cloud computing service that the cloud provider manages the stack of basic computing structures such as server, storage, network and physical infrastructure.
- Platform as a service (PaaS) has, in addition to the IaaS, management of middleware such as operating system.
- Software as a service (SaaS) is the most similar to running on-premise computing. The provider takes care of the whole computing service, including application and data.

Other services such as mobile as a service are debatable whether it can be a group of its own. However, if we are talking about the data centre cloud services, they mainly consist of the above three services. The services stack is shown in Figure 2.1.

The infrastructure as a service is the backbone of cloud computing services. IaaS is the physical aspects such as the building, data centre, mainframe, server and storage units. IaaS also covers the security and networking infrastructure necessary to operate a cloud. Essentially, this is where cloud services live.

The IaaS offers services such as:

Test and Development: Customers can efficiently manage the testing and development environments, with the ability to quickly power systems up or down. Infrastructure as a Service (IaaS) can significantly accelerate the application development and testing cycle, making it more cost-effective from a development standpoint.

Website Hosting: Customers can host their websites on IaaS platforms, which is generally less expensive than traditional web hosting services or operating their own servers.

Storage, Backup, and Recovery: IaaS provides scalable storage solutions that accommodate unpredictable demand. It also supports robust system backup and recovery processes in the event of system failures.

Web Apps: IaaS offers a comprehensive solution for web applications, encompassing storage, networking, and computing resources. Applications can be rapidly deployed and scaled according to demand.

High-Performance Computing: Supercomputers, essential for analysing vast datasets and solving complex problems, require significant resources and infrastructure. By running these workloads in the cloud, organisations can avoid the high costs associated with building and maintaining high-powered computing infrastructure.

Platform as a Service (PaaS) builds on top of Infrastructure as a Service (IaaS). It extends beyond hardware to include operating systems and database systems.

Development Framework: PaaS provides a development framework that developers can use to build or customize cloud-based applications. Much like using Excel macros, PaaS allows developers to create applications using built-in software components. Features such as scalability, high availability, and multi-tenancy are inherent, which minimizes the amount of coding developers need to do.

Analytics, Business Intelligence, and Big Data: PaaS supports analytics and data mining tools that can significantly enhance business decision-making processes.

Additional Services: PaaS enhances customer applications with additional services such as workflow management, directory services, security, and scheduling.

Lastly, *Software as a Service (SaaS)* represents the application layer hosted and run in the cloud. Examples include Google Apps, Salesforce, Dropbox, and Slack. SaaS offers the most complete cloud service model, albeit with less opportunity for customisation.

2.2 Cloud Market Space Complications

Cloud computing as a concept is a computing power renting scheme. The main idea is to have computers as a service, offload the computing workloads, or distribute the workloads to multiple computer units. However, choosing the right cloud instance is not easy, especially for inexperienced cloud customers. Several reasons contribute to the complexity of the cloud as follows.

Cloud instances and their problem of selecting the best cloud

The initial challenge for cloud customers is selecting the appropriate cloud instance specifications for their various applications. Customers have diverse requirements; for example, those needing high-performance CPUs require cloud providers to offer a range of options. These include different pricing schemes, performance levels, storage capacities, specialised hardware, and more. For traditional computers, the specification sheet typically serves as the primary indicator of performance. However, for cloud instances, the specifications listed may not always reflect the actual performance experienced by the user.

The performance level of computers is normally displayed through specification configuration. Normally, it is displayed using the main component of a computer. For example

- Central processing unit (CPU) type, model, speed, number of core.
- Random access memory (RAM) type, model, speed, protocol, size.
- Storage type, size, speed, vendor.
- Graphic processing unit (GPU) vendor, model, core counts, special hardware (CUDA).
- Network type, speed.

For example, Amazon EC2 T4g.nano has two vCPU (Custom built AWS Graviton2 Processor with 64-bit Arm cores), 0.5 GiB (No details), No GPU, network performance (up to 5 Gbps), Storage EBS (up to 2085 Mbps). This type of instance is not comparable to the m5.large with two vCPU (vCPU is a thread of either an Intel Xeon core or an AMD EPYC core), 8 GiB, network performance (Up to 10 Gbps), Storage EBS (Up to 4,750). Let us take the CPU as an example of the cloud specification display. It is used to be that the faster clock speed the processor has means higher performance. However, this is not always the case, given current cloud workloads. Additionally, there are differences in the instruction set, architecture type, cache size, number of cores etc. These specification labels are hard to quantify. Furthermore, there are instance types with multiple CPU models that are randomly given to customers which increasingly complicate

the situation (43; 122). Therefore, comparing computers' performance is performed using some form of numerical performance values, i.e. benchmarks (58).

Benchmarking in cloud computing merits a dedicated chapter due to its complexity and the detailed analysis it requires. Benchmark numbers, typically aggregating scores from various tasks performed by a computer or cloud instance, are vital for comparing one machine to another. Effective benchmarking requires controlling for pre-conditions such as the specific instance in use and the timing of the benchmark. Brokers with pre-purchased instances can thus perform thorough benchmarking and create a database for fair comparisons.

Cloud computing encompasses various elements, including the platform, software, and security features. Each provider attempts to distinguish its offerings to attract customers. This document, however, primarily focuses on general compute units, which allow customers to run any type of workload. These units are particularly suited to seasoned cloud users who require specialized cloud instances.

Additionally, the process of selecting cloud instances is complicated by **Performance Variation**, as documented by several studies (73; 5; 70). Performance variability can arise from many factors, including the time of day, the type of instance, and the current load, among others. Such variations, often uncontrollable by cloud customers despite guarantees from service level agreements, make even ostensibly identical instances perform differently.

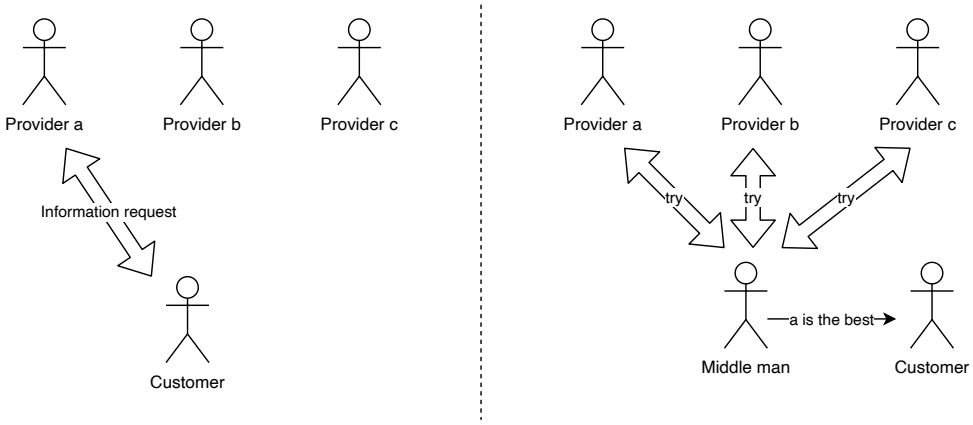


Figure 2.2: The figure shows the difference between the scenario with and without the middle man. Cloud instances are more complicated than typical computers. The instances are units of a computer that are segmented from a larger mainframe in a data centre. The specification page is a basic description of computer specifications that are not comparable across multi-cloud.

Choosing the correct cloud instance is not an easy task for customers. Therefore, a middle man "brokerage service" can help to streamline the selection process.

Cloud computing performance

The complexities of computer performance are extensive and beyond the scope of this discussion; however, we can distill performance in cloud computing and brokerage systems down to a few key parameters. Typically, the performance of cloud compute units is defined primarily by the number of virtual CPU cores and the amount of RAM available. Customers can later augment their configurations with additional storage and network speed enhancements.

In this work, we will concentrate on CPU and RAM, as these are central to cloud instance performance. It's important to note that CPU performance varies depending on the type of workload; some CPUs may perform excellently under certain conditions but poorly under others. This variability is common and a key reason why cloud providers often present their instance performance using generic CPU benchmarks, even though not all CPUs are created equal.

Nevertheless, it is widely accepted that CPUs can be effectively compared using aggregated benchmark scores. One critical component that is often lacking is a continuously updated database of cloud instance performance.

Cloud brokers, which manage multiple types and configurations of cloud instances, are well-positioned to maintain such a database. This resource allows them to directly compare the performance of different configurations. While this solves the problem for CPU performance comparability, RAM, storage, and network speed can be evaluated in a similar manner, ensuring comprehensive performance assessments.

2.3 Cloud Brokerage System

Cloud computing services have significantly evolved over the past few years, transitioning from simple off-site backups to comprehensive virtual computing systems (7; 112). With improvements in network speeds, customers can now offload entire computing workloads to the cloud. This growth has intensified competition among providers, driving innovation and diversification of offerings from general compute instances to specialized artificial intelligence-focused instances with pre-installed models. In such a competitive market, price is a primary motivator for customers, prompting providers to keep prices competitive. When price reductions reach their limits, providers often introduce a greater variety of options, such as different pricing schemes and value-added services. However, these choices can overwhelm, especially less experienced users, suggesting the potential benefit of a unified marketplace to simplify cloud adoption.

The role of a cloud broker, as defined by NIST, is to act as an intermediary, managing interactions between cloud users and providers (90). There are three main types of services a cloud broker

might offer:

1. **Aggregation**: Combines services from multiple providers into a single offering for users, handling all related complexities.

2. **Arbitrage**: Offers more flexible services tailored to specific user needs by selecting appropriate services from various providers based on usage analysis.

3. **Intermediation**: Adds valuable services to cloud offerings, including monitoring, anomaly detection, and enhanced security.

Cloud brokerage solutions are prevalent in both commercial and academic settings, each with distinct focuses. Commercial solutions aim to add value and facilitate the transition from local systems to the cloud, whereas academic solutions tend to emphasize technical frameworks and financial benefits for users without imposing additional charges.

Despite the availability of tools from providers to manage cloud instances effectively (57), recent surveys indicate persistent challenges in cloud computing that brokers can address (33). The overwhelming number of choices and the resulting complexity are key drivers for the emergence of cloud brokerage systems. These systems often cater to specific problems while also charging additional fees to simplify processes. For small customers, the flexibility of renting cloud instances from multiple providers involves extra work with limited benefits, unless they need to expand capabilities beyond what their current provider offers. Here, cloud brokers can provide a seamless service that connects multiple providers without being confined to just one.

Brokers also have the potential to facilitate special cloud services such as VM migration and data transfer, further enhancing their value to customers.

2.3.1 Broker's Challenges

The concept of a cloud broker can be as complex as the functionalities it is designed to offer. As a service, a broker must provide valuable functions to justify its existence. However, adding more functionalities can complicate the system and undermine its simplicity and optimization. This delicate balance between functionality and simplicity is crucial, especially given the inherent complexities of cloud computing (41). The focus, therefore, should be on simplifying existing cloud functionalities.

Cloud computing complexity stems from its components, which primarily include:

1. **Cloud Users**: These are the customers who utilize the broker services and are often the most

unpredictable and challenging component to manage.

2. **Cloud Providers**: These are the sources of the cloud instances for the broker and the originators of the cloud broker concept. Providers compete to attract more customers by offering better value for money (121). However, when costs begin to surpass revenue, providers may shift to a segmentation marketing strategy (31; 104), offering more tailored choices to meet specific needs. This can lead to premium pricing for specialised instances and possible penalties or inconsistent performance for lower-tier options (73).

3. **Cloud Brokerage System**: This refers to the mechanisms of the broker system itself.

Cloud providers were initially driven by competition, striving to attract more users by enhancing value. However, as strategies pivot towards market segmentation, providers not only offer more diversity in their services but also introduce complexity into the market. This complexity can dilute one of the key advantages of cloud computing—its simplicity and flexibility—forcing users to spend additional time optimizing their workflows and adapting to a constantly evolving cloud environment.

2.3.2 Commercial Cloud Brokers

The market currently features a wide array of cloud service brokerages. These commercial solutions typically charge an additional fee for their services, with pricing often based on a subscription model. This model allows for lower initial costs, but requires customers to continue paying for as long as they utilize the service. While most vendors do not publicly disclose their pricing, customers and businesses interested in these services can contact them directly to request a quote.

Typical features of cloud brokerage solutions on the market include:

- Marketplace services
- Cloud engine services or a multi-cloud solution
- Integration services
- Security services

A marketplace service is a middle man that provides an exchange of cloud services between customers and providers. The service is similar to a stock market where the transaction is done through a middle man (market) rather than directly with companies. This type of broker allows for the convenience of exchanging cloud instances for a fee.

If the customers want to distribute their application through multiple providers, an engine that manages cloud virtual machines across multiple providers or a multi-cloud solution is the type of broker that provides such a service. The multi-cloud allows customers to connect and utilise multiple providers or even multiple geo-locations. The customers can use the unique strength of each provider in their applications. Typically, a multi-cloud solution is possible without the broker intervention. However, as with most broker services, the usage of a broker can provide the customer with a better experience. Similarly, multi-instance types of service also help customers enhance their application performance.

Next, one of the major concerns when using the cloud is security. Cloud users are sharing server space with multiple users in the same data centre. Thus, the security of and safety of the users' data is critical. Thus, there is a broker service that guarantees users' data safety with an additional security layer.

2.3.3 Cloud Brokerage System Research

The commercial space is not the only active solution. In the academic space, several ongoing pieces of research are conducted on all aspects of the cloud broker. According to (9), there are four following aspects of cloud computing research.

1. Cloud service broker for performance
2. Cloud service broker for application migration
3. Cloud broker model
4. Data for cloud broker

Cloud service broker for performance. There are cloud brokers specifically focused on optimizing cloud instance performance. These services encompass a range of tools including benchmarks, resource matching, performance indicators, comparison, and prediction. Notable frameworks such as STRATOS and OPTIMIS are designed to capture performance and match critical performance indicators (CPIs) effectively (92; 39). The CPIs encompass seven categories: accountability, agility, assurance, financial, performance, security, privacy, and usability. These frameworks aim to address the common challenge of aligning customer requirements with the appropriate cloud instances efficiently.

CloudBench is another benchmarking framework that allows users to specify the importance of memory, processor, computation, and storage for their applications through a set of four weights (101). These weights create a profile of the customer's application needs, which, combined with

cloud benchmark data, helps generate a ranking of virtual machines (VMs) that could optimize application performance.

Moreover, traditional computer benchmarking tools such as SpecInt and PassMark are also applicable to cloud environments (52; 56). These benchmarks provide a foundational metric for comparing different cloud instances.

Cloud service broker for application migration. Application migration, if done efficiently, is a great strength of cloud computing. Since cloud computing can be viewed as elastic, migrating an application from one instance to another can help the application achieve higher performance when moving to suitable cloud instances. *CloudProphet* and *RightScale* is working on a similar model of application migration to a more suitable cloud instance (74; 12). *CloudProphet* uses prediction from a local running workload to identify the best cloud instance for application migration. This saves time on the real cloud and offloads that to the local machine. The *RightScale*, on the other hand, use pre-set criteria which trigger an action when the application needs a better cloud instance.

Overall, cloud migration is still a costly process that needs a workaround to bypass a time-consuming data transfer process.

Cloud broker model The cloud broker model is a crucial area of focus in cloud computing research, serving as a pivotal point where performance and migration concerns intersect. This model is central to discussions in the cloud broker domain.

As previously discussed, frameworks such as STRATOS and OPTIMIS exemplify broker models aimed at performance matching. These systems require comprehensive benchmarking and performance profiling to function effectively. The data gathered through these processes inform the model's decisions, helping it to identify the optimal configuration for achieving its objectives.

Similarly, *SMICloud* adopts an approach that prioritizes the Quality of Service (QoS) of cloud instances as a key decision-making indicator (44). It ranks instances based on QoS parameters like response time, sustainability, data center performance per energy (including carbon emissions), and other factors such as suitability, accuracy, and interoperability, among others. SMICloud introduces an Analytical Hierarchical Process (AHP)-based ranking mechanism to align user-defined QoS requirements with cloud provider offerings, presenting a multi-criteria methodology that focuses on selecting instances for optimal performance (110).

There are also cloud broker models specifically designed to maximize profit. Research such as that by the authors of (84) investigates the feasibility and profitability of a reselling broker model. In this model, a broker rents reserved instances and resells them to customers, prioritizing

Table 2.1: An example of the specification page on the Amazon EC2 cloud provider.

Instance size	vCPU	Memory (GiB)	Instance storage (GiB)	Network bandwidth (Gbps)	EBS bandwidth (Mbps)
c6gd.medium	1	2	1 x 59 NVMe SSD	Up to 10	Up to 4,750
c6gd.large	2	4	1 x 118 NVMe SSD	Up to 10	Up to 4,750

Each vCPU is a thread of either an Intel Xeon core or an AMD EPYC core, except for C6g, C6gn, T2 and m3.medium.
Each vCPU on C6g and C6gn instances is an AWS Graviton2 core processor.

higher-demand queries to maximize the utilization of reserved instances and increase profitability. This approach highlights a strategic method to enhance profit margins within the cloud brokerage industry.

Data for cloud broker. Cloud broker, as with other platforms, works with data. Let us assume that the broker algorithm uses the running frequency of the CPU as the performance indicator. Then, this data will become the deciding factor of the outcome of the algorithm. Thus, obtaining accurate data is one of the focuses of cloud brokerage research.

Data for cloud brokers can be obtained through two options; providers listed data and third-party extracted data. The providers' data are publicly available. Cloud instance specification data can be obtained directly from the providers' websites. However, Cloud providers list performance metrics are *not* standardised between different providers. Table 2.1 shows an example of the cloud instance specification page from Amazon EC2 cloud (27). The example shows one of the potential problems with the cloud specification page. For example, the CPU units are not consistently displayed. Therefore, the performance value is also not uniform across all instances.

Public data is also a place where the broker can obtain other data types that are not associated with the performance. The data include geo-location of the infrastructure, service level agreements, pricing scheme etc. Most of this data is straightforward and can be taken as a reference.

Third-party extracted data represents another critical type of cloud data, primarily utilized due to inconsistencies in cloud specifications. To enable fair comparisons, cloud customers often need to meticulously examine cloud specifications and consult reliable benchmark databases. To facilitate this, services like the *Yahoo Cloud Serving Benchmark* and *CloudScore* offer databases filled with benchmark scores, allowing customers to directly compare cloud instances using consistent metrics (26; 32). Similarly, cloud brokerage systems can employ these methods to conduct value assessments of cloud instances, ensuring comparisons are made on an equitable

basis.

Cloud brokerage remains a dynamic field of research, ripe with opportunities ranging from performance enhancement to system optimization. The field continues to evolve, incorporating sophisticated techniques to better serve user needs and improve cloud service delivery.

Next, we will delve into time series data, a fundamental data type extensively used across various scientific and technical fields including finance, machine learning, and beyond. Time series data is crucial for analyzing trends over time, forecasting future patterns, and making data-driven decisions in real-time environments. This data type is particularly important in cloud computing for monitoring system performance, predicting load and usage, and optimizing resource allocation dynamically.

2.4 Time Series

As previously mentioned, the cloud brokerage system operates on a time scale. Hence, individual data points in the system can be transformed into time series data for better management and maintainability.

A time series data is a sequence of indexed data points ([117]). The data has two components (domains), the frequency domain and the time domain. Time series data is usually used for displaying one-way indexing data, hence the usage of time. Numerous applications use time series data, such as stock market price movements, daily temperature levels, compute resource usage during the day, etc.

Time series data is characterized by the frequency of its measurements, typically categorized into discrete or continuous forms. Discrete time series consist of data points collected at consistent intervals, representing sequential measurements in equal time spaces. This type of time series is commonly found in practical data collection, where measurements are taken at fixed and regular intervals, such as daily temperature readings, monthly sales data, or yearly financial reports.

In contrast, continuous time series are more theoretical and often used in model representations rather than as direct data measurements. They assume data points can be observed at any instant within a given range, making them a useful conceptual tool in mathematical modeling and simulations. Continuous time series are used to create models that predict or simulate smoother processes, where data points might be interpolated to infer values at times not explicitly measured.

Both types of time series have critical applications in various fields, leveraging their unique

properties to analyze and predict trends and patterns over time effectively.

Next, the **Notation** of time series in this work and in general is accepted as shown below.

A common notation for a discrete time series can be represented as

$$X = \{x_1, x_2, ...\}$$

or

$$X = \{x_t : t \in T\},$$

where T is the time index set. Either notation is accepted and widely used.

X or $X(t)$ denotes a time series and x_i denotes individual value of each time step i. i can be seen as the "time" of measurement or observation. It can be indexed as both real-time or just the ordered time step.

The time series by itself is a display of data. An additional analytical works need to be performed get the following information:

- Obtain an understanding of the underlying structure that produced the observed data.
- Fit a model and proceed to forecasting, monitoring or even feedback and feed-forward control.

For example, the stock market index Dow Jones Industrial Average (DJIA) is shown in Figure 2.3. DJIA can be observed as a time series and then plot as values against the time graph. We can see an accumulative index (value) movement from one time step to another (day) from the graph. The green line on the graph is an example of a time series analysis result. The green line is predicted using long-term-short-term memory artificial neural network prediction of the index, then plot over the original data. The green line can be used as a prediction such that traders can buy and sell the DJIA according to the value predicted.

A time series has a property to measure "randomness". This is in the form of stationary or non-stationary. It is defined such that the non-stationary is unpredictable and cannot be a forecast or model after. This means that each movement in a non-stationary time series is random. On the other hand, being stationary is a property of time series that does not depend on time. The time of observation should not matter in a stationary time series. The time series should look similar at all observation points. In other words, time series with the trend and seasonal variation are not stationary.

The type of data that gives problems to the predictors and users are, of course, unpredictable. One of the models that try to capture the non-stationary property is the random walk model.

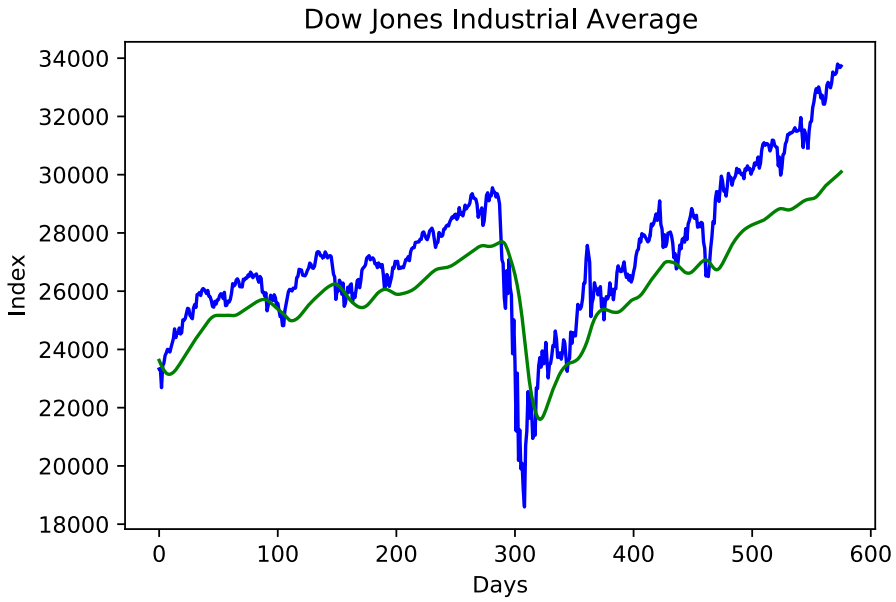


Figure 2.3: The blue line is Dow Jones Industrial Average (DJIA) movements. The index movement behaviour follows a random walk pattern. The green line shows a rolling prediction of the DJIA using long-term-short-term memory artificial neural network. The relatively simple up and down movement of the index proved to be difficult to predict.

Random walk is vital in trading in a stock market or prediction of human behaviour. Since most of the theories and assumptions about random walk theory are about how unpredictable random walk is, it is productive to transform the random walk model, in this case, a non-stationary model, into a usable one.

Non-stationary time series data often lack drift or deterministic trends, which can complicate statistical analysis due to violations of stationarity assumptions. One common technique to address this issue is differencing, which helps to stabilize the mean of the time series by removing changes in the level of a time series, thus potentially eliminating trend and seasonality.

The differenced series is calculated as the change between consecutive observations in the original series. This can be mathematically represented as:

$$\hat{y}_t = y_t - y_{t-1}$$

where $\hat{y}_t$ is the value of the differenced series at time t, y_t is the value of the original series at time t, and y_{t-1} is the value of the original series at time $t-1$.

This transformation results in a differenced series having only $T-1$ values if the original series contained T observations. This reduction in the number of data points occurs because the first

observation in the original series does not have a preceding value from which to subtract and thus is not included in the differenced series.

When the differenced series is white noise, the model for the original series can be written as

$$y_t - y_{t-1} = \varepsilon_t$$

where ε_t denotes white noise. Note that, this equation is the same as the *random walk* model.

$$y_t = y_{t-1} + \varepsilon_t$$

Random walk models are characterised by long periods of trending upwards or downwards, punctuated by sharp changes in direction. In forecasting, the random walk model is often used to create naive forecasts, where the forecast for the next period is simply the last observed value. This approach is based on the premise that future movements are unpredictable and equally likely to increase or decrease, as follows:

$$y_t = y_{t-1} + \varepsilon_t$$

where ε_t represents the random error at time t.

However, if a random walk model incorporates a drift component, the model is modified to include a constant term, c, representing the mean change between observations:

$$y_t - y_{t-1} = c + \varepsilon_t$$

This can be rearranged to show that:

$$y_t = y_{t-1} + c + \varepsilon_t$$

In this model, c signifies the average incremental change. If c is positive, the series tends to drift upwards over time, indicating a general increase in y_t. Conversely, a negative c indicates a downward trend.

The concept of drift is essential, especially when considering longer observation periods, T. This introduces an element of non-stationarity to the series, which can be addressed using differencing techniques. Differencing helps in stabilizing the mean by removing changes in the level of the series, making the time series stationary:

$$\Delta y_t = y_t - y_{t-1}$$

Time series analysis, including techniques such as differencing, is a vast area of study. This chapter introduces only a fraction of the broader field, focusing primarily on methods to manage and analyse non-stationary data.

2.4.1 Time Series Analysis

Time series analysis for all intended purposes is about understanding the nature of the time series data. Since the time series often associate with financial values, a good understanding of the data is always welcoming for profitability. The methods of achieving the said task, however, are varied depending on the purpose.

Curve Fitting is a statistical method used to find the function that *best fits* a set of data points by minimizing the difference between the observed values and the values predicted by the model. This process, described in (51), is essential for understanding the relationship between two variables, which can be visualized through a scatter plot.

Depending on the distribution of data, this relationship may appear linear, exponential, quadratic, or follow another mathematical form. The relationship is considered strong if the actual data points closely adhere to the predicted curve.

In a **linear relationship**, a change in one variable results in a proportional change in another, represented by the equation:

$$y = mx + b$$

where y is the dependent variable, x is the independent variable, m represents the slope, and b the y-intercept. Here, the influence of the independent variable x on y remains constant across the data.

Conversely, in a **non-linear relationship**, the impact of the independent variable on the dependent variable varies across the data set. For example, a quadratic relationship might be expressed as:

$$y = ax^2 + bx + c$$

where a, b, and c are coefficients that define the curvature of the graph, indicating that the effect of x on y changes with different values of x.

Function approximation is a technique for estimating an unknown underlying function using historical or available observations data (109; 60). The term mainly associates with machine learning techniques such as an artificial neural network. A mathematical function governs the mapping of input and output data, called the mapping function, and it is this function that an

artificial neural network tries to estimate.

Prediction and forecasting is a method of using the curve fitting or function model to estimate future values (119). A forecast refers to a calculation or an estimation that uses historical data combined with recent trends to estimate a future outcome. On the other hand, a prediction is an actual act of indicating that something will happen in the future with or without prior information.

In a time series, future values estimation is usually referred to as forecast values.

2.4.2 Time Series Model

Models for time series data can have many forms and represent different stochastic processes. It is essentially a model fit to a given value of data. The main idea of using a time series model is to identify the behaviour of the data in a time domain. The most popular model is the autoregressive integrated moving average (ARIMA), and its extensions (54).

Although, there are numerous variations of the ARIMA model, all of which improve upon the original on certain aspects. This section will focus on the original model since most of the core ideas are similar.

A time series model can be simply divided into three classes. First, is the autoregressive (AR) which is the variable of interest determined using a linear combination of past values of the variable. It is essentially a regression of the variable against itself.

$$x_t = c + \sum_{i=1}^{p} k_i x_{t-i} + \varepsilon_t$$

where c is a constant, k_is are parameters of the model and ε_t is white noise.

The next class is the integrated (I) indicates that the data values have been replaced with the difference between their values and the previous values.

Lastly, the moving average (MA) specifies that the output variable depends linearly on the current and past values of a stochastic term.

$$x_t = \mu + \varepsilon_t + k_1 \varepsilon_{t-1} + \ldots + k_q \varepsilon_{t-q}$$

where μ is the mean of the time series, $k_1, ..., k_q$ are the parameters of the model, and εs are the white noise.

Together, they form a time series model call autoregressive integrated moving average (ARIMA)

models.

The ARIMA model is widely used and accepted by the community to have all of the necessary components to predict future time series values. However, it too has a problem finding the optimal parameters, which means that each prediction result can be varied depending on the parameters configuration. Auto-ARIMA is a method of build in an ARIMA model by searching for the best combination of parameters p, q, d. Thus, it should yield the best performing prediction.

2.4.3 Feature Engineering

The processing of time series data, similar to other data types, requires careful preparation. Although defined variously by different experts, the commonly accepted steps for machine learning data processing include:

1. **Gathering Data:** Collecting relevant data from various sources, which may involve pulling from databases, APIs, or sensors.

2. **Cleaning Data:** Removing inaccuracies and inconsistencies such as outliers, missing values, and duplicate entries to improve data quality.

3. **Feature Engineering:** Transforming raw data into features that better represent the underlying problem to the predictive models, enhancing model accuracy and performance.

4. **Defining a Model:** Choosing a suitable model based on the nature of the problem and the data. This step involves selecting the algorithm that best fits the needs of the prediction task.

5. **Training, Testing, and Prediction:** Training the model on a designated training set, evaluating it on a testing set to gauge its performance, and using the model to make predictions or decisions based on new data.

Feature engineering is the process of obtaining information from raw data. Typically, feature engineering involves human inputs and trial and error until the best-fit features are found. Constructing a feature set is very time-consuming, particularly when the information is complex, i.e. multiple tables. There are two main categories of feature engineering: transformations and aggregations. Time series as present in the cloud broker can be considered as raw data with indexed value features (50). Features of the data are measurable properties of the observed data. The feature extraction in a time series is a “characteristics” capturing of the data. Basic characteristics are used to train a model of interest. Feature extraction is similar to feature engineering. It involves the concept of reducing data characteristics to describe the data. A complex data can be difficult to describe and illustrate; thus, feature extraction is used to compute,

identify, and select a suitable set of *features*. The feature's extraction process is tedious and time-consuming. Therefore, an automated time series feature extraction is useful in helping training ML algorithms and testing a scale-up hypothesis in the form of a curated package, *TSFRESH* (24). The meaningful features are extracted from a raw time series using a combination of features generating functions. These features are ready to be used in any ML algorithms.

Data features do not only reduce the computation time of the system. In many cases, using the right features set can also lead to a better result of the system (64).

2.5 Gaussian Process

A random process is also known as a stochastic process. It is an infinite collection of random variables $\{X(t) : t \in T\}$, where T is the probability space. Usually, the index t is time. However, the index could be any spatial dimension. Some might consider the random process to be a time-varying function.

There are two possible interpretations for a random process. For a random process $X(t,\omega)$ of two variables $t \in T$ and $\omega \in \Omega$. Ω is a sample probability space, and T is a discreet or continuous set.

- for fixed ω: $X(t,\omega)$ is a deterministic function of t.
- for fixed t: $X(t,\omega)$ is a random variable.

For most cases, we are interested in the fixed t while $X(t,\omega)$ is a random variable. This model is also called a random walk model.

2.5.1 Gaussian Process for Time Series Modelling

We can assume that a time series can be format into a regression problem of the form $y = f(x) + n$, in which f is an unknown function, and n is an additive noise process. There are two goals for the problems: evaluating the function f and the probability distribution of y for some x.

The observable data pairs are also assumed to be present, i.e. (x_t, y_t). Additionally, the observations are accurate. To approximate the function f, there are mainly two approaches, curve fitting and function mapping. Curve fitting assumes that y is ordered by x. In a time series, the datum x is a time variable. With inference, the process fit a curve to the set of x, y points. The relationship between x and y is not fixed but conditioned on observed data. Prediction results are the extrapolation of the curve that models the observed past data. The function mapping approach considers inference of a function f, which maps some observed x to an outcome variable y.

The Gaussian process is an inference about functions.

A mean vector and covariance matrix define Gaussian distribution. The covariance K must be positive definite, i.e. for any vector x, $x^T Kx \geq 0$. Therefore, K is symmetric with a positive diagonal, and all eigenvalues of K is greater than 0.

The (i, j) element of the covariance expresses how variable i is dependent upon variable j.

A Gaussian process is the generalisation of a multivariate Gaussian distribution to a potentially infinite number of variables.

Predictant $y_* = y(x_*)$

Data $y_d = y(x_d)$

$$p(y_*, y_d) = N\left(\begin{pmatrix} y_* \\ y_d \end{pmatrix}; \quad \begin{pmatrix} \mu(x_*) \\ \mu(x_d) \end{pmatrix}, \quad \begin{pmatrix} K(x_*, x_*) & K(x_*, x_d) \\ K(x_d, x_*) & K(x_d, x_d) \end{pmatrix}\right)$$

Mean $m(y_*|y_d) = \mu(x_*) + K(x_*, x_d)K(x_d, x_d)^{-1}(y_d - \mu(x_d))$

Covariance $C(y_*|y_d) = K(x_*|x_*) + K(x_*|x_d)K(x_d|x_d)^{-1}K(x_d|x_*)$

The covariance matrix ensures that close together values in the input space produce output values close together. This covariance matrix, along with a mean function to output the expected value of function f, defines a Gaussian Process.

2.6 Causal Analysis and Additive Noise Model

Finding a correlation between variables is standard practice in scientific experiments. It is a process of finding a connection between groups of observable data to understand the behaviour of the system. The relations are helpful for building a model that would support the assumption of the experiment. However, the correlation relation does not imply causal relation. The causal relation is essential in performing good science. An unintentional correlation might punish a critical operation; thus, relationship finding is of importance.

Causal analysis is the study of establishing cause and effect. It usually involves collecting data (observable data) or run multiple experiments then perform the analysis. The causal inference requires different techniques than just function mapping or data fitting. It requires additional assumptions to produce a good causal result. The most important question that the causal inference has to answer is that the correlation is not causation.

The causal discovery was studied to find causal relations by analysing the observable data. The data are produced by underlying causal and sampling data. Practically, all science experiment is about confirming hypotheses which are finding the causes and effects. The causes are experimentally identified by adjusting parameters and repeatedly executing the experiment until the conclusion is drawn. However, these methods are expensive, time-consuming and in some cases impossible to redo. Thus, inferring causal relation from the existing data can are interested in many scientists and researchers (45).

Causality in a time series is a difficult task to achieve due to multiple reasons. Time series data are generated dynamically. The generation process might be non-linear, the observation rate might be slower than the generation rate. Thus, the accuracy of causality on a time series are sensitive to many factors mentioned above.

Several methods deal with time series data: We can partition the data into "windows" of corresponding disjointed data. Then we can use each window as a data unit. We can estimate a lagged unit to be the data analysis unit. This is similar to a popular time series analysis autoregression. The causality of the time series using the autoregression is called "Granger causality". We can omit the time component of the time series and treat the data as time-independent. Between all causality approaches, they possess both strengths and weaknesses depending on the usage.

Causal inference methods with the Additive Noise Model

The basic idea of causal inference starts from two observable data vectors. Thus, given two vectors containing observed data points, namely X and Y, a causal discovery method can find if one of the vectors is a cause of another. In this work, we use a causal discovery method called the additive noise model (55).

The additive noise model (ANM) is one of the most popular methods used in causal inference. The basic premise of a causal relationship between two variables, X and Y, is that changes in X predict changes in Y or vice versa. The association between the parameters is defined through generative functions, f and g, such that:

$$Y = f(X) + n_{y,f}$$

$$X = g(Y) + n_{x,g}$$

where $n_{y,f}$ and $n_{x,g}$ are the residuals or noise components. Fitting Y as a function of $f(X)$ gives the residual $n_{y,f} = Y - f(X)$, and fitting X as a function of $g(Y)$ gives the residual $n_{x,g} = X - g(Y)$.

The direction of causality is determined by the independence of the residuals:

- If $n_{y,f}$ is independent of X and $n_{x,g}$ is dependent on Y, then $X \to Y$.
- If $n_{x,g}$ is independent of Y and $n_{y,f}$ is dependent on X, then $Y \to X$.
- Otherwise, the causal relationship remains inconclusive.

It has been asserted that the method of fitting functions and the criteria for independence should not compromise the results of causal inference. However, studies such as those found in (69) show that the performance of ANM is influenced by factors such as the tails of the additive noise and the sophistication of the regression algorithms used. These factors do not alter the causal relationships but do affect the rate at which causal relations can be correctly identified.

Moreover, ANM involves complex regression tasks and computationally intensive tests for independence, making scaling to systems with a large number of variables problematic. A novel framework, *KIKO*, has been developed to address these challenges by using a simpler, one-way regression and a rapid independence test based on the Hilbert-Schmidt Independence Criterion (HSIC), as detailed in (8). This approach significantly reduces the computational demands of ANM, facilitating faster and more efficient causal analysis.

Additionally, the indirect additive noise model between X_1, X_2, X_3 is investigated by the authors of (15). The model of $X_1 \to X_3$ is first established. The additive noise model has been demonstrated to be effective, but the model is incomplete—indirect causal inferences that might influence the model causes by intermediate variables. A cascade model estimates the model, including the unmeasured intermediate variables. A causal model, $X_1 \to X_2 \to X_3$, can be inferred even when the X_2 is unmeasured. This is similar to our system of a broker, which has intermediate variables of inventory composition.

There are other causal discovery techniques such as bi-variate fit, which gives the state of the model whether or not it is the causal model (80). Conditional Distribution Similarity Statistic re-scale the values of effect in respect to the cause (42). For this method, if the standard deviation is low, then there is a likely chance that the parameters might be causal pairs. There are many more studies on the causal discovery, all of which aim at identifying or confirming the relationship between interested parameters. The relations can help researchers understand the mechanism behind the system and ultimately can regulate the system optimally.

Since, the causal discovery methods rely on the correlatable independency (55; 42; 46). In this work, we use the *Hilbert Schmidt independence Criterion* (HSIC) as the independence test (47). HSIC is calculated as follow: let x and y be two sample data of length n. Gram matrices K and L

are defined as:

$$K_{i,j} = exp((x_i - x_j)^2/sig^2)$$

$$L_{i,j} = exp((y_i - y_j)^2/sig^2)$$

and matrix H is defined by

$$H_{i,j} = \delta_{i,j} - \frac{1}{n}$$

Let $A = HKH$ and $B = HLH$ then

$$HSIC(x,y) = Tr(AB)/n^2$$

where *sig* is the Gaussian kernel width (normally 1).

After the test, three possible conclusions can be drawn from vectors of x_i and y_i data:

- *Independent*: in this case, the causal relation is not conclusive, as there is no casual conclusion that can be drawn from the data.
- *Mutually dependent*: x and y correlate, but it is insufficient to conclude any causal relation. In this case, the causal relation from both directions of x and y will be considered as correct.
- *Conclusive*: The causal direction from x to y can be concluded.

Only *conclusive* is accepted as a causal pair.

With the additive noise model, we can build a causal model based on identified causal pairs. Once the relations are established, they can be represented with a *Directed Acyclic Graph* (DAG). The cause parameter is partially responsible for the change in effect parameter (14).

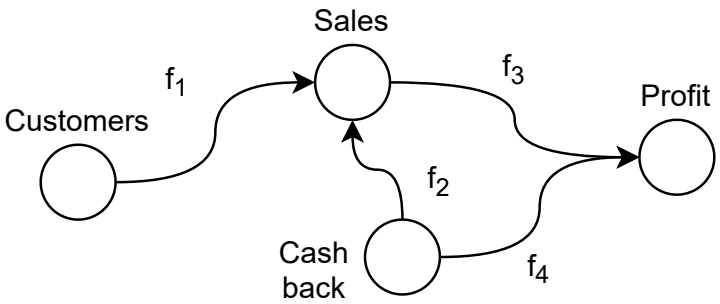


Figure 2.4: A Directed Acyclic Graph (DAG) showing the causal relations between the variables: $Customers$, $Sales$, $Cashback$, $Profit$. The behaviour between variables follows the function f_i. From the example, the $Profit$ adhere both the $Sales$ and $Cashback$ with f_3 and f_4 respectively. In other words, the $Customers$ and $Cashback$ cause $Sales$; and $Sales$ and $Cashback$ cause $Profit$.

Figure 2.4 shows a DAG of sales-related parameters in an arbitrary shop. A DAG has two components: vertices and edges. The vertices are the data identity (name). In this case, a set of already identified causal pairs. The edges are the causal links between the paired variables. Typically, the vertices do not carry any numerical values. On the other hand, the edge can be assigned correlation functions, f_i. The $\rightarrow$ indicates cause and effect parameters. Parameters are nested in a parent function. For example, if $Profit$ depends on $Sales$ and $Cashback$, which annotates as

$$Profit|Sales, Cashback$$

then

$$x_{PA}^{Profit} = Sales, Cashback$$

Thus, if the aim is the $Profit$ then

$$Profit = F(x_{PA}^{Profit})$$

where function F is a result of functions f_3 and f_4. The DAG is typically used to illustrate and analyse the system behaviour. A system admin should pay close attention to the relationship between all the identified variables in this case.

2.6.1 Discussion

From our review, it is evident that a cloud broker's inventory might include multiple types of instances, ranging from long-term instances valued for their cost-effectiveness to short-term instances prized for their flexibility. Balancing these varying instance types can be addressed through a combination of prediction, regression, and optimisation techniques. Individually, these methods are effective at resolving specific challenges; however, integrating them into a cohesive strategy is more complex.

For instance, predictive algorithms can forecast future demand for cloud instances. Yet, such predictions alone are insufficient for structuring a broker's inventory. Consider a scenario where customer demand for a specific type of instance is the prediction target. Relying solely on this data, without additional context, would restrict inventory management to a simplistic one-to-one model—where each customer order directly corresponds to a single reserved instance. Moreover, while customer demand may fluctuate both upwards and downwards, the number of cloud instances (especially reserved ones) is often less flexible.

Moreover, causal discovery techniques are instrumental in identifying cause-and-effect relationships within the data. However, knowing these relationships alone does not suffice for

constructing a profitable broker model. The crucial element here is a suitable profit model that the broker can adopt to decide the optimal inventory composition. Here, the Gaussian process utilised in additive noise models becomes valuable as it helps estimate the functions connecting data points. With these functions and optimal data, we can develop an effective profit model based on actual observations.

It becomes clear that no single solution exists for solving the complexities of cloud inventory optimisation. There is a technological gap that this work aims to bridge by applying these theories and methods to the specific challenges of cloud brokerage inventory management.

2.7 Summary

Cloud brokerage system research requires theoretical background materials from many areas. It is a very diverse topic that requires a wide variety of expertise. In this chapter, we have introduced some of the high-level concepts involved in a cloud brokerage system. We start from cloud computing to the time series data analysis used in the brokerage system. In the next section, we will detail the theories and technologies in our brokerage system.

CHAPTER THREE

BACKGROUND

In this section, the detailed background of theories and technologies used in this work are presented. First, a profitable cloud brokerage model from reselling the reserved instances inventory is explained. It shows the feasibility of the broker model. Next, the time series forecasting model and the Gaussian process are explained as part of the main components of the auto profit model generator in this work. Lastly, the causal inference with the additive noise model is also explained in detail.

3.1 Profit Maximising Cloud Brokerage Systems

In this section, a mathematical analysis of the profit maximising cloud broker is presented according to ([84]). Profit maximising brokerage leverages the price difference between two pricing schemes of on-demand and reserved to earn a profit. The brokerage method attracts customers by leveraging the lower cost of the reserved instances and adjusting the selling price of instances to the cloud customers.

For a given user distribution data, it is possible to find the optimal cloud inventory configuration. In this example, the $M/M/n/n$ queuing model is used as the requests intake for the broker ([103]). The $M/M/n/n$ queuing model assumes all arrival user queries to be Poisson with an arrival rate of λ (number of queries per time unit). The λ is affected by two factors, total demand (λ_{max}) and the resource reselling price.

For a broker with only **reserved instances** in its inventory, the linear price demand is calculated as:

$$p = p_n + (p_{od} - p_{re})\frac{\beta - \beta_{re}}{\beta_{od} - \beta_{re}} \tag{3.1}$$

where β is the price and p is the market share ratio. In general, the lower sell price β from the on-demand price β_{on} attracts more customers. This relation is crucial as the profit of the broker comes from the differential of price between β and β_{on}.

Next, the queue length for the multi-server system is defined to be n. Then, the average service rate is denoted as μ. Lastly, the server utilisation is $\rho = \lambda/n\mu$. In the queuing system, π_k is the probability of k, $k < n$ service requests in the queuing system. In this system, the queue length and the maximum number of requests is the same number n.

We have

$$\pi_k = \pi_0 \frac{1}{k!} \left(\frac{\lambda}{\mu}\right)^k, k = 1, 2, ..., n$$

where

$$\pi_0 = \left[\left(\sum_{k=0}^{n} \frac{\lambda}{\mu} \right)^k \right]^{-1}$$

Since the broker only employs the reserved instance, there also has to be some customer requests that do not get processed.

$$P_L = \pi_n = \left[\left(\sum_{k=0}^{n} \frac{\lambda}{\mu} \right)^k \right]^{-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^k,$$

number of loss customers in unit time is

$$\lambda_L = \lambda P_L = \lambda \left[\left(\sum_{k=0}^{n} \frac{\lambda}{\mu} \right)^k \right]^{-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^k$$

This brokerage system can denied services to some of the requests in order to maximise the profit.

The profit of a broker comes from the revenue and cost. The cost per unit of time for the cloud broker is

$$C = n\beta_r e$$

Essentially, this is the size of the broker inventory, i.e. number of the reserved instance being deployed.

The revenue of the broker is calculated directly from the customer demand distribution together with the price demand from equation 3.1.

Assuming that the U is the unit of time, β is the sell price and average execution time is $\bar{t}$ then E is the expected charge for a service request.

$$E = U\beta \frac{1}{1 - e^{U/\bar{t}}}$$

Now we can assemble the profit

$$Pro = R - C \tag{3.2}$$
$$= \lambda(1 - P_L)E - n\beta re \tag{3.3}$$

From equation 3.2, we can see that two parameters can be used to optimise the profit, the sale price β and the size of cloud inventory n.

Optimal price for maximised profit, $\max Pro(\beta)$

$$\frac{\partial Pro}{\partial \beta} = 0$$

$$\frac{\partial Pro}{\partial \beta} = \lambda(1 - P_L)T + \beta T \frac{\partial \lambda}{\partial \beta}\left[(1 - P_L) - P_L n(1 - \rho)\right]$$

Optimal size for maximised profit, $\max Pro(n)$

$$\frac{\partial Pro}{\partial n} = 0$$

$$\frac{\partial Pro}{\partial n} = \lambda \beta T P_L \left(1 + \frac{1}{2n} - ln(e\rho)\right) - \beta_{re}.$$

With the analysis above, we can, in theory, find the optimal size and price of the cloud inventory for a given customer data as shown in Figure 3.1. The limitations are the unplaced requests and the variety of cloud instances. The unplaced requests are either discarded or re-queue. Additionally, the deployment model is restricted to a one to one deployment. Nevertheless, theoretically, finding the optimal parameters of a reserved instance inventory broker system is possible.

3.2 Time Series Forecasting Models

In this work, the time series model is used widely for data analysis purposes. One of the most used models is the Auto-Regressive Integrated Moving Average (**ARIMA**) model. Therefore, it

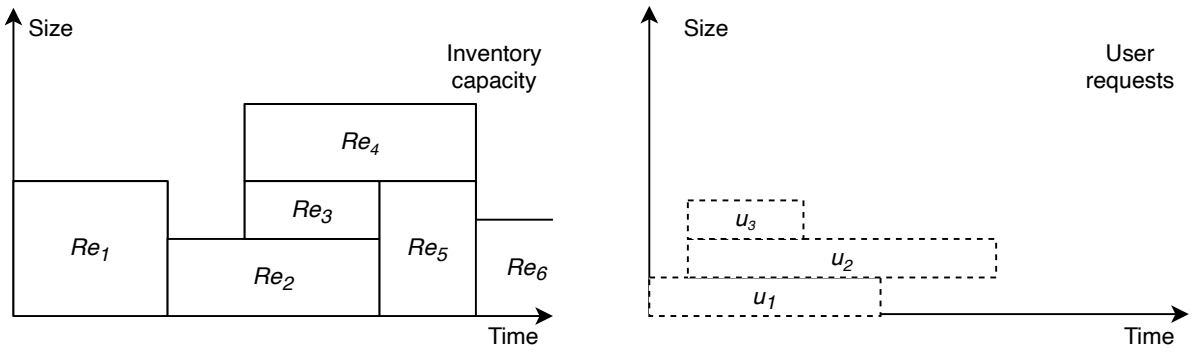


Figure 3.1: (Left) The graph shows the structure of a broker inventory with number reserved instances. (Right) The graph shows the user request slots.

is worth mentioning the details of the model in this section.

Assume that a time series observation data is represent in a series of $y_1, y_2, y_3, ..., y_n$. A forecast y_{t+h} is based on the prior data points up to time step t.

ARIMA models

Although, an ARIMA model is one of the most well-known time series forecasting methods. It also suffers from the common problem of the order selection process. Nevertheless, first, we need to define the ARIMA model. The standard and basic theory were provided in Chapter 2. In this chapter, we will go into more detail about the model.

A non-seasonal ARIMA model, denoted as ARIMA(p,d,q), is represented by the equation:

$$\phi(B)(1-B^d)y_t = c + \theta(B)\varepsilon_t \tag{3.4}$$

where $\{\varepsilon_t\}$ is a noise process with mean zero and variance σ^2, and c is a constant. The backshift (or lag) operator B is defined such that:

$$By_t = y_{t-1}$$

In this context, $\phi(B)$ and $\theta(B)$ are polynomials of the backshift operator B, representing the autoregressive and moving average components of the model, respectively:

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p$$
$$\theta(B) = 1 + \theta_1 B + \theta_2 B^2 + \ldots + \theta_q B^q$$

Here, $\phi(B)$ is a polynomial of order p and $\theta(B)$ is a polynomial of order q. These polynomials modify the series y_t by incorporating terms that account for past values (lags) up to p for ϕ and q for θ, influencing the current value of y_t based on the specified autoregressive and moving average parameters.

A seasonal ARIMA$(p,d,q)(P,D,Q)_m$ model is defined as

$$\Phi(B^m)\phi(B)(1-B^m)^D(1-B)^d y_t = c + \Theta(B^m)\theta(B)\varepsilon_t$$

Selecting the best model of ARIMA (automatic ARIMA) is simply choosing the value of p,q,P,Q,d,D. If the values of d,D is known then Akaike's Information Criterion (AIC) is used to as the selector.

$$\text{AIC} = -2\log L + 2(p+q+P+Q)$$

L is the maximum likelyhood of the differenced data $(1-B^m)^D(1-B)^d y_t$

Maximum likely hood estimation:

$$\sum_{t=1}^{T} \varepsilon_t^2$$

The automatic ARIMA (Auto-ARIMA) utilised the information gain criteria to find the best fit model for the given data.

Assuming that a seasonal time series ARIMA model is considered. Let ARIMA$(p,d,q)(P,D,Q)$ be the model where p and q can have value from 0 to 3, and P and Q can take the value from 0 to 1. There are a total of 480 models. If the range of p,d,q,P,D and Q is wider, the number of possible models increases. The Auto-ARIMA uses a step-wise algorithm to choose the model with the lowest AIC without performing the model fitting on all possibilities.

Step 1: A small fixed number of models i.e.

- ARIMA(2,d, 2) if $m = 1$ and ARIMA(2,d, 2)(1, D, 1) if $m > 1$
- ARIMA(0,d, 0) if $m = 1$ and ARIMA(0,d, 0)(0, D, 0) if $m > 1$
- ARIMA(1,d, 0) if $m = 1$ and ARIMA(1,d, 0)(1, D, 0) if $m > 1$
- ARIMA(0,d, 1) if $m = 1$ and ARIMA(0,d, 1)(0, D, 1) if $m > 1$

Out of all these models, the auto-ARIMA choose the model with the lowest AIC.

Step 2: Consideration of branch models

- p, q, P and Q is allowed to change by ± 1;
- p and q both change by ± 1;
- P and Q both change by ± 1;

Whenever the lowest AIC is found, then the auto-ARIMA choose that model as the current best. The algorithm is bound by the maximum specified values of each parameter and thus convergence.

3.3 Gaussian Process

The Gaussian distribution is defined by the mean and variance of the data. Similarly, the Gaussian process is defined by a mean function and a covariance function $(m(x), K(x,x'))$.

$$f(x) \sim GP(m(x), K(x,x'))$$

Properties of K

1. $K_{ij} = E((Y_i - \mu_i)(Y_j - \mu_j))$
2. K is positive semi-definite
3. $K_{ii} \geq 0$
4. if Y_i and Y_j are independent then $K_{ij} = K_{ji} = 0$
5. otherwise $K_{ij} = K_{ji} > 0$

The covariance function used is the Radial basis function (RBF) kernel, as shown below.

$$K_{ij} = \tau e^{\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{\sigma^2}}$$

The evaluation of f can be done by estimating the $m(x)$ and $K(x,x')$. In theory, the best result comes from an infinite number of the evaluation of the two functions. In reality, however, this is not practical. We can sample the function at a finite arbitrary set of points $X : \mathbf{y} = f(X)$

$\mathbf{y} \sim \mathcal{N}(\mu, \Sigma)$ with $\mu = m(X)$ and $\Sigma = k(X,X)$.

We can use the Gaussian process to build regression models.

Let assume that we want to make a predictions $\mathbf{y}_2 = f(X_2)$ for n_2 new samples from the observed data points $(X_1, \mathbf{y}_1)$. Since the sample number are finite we have

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right)$$

where

$$\begin{aligned} \mu_1 &= m(X_1) \\ \mu_2 &= m(X_2) \\ \Sigma_{11} &= k(X_1, X_1) \\ \Sigma_{22} &= k(X_2, X_2) \\ \Sigma_{12} &= k(X_1, X_2) \\ \Sigma_{21} &= k(X_2, X_1) \end{aligned}$$

From the conditional distribution

$$\begin{aligned} p(\mathbf{y}_2 | \mathbf{y}_1, X_1, X_2) &= \mathcal{N}(\mu_{2|1}, \Sigma_{2|1}) \\ \mu_{2|1} &= \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{y}_1 - \mu_1) \\ \Sigma_{2|1} &= \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12} \end{aligned}$$

We can predict $\mathbf{y}_2$ from the input sample X_2 by using $\mu_{2|1}$

3.4 Casual Inference using Additive Noise Models

Let V be finite index set, $(X_i)_{i \in V}$ be random variables of a measured model of interest. $(E_i)_{i \in V}$ be noise. All random variables are real number and the noise have density with jointly independent.

$$p(e_V) = \prod_{i \in V} p_{E_i}(e_i)$$

$$X_i = f_i(X_{pa(i)}) + E_i, i \in V.$$

$$\mathbf{X} = \mathbf{f(X)} + \mathbf{E}$$

For a given two random variables X and Y, X is assumed to cause Y if

1. Y can be obtained as a function of X plus a noise term independent of X, but

2. X cannot be obtained as a function of Y plus independent noise; then, we infer that X causes Y.

In this case, where both hold simultaneously, the causal model is termed identifiable. The identifiability of the causal discovery depends on some conditions. For example, if all function f_i is linear, then the causal graph is entirely identifiable. For the non-linear case, if the noise variable is Gaussian, then the function is also identifiable (55).

The additive noise model from observational data

The causal model is built from the finite data set $\{x^{(n)}\}_{n=1}^{N}$. The causal mechanism $\{f_i\}_{i\in V}$ can be built from the observable data.

$$\underset{\hat{\mathbf{f}}}{\operatorname{argmax}} p(\hat{\mathbf{f}}) \prod_{n=1}^{N} \left(\left| \mathbf{I} - \nabla \hat{\mathbf{f}}(\mathbf{x}^{(n)}) \right| \prod_{i\in V} p_{E_i}(x_i^{(n)} - \hat{f}_i(\mathbf{x}_{pa(i)}^{(n)})) \right) \tag{3.5}$$

bivariate case:

$p(\hat{\mathbf{f}})$ is the prior distribution of the causal mechanism. The equation 3.5 is the minimiser of the noise variables. If the estimated functions lead to noise estimate mutually independent, then the model is accepted. The accepted model is the causal relation between parameters. The effect parameter behaves according to the inferrence function and depending on the cause parameter.

The function $\hat{\mathbf{f}}$ is estimated using the function approximation method, in this case, the Gaussian process.

The negative log-likelyhood $L := -\ln p(\mathscr{D}|\hat{f}_X, \hat{f}_Y)$ with the observable data $\mathscr{D} := \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$. The negative log-likelyhood then becomes,

$\mathscr{L} = -\sum_{i=1}^{N} \pi_{E_Y}(y^{(i)} - \hat{f}_Y(x^{(i)})) - \sum_{i=1}^{N} \pi_{E_X}(x^{(i)} - \hat{f}_X(y^{(i)})) - \sum_{i=1}^{N} \log \left| 1 - \hat{f}_Y'(x^{(i)}) \hat{f}_X'(y^{(i)}) \right|.$

The $E_X \sim \mathscr{N}(0, \sigma_X^2)$ and $E_Y \sim \mathscr{N}(0, \sigma_Y^2)$ is Gaussian noise and the Gaussian process for f_X and f_Y

$\hat{\mathbf{x}} := f_X \mathbf{y} \sim \mathscr{N}(0, \mathbf{K}_X(\mathbf{y}))$ where $\mathbf{K}_X$ is the Gram matrix with entries $K_{X;ij} = k_X(y^{(i)}, y^{(j)})$ and similarly for f_Y.

$$\min_{\hat{\mathbf{x}},\hat{\mathbf{y}}} \mathscr{L} = N\log\sigma_X + N\log\sigma_Y + \frac{1}{2}\log|\mathbf{K}_X| + \frac{1}{2}\log|\mathbf{K}_Y| + \min_{\hat{\mathbf{x}},\hat{\mathbf{y}}}\left(\frac{1}{2\sigma_Y^2}||\mathbf{y}-\hat{\mathbf{y}}||^2 + \frac{1}{2\sigma_X^2}||\mathbf{x}-\hat{\mathbf{x}}||^2 + \frac{1}{2}\hat{\mathbf{x}}^T\mathbf{K}_X^{-1}\hat{\mathbf{x}} + \frac{1}{2}\hat{\mathbf{y}}^T\mathbf{K}_Y^{-1}\hat{\mathbf{y}} - \sum_{i=1}^{N}\log\left|1 - \left(\frac{\partial k_Y}{\partial x}(x^{(i)},\mathbf{x})\mathbf{K}_Y^{-1}\hat{\mathbf{y}}\right)\left(\frac{\partial k_X}{\partial y}(y^{(i)},\mathbf{y})\mathbf{K}_X^{-1}\hat{\mathbf{x}}\right)\right|\right)$$

Gaussian covariance kernels

$$k_X(y,y') = \lambda_X^2 \exp\left(-\frac{(y-y')^2}{2\kappa_X^2}\right) + \rho\,\delta_{y,y'}$$

and similarly for k_Y

The optimisation problem is solved numerically using a conjugate gradient (95). Thus, the causal model is established (accepted) between two parameters of the observed data.

3.5 Summary

This chapter presents the essential methods which are presented throughout the dissertation in detail. We start with the profit maximising of the cloud broker with a limited deployment model that is feasible for a given customer data. If the deployment model is allowed to be more flexible, generating more profit is also a possibility. Lastly, the Gaussian process and the additive noise model can work with many data types, including the time series, which is the primary data type of our broker system. The problem formulation and the justification of the methods are explained in detail in the next chapter.

CHAPTER FOUR

CLOUD BROKERAGE STRATEGY AND CLOUD INVENTORY OPTIMISATION

We have discussed cloud computing and cloud brokerage system background, e.g. the methods and technologies in respective topics. In this chapter, we will analyse the setting of the bulk-purchasing cloud brokerage strategy and its challenge. We will then provide an abstract model for the key task of bulk-purchasing cloud brokerage, i.e. inventory management as an optimisation process. We will also briefly introduced our two solutions for inventory optimisation at a high level, i.e. a risk-based approach and a model-generator approach. The details of these solutions will be presented in details in § and §, respectively.

4.1 Cloud Brokerage System

We have established that a brokerage service can use the information of cloud instances in its command and compare them equally. Thus, it posses the knowledge of actual performance per price of each cloud instance. This knowledge helps to build a good service cloud broker.

The entry price of cloud computing is gated by the higher price per time unit of the on-demand. If the broker brought the price down, then there would undoubtedly be beneficial to both the providers and customers.

Cloud providers offer various pricing schemes for the same *Virtual Machines* (VM) instance with different lease options and prices. The price differences between each scheme can be pretty significant, e.g., a reserved instance can offer up to 75% discount compared to the on-demand price (27). The numbers of options give users the flexibility to tailor VM instances to the requirements of their applications. However, it also complicates the choice of VM, given that there have already been large numbers of VM configurations offered in the market.

The type of cloud broker in this work is similar to how a stock market broker works. In other words, there are multiple parties exchanging cloud instances through a broker where the appropriate pricing is determined. However, cloud instances are not commodity goods, nor do they have a variable value. They are fixed by the cloud providers, which are the main suppliers of the cloud instances. Thus, a cloud broker job is to mass gathering the demand to buy the cloud instances from providers. Mass buying is possible through the collective buying power of small customers.

Cloud computing power as a market

Virtual merchandises are not standard trade merchandise in a market. The concept of virtual merchandise is challenging to grasp. As the word trading usually applies to the exchange of objects. Now a day, virtual merchandise is a lot more common, most notably virtual currencies and, of course, cloud virtual machines. A virtual machine, or in other words, the right to use computing power on the data centre, is considered a product that can be sold or let by the cloud providers. As mention before, the lucrative cloud market opens many opportunities, one of which is the pricing scheme exploitation.

As one of the fastest-growing sectors, cloud computing as a platform has developed enough into a saturated market space. Thus, a cloud broker emerges as a tool to leverage the high complexity of the market. We have discussed the type of cloud brokers that operate on reselling cloud instances for a profit in the previous chapter. This business model is similar to crowd-sourcing, which proved to be successful in a big market (2).

Mass buying strategy: Commonly associated with bulk buying, mass buying is a collective buying power of multiple individuals through a middle man. Buying power is used to negotiate with the merchant or, in our case, cloud provider. In a cloud market space, a more common strategy is to do a time-shared machine. Time-sharing is also another type of mass buying, i.e. getting lower prices by combining more buying power. Mass buying may mean renting a longer but cheaper price instance of the same quality (cloud instance performance). With collective orders from cloud customers, a broker can create enough demand to fill the space while customers shared the time slots. Therefore, a broker inventory can consist of multiple long

term instances, which are significantly cheaper but offer the same performance as other pricing schemes. Thus, the customers would enjoy lower cloud instance costs.

Reserved instances as the primary commodity

Cloud providers segment their pricing schemes based on the duration of renting. Shorter renting time usually cost more per time than the longer one. This is to reduce the risk of the unpredictability of the customers' demand from the provider side. Therefore, the providers can afford the lower letting price of their instances. The reserved instances are the type of long term contract instances that offer low prices and stability of the service termination. Hence for a brokerage system, it is the most suitable primary cloud instance in the inventory.

The problem with using the reserved instance is the contract time. Similar to the providers, a broker needs to eliminate the uncertainty to operate efficiently. When a broker orders new reserved instances, it creates a commitment that needs to be filled.

The long contract time of the reserved instance creates the following problem. The first is commitment. By making a long commitment, a broker has to make sure that there are enough customers to occupied the instance. The second is the cost; a broker must keep the reserved instances thus must keep paying to the providers—the cost which needs existing funds (cash on hand). The third is the update circle. Computer hardware is updated and improved rather quickly. Hence, having a long commitment to one computer means that the performance per cost will be lower in the next iteration of the update than in the new instance.

Suppose all the issues are solved or avoided. This type of cloud brokerage service, if successful, can offer cloud customers both the benefit of cost-saving, flexibility, and ease of choosing the highest performance cloud instance for the money.

4.2 Cloud Computing as Commodity Model

A broker as a market concept is standard in many commodities exchanging ([23]). In a cloud market space, the concept is still relatively new as the cloud market itself.

Dating back to the time-sharing of a mainframe computer, users are competing for a time on the computing resource. During peak usage, the computing resource is arguably more valuable than during quiet time. Following a similar concept, a "better" cloud instance should cost more than a "worst" one. However, in reality, describing a good cloud instance is complicate. There are several aspects of the cloud that customers consider when choosing a cloud.

- Performance: As one of the most influences on the value of a cloud instance, performance is

the first metric that customers usually use to decide when getting a cloud instance.

- Location: Technically, a location is not something customers see upfront. However, location often plays a huge factor in latency value between customers and the data centre.
- Speciality: A special instance such as GPU accelerated for AI workload or specific operating system.
- Security: Security is a big factor when choosing cloud instances, as customers trust providers with their data.

There are other aspects: service type parameters, Quality of service (QoS), and service level agreements (SLAs). These usually are promises that the providers must honour and compensate when they fail to deliver. These are very important in a mission-critical cloud application. For example, the uptime of a life support monitoring application cannot fail, and therefore a very high level of SLA need to be upheld by the provider. Thus, choosing the right cloud is not easy.

In this work, we focus on the cloud instance as a market model, assuming that the performance of cloud instances is quantifiable. A broker manages cloud instances and redistributes virtual machines or instances to customers with similar user experiences as renting from original providers. The goal is to gather many customers' orders and assign them to a reserved instance for a better price per performance value. The exceed profit will also be redistributed to the broker customers making the cloud cheaper.

4.3 Cloud Broker Inventory Strategy and Problem Formulation

Cloud broker inventory is the main cost of broker operation. Given customers queries data, a cheaper inventory cost leads to a better profit. Accomplishing the minimum cost of cloud inventory is a game of matching the right combination of cloud instances in the broker inventory. As mentioned before, the reserved instance cannot be terminated before the expired date. Therefore there are some restrictions that the broker system must overcome. There are also some techniques that a broker can apply to increase the profit.

4.3.1 Problem Formulation

With the estimated user demand, the system can optimise the broker's target, i.e. utilisation, profit, etc. The effectiveness of the system depends entirely on the forecasting of the data.

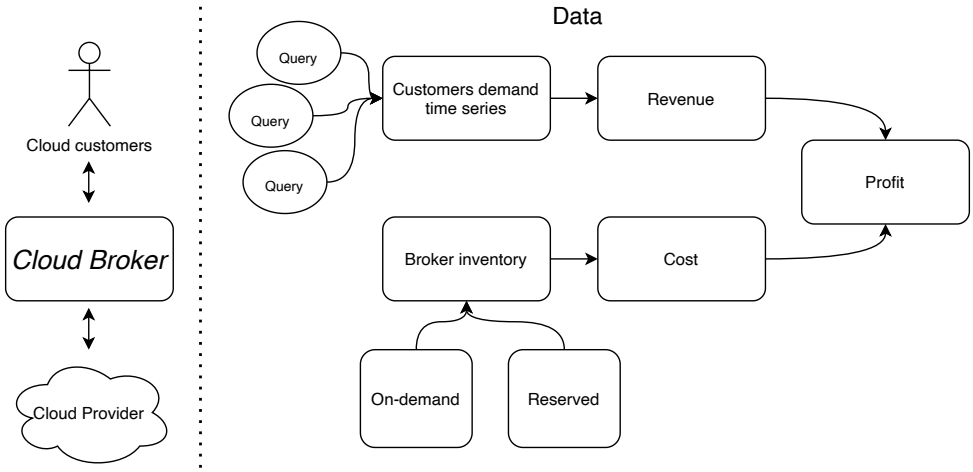


Figure 4.1: Cloud brokerage problem in this work focuses mainly on the inventory adjustment system. Individual queries from cloud customers create customer demand data. The demand data also contributes directly to the revenue of the broker. On the other hand, a broker inventory data consisted of cloud instances data, i.e. on-demand and reserved instances rent by the broker. This part of the data contributes directly to the cost of the broker. Lastly, profit is the value of revenue over cost.

However, the prediction accuracy goes down if the prediction points are further away from the supplied data. Together with the length of the reserved instance, a high-efficiency system is challenging to achieve.

One of the approaches to relinquishing the optimisation process and not relying heavily on the far end of the regression results is a just-in-time (JIT) decision-making system. On the other hand, coming up with effective decision making is not trivial. One of the approaches is to find a parameter (indicator) and a correspondence function when making a decision. The decision is triggered by a specific value or threshold of the parameter; hence the decision is made. It can be effective if the parameters and the decision functions mimic or relate to the behaviour of the system. This method is, in fact, similar to the artificial neural network model. The difference is that our decision model tries to figure out the connections between parameters while the ANN try to map out all connections with a giant net.

Cloud brokers collect query data from cloud customers on the front end and then distribute customers the computing instances. On the back end of the system, the individual query collectively creates usable data for analysis as shown in Figure 4.1. The broker system consisted of two data types, customers' query data and inventory data. The inventory data is a collection of cloud instances. The data is defined below.

Customer queries: `USR = {ID,Start,Spec,TIME} | {ID,Terminate,TIME}`

On-demand instances: `OND = {ID,Start,Spec,TIME} | {ID,Terminate,TIME}`

Reserved instances: `RESV = {ID,Start,Spec,Terminate,TIME}`

where, `Start` is submission time of customer resource query, starting time of the **On-demand** and **Reserved** instance. `Terminate` is submission time of the terminate query, termination time of the **On-demand** and **Reserved** instance. `Spec` is performance tier of the cloud instance. `TIME` is time unit of the internal clock of a broker.

The internal clock of a broker, as the name suggests, is keeping count of the time. The unit of time is a discrete cutoff of the barrier of time series data in a broker. For example, if the broker process data every second, then the unit of time of the second. All user queries between t and $t+1$ second are accumulate to form a time series data and similarly for the inventory data.

Discrete time is often employed when empirical measurements are involved because, typically, it is only possible to measure variables sequentially. For example, customer queries can happen continuously, but the broker can only take the queries discretely. Therefore, all of the data in a broker is discrete time series values that show sequences of the chosen time step. The variables of time series are indexed, which specify the time that the measurement takes place. For example, x_t refer to the value of a time series at a specific time period t.

On the other hand, in other areas such as physics or biology, a model of exact description often used continuous time. This is because the model itself can be derived mathematically. The values of a variable x is denoted as $x(t)$.

In a broker setting, the system measures customer queries in discrete time, developing a time series, which will be processed in a later stage of the system.

4.3.2 Optimisation of Targets

The main objective of a broker is to satisfy customers. Therefore, we assume that the broker can fulfil all the customers' demands, i.e. all customer queries are allocated with computing resources. We call this necessary condition.

After satisfying the necessary condition, it can aim to have the highest profit possible or optimal profit. A high-level example of profit optimisation in a cloud broker is given as follows.

The definition of optimisation is that it is a process of finding the best value of an objective function. It is a method of maximising or minimising a function by choosing the right set of variables from a pre-defined boundary set.

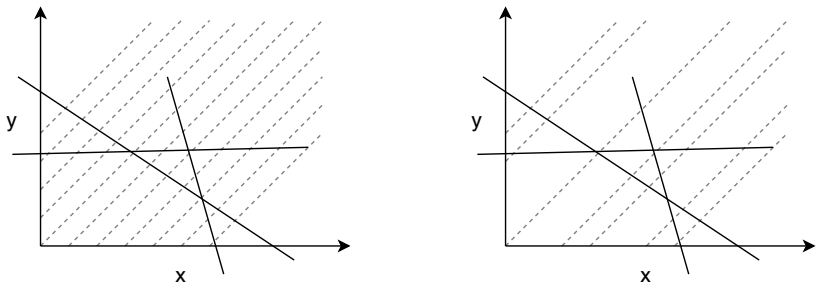


Figure 4.2: (LEFT) Typical two parameters optimisation problem with three constraints. The domain (in dotted pattern) is available throughout all of the evaluation points i.e. corners of the constraints. (RIGHT) In future values optimisation, the domain is not fully available. Thus, the system has to fill these missing data points by predictions.

A typical optimisation problem can be represented as a function f,

$$f : D \rightarrow \mathbb{R}$$

where D is the set of variables and $\mathbb{R}$ is a real number. The optimisation is to find $x_0 \in D$ for minimising and the opposite for maximising.

$$\forall x \in D, f(x_0) \leq f(x)$$

The profit of a broker is calculated from two sources *Revenue* and *Cost*. For simplicity, let us assume that the revenue of a broker is not affected by outside parameters, i.e. the price of the reselling instances. Therefore, the revenue of a broker is one parameter that the broker has no control over. The only parameter that the broker can change is the cost of running a broker itself. Let us assume that we can ignore the fixed cost such as salaries, advertisements etc. Then the cost consists purely of the cloud inventory. Thus, the combination of cloud instances in the inventory is the most critical parameter to consider. Additionally, the cost per performance goes down when the performance level goes up. A higher-performance cloud instance offers better value than the lower performance one.

The pre-defined boundaries of a broker problem are the restrictions from the inventory and the customers' queries. The combined performance level of queries cannot exceed the combined performance of the inventory at any time. The time is crucial since the total performance level (capacity) of the inventory changes over time, i.e. from the expiration of instances in the inventory and similarly in the customer queries. A fitting constraint such that individual query performance cannot split, i.e. high-level performance query, cannot be split and placed onto multiple low-level instances.

Cloud broker optimisation is also a special case. Typically, an optimisation process is done when the information is readily available, as shown in Figure 4.2—for example, minimising the cost of the lunch while having to fulfil the nutrient needs. The information about food and its nutrient is known to the optimiser. Therefore, it can choose the optimal meals set with the lowest cost while having all the necessary nutrients.

However, the *Revenue* part of the broker, while unaffected by the broker, the data is not fully available throughout the reserved instances contract duration. To perform optimisation, the broker has to estimate the demand, which also estimates the optimisation values. Nonetheless, estimation is an essential part of many systems that deal with the uncertainty of the future.

4.3.3 Data Estimation and Inventory Adjustment

Time series data in a broker, like any time series, the main focus of *time* is to forecast the future values. There are various approaches to dealing with the forecasting of a time series.

Data estimation target

From a broker system, it is easy to assume that if the demand can be estimated accurately, then the inventory of a broker can be prepared accordingly.

Customers' queries individually consisted of two main parts, specification (performance level) and time. We are ignoring the timestamps since they are encoded while transforming individual datum into a time series. The specification part can be quantified, assuming that the numerical results of benchmarks can capture cloud instances' performance. Time series data is partitioned based on *time* of the numerical performance level. Together, they form a demand per the time of cloud customers.

For example, we build a simple broker which takes one type of performance tier, i.e. customers only have one choice. The demand curve is relatively simple, as shown in Figure 4.3. If a customer starts a query, then the demand curve goes up by one unit. On the other hand, if a customer terminates the query, then the demand curve goes down.

On the broker side, it has to match the demand with the supply of cloud instances. From Figure 4.4, the system rents reserved instances that are re-assigned to the customers.

The deployment model of this example is one-to-one, one query for one instance. The broker needs to estimate the `number of queries` per time unit. The problem is relatively simple to solve with a prediction algorithm. Higher accuracy prediction algorithm will lead to better planning of the inventory.

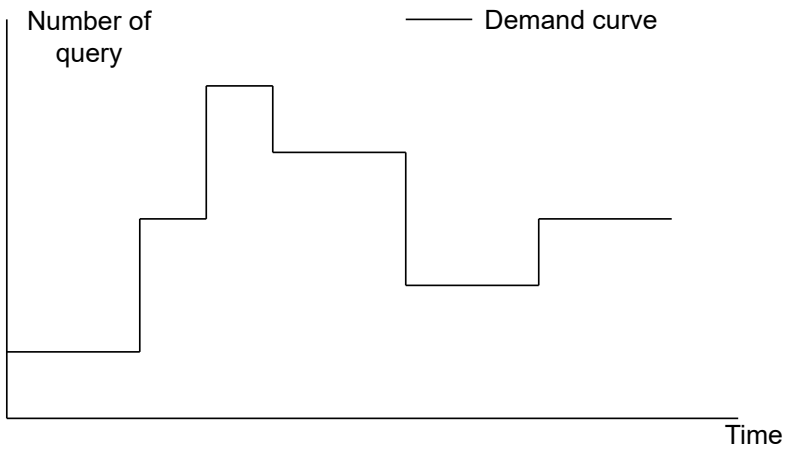


Figure 4.3: The graph shows an example of number of query over time. The vertical axis is the number of query collectively can roughly estimate the demand of cloud customers. The horizontal axis is the time.

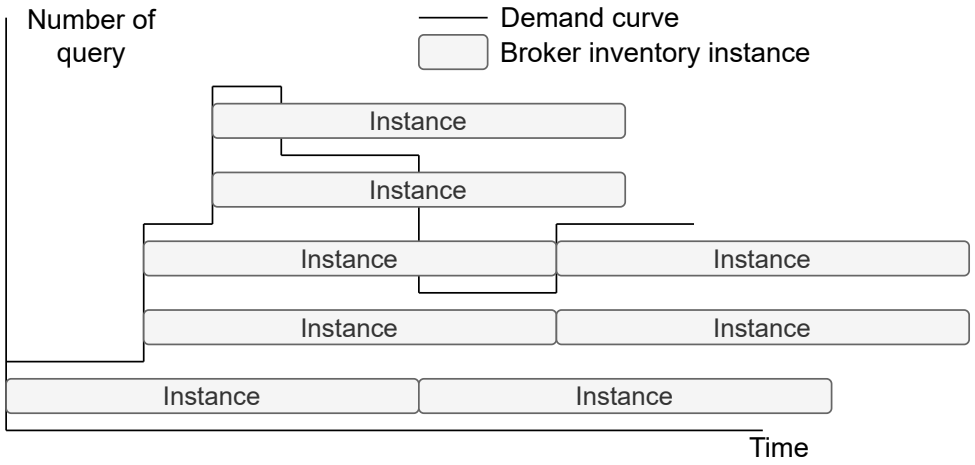


Figure 4.4: Cloud broker inventory consists of many cloud instances which are there to accommodate customer queries. In this graph, we assume that the query demand is quantifiable, and therefore we can assign them to each cloud instance in its inventory. The graph also shows that if the duration of the instance is fixed, then there is a chance that some of the instances might not be utilised which is the main problem of the profit loss.

The result is in the form of predicted values of the required reserved instance. For example, if the algorithm estimate that the broker needs k spot at time t, then the broker must prepare a combination of a reserved and on-demand instances of size k at time t. However, demand matching does not equal profit maximisation. For this, we need to talk about the cost.

$$Cost(t) = \sum_{i=0}^{n_t} ResvC_i + \sum_{j=0}^{m_t} OndC_j \tag{4.1}$$

where n_t and m_t is the number of new instances being put in the broker inventory at time t. *ResvC* and *OndC* is the average cost of reserved and on-demand instance at time t respectively.

Let assume that the cost function is

$$Cost(t) = A + B.$$

Let us assume that A and B are a list of all the values solution at each time step.

$$A = [a_1, a_2, a_3, ...]$$

$$B = [b_1, b_2, b_3, ...]$$

The optimisation aims to find the correct value of each a_i and b_i in A and B. Since the broker uses an estimated data, the system might overestimate or underestimate some of the value. If the system overestimates values of A and they become higher than optimal. Then it might cost damage to the profit since *Resv* has a significant upfront cost. On the contrary, if the system underestimates values of B and they become higher than the optimal values. The total cost per hour goes up, and fewer *Resv* instances are being deployed. This is called an opportunity cost. Balancing the two is an act of placing bias in the decision making in favour of one type of an instance at a certain time. A proposed solution for this problem can be found in Chapter 5.

The previous example is an example of a one-to-one deployment model. If a broker becomes complex, then an estimation of data also becomes more complex. For example, the broker employs more than one performance tier. Since each of the cloud instance performances is quantifiable, then, in theory, larger instances can accommodate multiple smaller queries. The system is not a one-to-one deployment anymore. The estimation targets would have to be modified. The broker cannot simply estimate the demand for one specific instance and then prepare the said instances in the inventory and hope to achieve optimal profit.

The broker can utilise a regression method. For example, f is the regressed function from the optimal data,

$$\mathbf{x} := f(\mathbf{y}) \tag{4.2}$$

where $\mathbf{x}$ and $\mathbf{y}$ is the targets (number of each instance type) and customers time series data respectively. The regressed function essentially is a relation between two parameters. If the broker can identify this relation, then it can make adjustments based on this connection to achieve the desired effects.

However, the regression model also has many challenges in itself, such as data selections, independence of the selected data etc. A proposed solution is presented in Chapter 6 which aim to generate the profit model automatically. The system utilises a method similar to a causal discovery method which infers the underlying relation of associate parameters.

Causality and cloud brokerage system

For the most part, if one wants to find straightforward relation between a variable and a target, the correlation test or regression can effectively do the job. If one wants to find a relationship between two variables, it can be found regardless of the data. Additionally, in a complex system where multiple variables are entangled in a web of the relationship, it is rather challenging to find a true relationship where one variable affects another if there is a change in the former.

One example in a broker system is the number of all instances in the inventory or size and the profit. Both of the variables are going to correlate, i.e. the change in size will change the profit. Nevertheless, the correlation might not have a real underlying effect on one another.

Therefore, a causal discovery technique can infer underlying relationships and their correspondence variables to identify whether or not these parameters adjustment would work accordingly. Hence, we are going to explore the cloud brokerage system with an additive noise model.

4.4 Proposed Solutions

Research motivation summary

The heart of the broker inventory optimisation problem is the long term planning with constraints. The primary instance type in the inventory is the reserved instance which offers a balance between price and stability. The conditions of reserved instances are long contract time and non-reducible (cannot be cancelled before the expiration of contract). Therefore, for a broker operation to generate profit while fulfilling a broker's condition, it has to overcome these two issues.

The long contract duration means that the broker must forecast the customer demand into the future. Moreover, the fixed expiration time means that the broker has to make a good decision when deploying the instance to overcome the idling issue. The indication of making a good decision in a broker setting is not always about the accuracy but rather the overall cost as well. Thus, optimisation is a good starting method. Nevertheless, since the optimisation needs complete data, other approaches of peak locator must be considered.

In a complex system, often time researchers observe data to understand the overall system interaction. With the observed data, a model of the system is built and used as *rules* govern how each entity behave and influence each other. Thus the two approaches are proposed to capture the profit behaviour of the brokerage system with the inventory adjustment system as the following.

We propose two solutions that solve slightly different problem settings for the given cloud broker inventory adjustment problem.

Risk-based solution Since we have established that parameters in a broker system can affect profit, then there are some potential solutions to the system at finding the optimal broker inventory. First, we propose a pre-defined set of parameters that influence the profit, and the decision-making function is made around the risk factors. We called this approach a risk-based brokerage system. The risk assessment was done based on what might cause the system to lose money, i.e. empty reserved instances, excessive use of on-demand etc.

Automate profit model generator At some point, the broker system can become complex and thus, pre-defined risk factors might not cover the case. We propose an automatic feature selection and profit model generator. The generated model is used to decide the broker inventory, similar to the risk-based model.

The selection of parameters to build a model is usually involves an experienced manual selection process (63; 6). From Figure 4.5, we propose a system that utilised the additive noise model to achieve both parameter selection and functional estimation. The left-hand side shows a set of variables that potentially affect the profit. The right-hand side shows after testing the variables set with the additive noise model reveal the underlying relation F between variable e_k and profit.

4.5 Scope of the Work

Although this work mentions many issues of cloud brokerage systems and cloud computing in general, the focus of this work is mainly on inventory optimisation using time series models. A complete broker system consisted of multiple components.

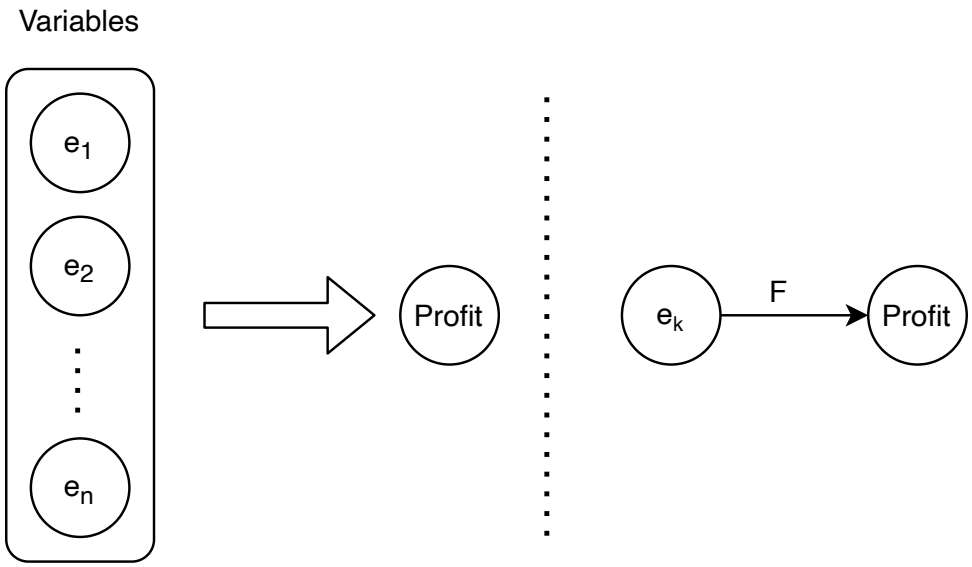


Figure 4.5: The automatic model builder filter the parameters which associate with the profit and infer the model F. With the model, we can form the relationship of e_k and profit. Thus, we can make an effective adjustment at e_k to influence the profit.

- Performance capture: benchmark and the creation of the performance data database. This stage is important for a fair comparison between multiple cloud instances.
- Broker optimisation: optimisation of a broker inventory after attaining the data from the first part to make a good profit and in return give cashback as an incentive to customers.
- Resource allocation: the framework for allocating the customer queries with the real cloud virtual machine inside the cloud instance.
- Other frameworks: handling of security and other services.

For our broker model, a complete system consisted of three parts, the performance capture system, inventory optimiser, and query allocation. Although each part of them are equally important, the main work in this work is focused on the data estimation and inventory optimiser as shown in the dotted rectangle in Figure 4.6. The other parts are based on other existing research.

Solutions comparison

In this work, we have proposed two solutions to the inventory optimisation problem. Both solutions are based on a mathematical profit model. Although they work on the same problem, there are differences in places, and each of them has strengths and weaknesses.

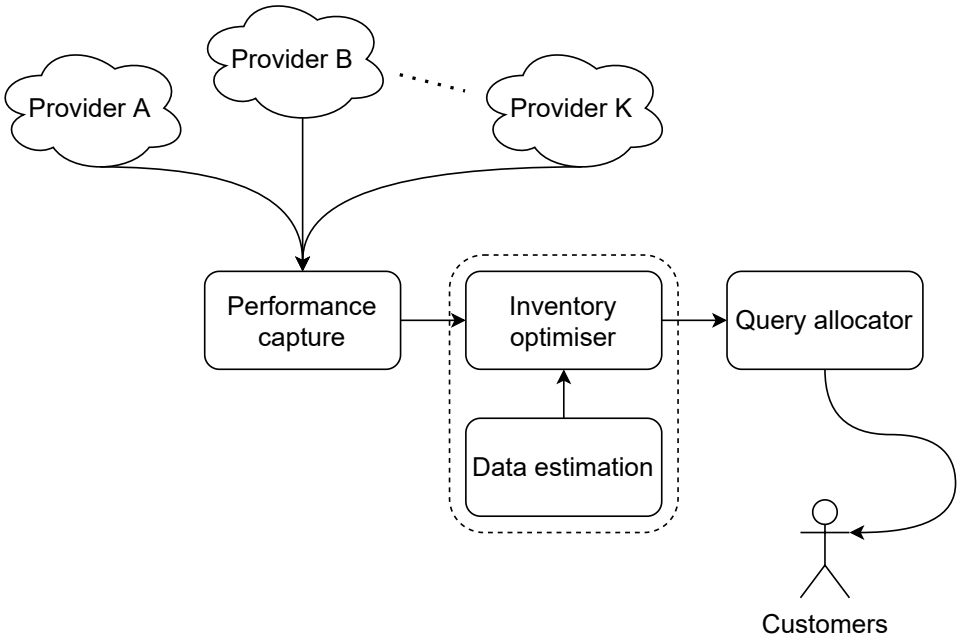


Figure 4.6: There are many areas of cloud brokerage system research. In our broker model, there are three main parts which contribute to the system, performance capture, inventory, and allocation. However, only the dotted rectangle is the main focus of this work.

- Pre-defined risk factors: use human-defined risk factors to aid the decision making of the broker.
- Automatic profit model generator: trying to discover underlying relationship from features of data and the profit.

Differences

Deployment model: risk-based broker only investigate on a one-to-one deployment model. This is to simplify the effect of risk factors. As for the additive noise model, the deployment is a mixture of multiple performance levels, and the cloud instances can host multiple customer orders.

General approach: risk-based use of pre-defined risk factors as an indicator for adjusting broker inventory. On the other hand, the additive noise model selects associated features from the feature set as a relationship pair to the profit. The broker inventory is modified using the model inferred from the mentioned pair.

Strengths and weaknesses

The risk-based broker uses pre-defined risk factors; therefore, the system is considered data

independent. If there is a change in the data distribution, the system behaviour does not change. Thus, it is resistant to changes in data distribution and operates independently of the input. This makes the system more stable. On the other hand, the system is limited to pre-defined risk. If the system is expanded, then a new set of risk factors have to be developed.

On the other hand, the auto-model generator can automatically identify the associate parameter from the given data. Thus, it requires less know-how for parameters configuration. Since it builds a profit model from the data generated directly from the broker system, it is adapted to the change in the broker system configuration. Nevertheless, the auto-model still need input data to build or form a model. So it inherits a similar problem to the other supervised algorithms.

4.6 Summary

In this section, we have explored cloud broker research from cloud computing challenges and problems to the cloud broker, particularly inventory optimisation of the cloud broker using various techniques.

Since the causal discovery is a recent technique that has not been applied to many studies yet, we have investigated using the additive noise model in a simple broker example. We found that it is as effective as the method promise to achieve.

We also introduce two approaches to solve the issue of using reserved instances as the primary resource in cloud broker inventory. Firstly, the risk-based broker utilises the concept of risk to aid the decision-making process in a broker. Secondly, the additive noise model broker selects and build a model from a feature set. Both techniques utilised different techniques to achieve a similar goal in two independent broker models.

CHAPTER FIVE

RISK-BASED CLOUD BROKERAGE

The concept of risk is mainly a subjective assessment used by human operators (40). For any operation, risks are something that might cause damage to the system. Thus for many, they need to be eliminated. On the other hand, in a financial sense, risks can be thought of as an opportunity. If the risks are managed well, then it could result in a big reward. The downside is that an expert must quantify them; otherwise, the big reward might turn into a significant loss. Broker operations can also run with some risks in mind from the uncertainty of the demand. Hence in this work, we utilise the risk concept to help with the inventory adjustment of our brokerage system.

5.1 Risk Concept Usage in a Cloud Brokerage System

In a broker buy-low-sell-high setting, the system has to stock enough cloud instances to accommodate customer demand. It can offload some non-crucial queries to more expensive but less risky instances such as on-demand or shorter-term reserved instances. The concept of risk has been associate with decision-making steps and more prominently in the financial sector. In this part, we are exploring some of the risk concepts being used in a system together with the algorithmic risks determinator.

The standard definition of risk in different applications is *effect of uncertainty on objectives* (79). Below are a few examples of risk management using an algorithmic system in a time series problem, i.e. stock market. The stock market is a volatile, fast-moving market that makes its data behave unexpectedly. Time series data in a stock market is subject to many studies, especially prediction and forecasting, to inform traders with better decision-making results. One of the

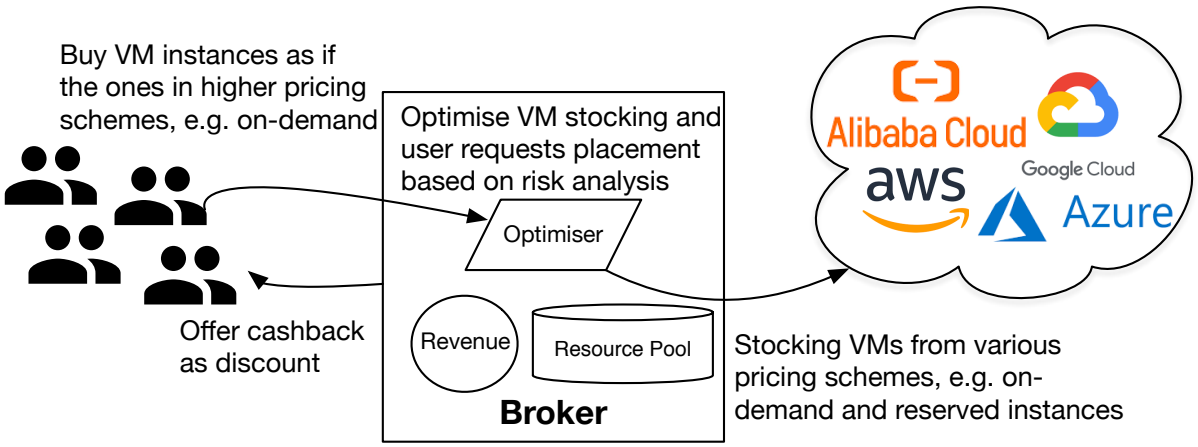


Figure 5.1: A broker model to simplify the choice of VMs from various pricing schemes. Users can both enjoy the flexibility as from on-demand instances and have discounts as from contract instance.

major trading systems is the technical trading rules. The authors of (36) and (3) uses a genetic algorithm to generate technical trading rules which work best. Similarly, a hybrid stock trading based on a genetic network with a value-at-risk system is also studied in (22). These trading *rules* are indicators that try to predict the behaviour of the data to deal with the risk of trading. Broker systems also associate heavily with risk while in operation. In this section, we are employing risk concepts and apply them in a broker environment.

In this chapter, we propose a broker model to simplify the choice of VMs from different price schemes, as shown in Figure 5.1. The key to the strategy is to stock VM instance types from discounted contract pricing schemes in a *broker inventory* and then to resell them as VMs with a pay-as-you-go scheme to potential cloud users.

Cloud users can purchase a VM instance as if the instance is from a higher pricing scheme (reserved) to enjoy the flexibility of the instance whilst receiving cashback offered by the broker to reduce the cost. For the ease of presentation, we will take *reserved instances*, which is a discounted scheme that requiring commitments of a long contract length. On the other hand, an *on-demand* is a flexible but more expensive pay-as-you-go scheme.

The challenge of the strategy is how to optimise the stock of VM instance types from different pricing schemes according to the number of user requests. The problem that might occur when the cloud inventory is not well optimised is shown in Figure 5.2. The figure illustrates the cases of over-stocking and under-stocking cloud inventory. Both situations are not well suited for the generation of high profitability.

One popular method at finding the right size inventory utilises the past data points to predict (time series forecasting) the user requests values. Then the system plans the size and composition

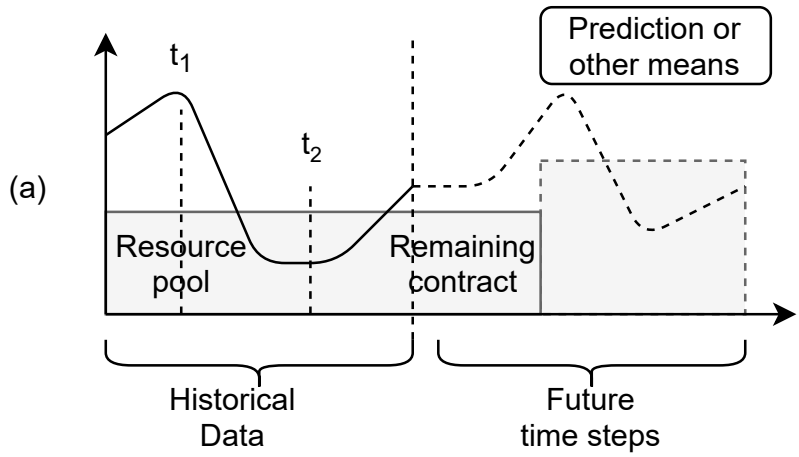


Figure 5.2: Solving a time series data problem involves predicting future values. For example, at t_1 the current user orders exceed the inventory size, some of the orders have to be offloaded to the on-demand instances; t_2 is a point where the inventory is underutilised. Accuracy predictions are the key to reduce the cost and make the optimal decision on which scheme type of VMs the user order will be placed on.

of its inventory accordingly (84; 115; 116). The effectiveness of this method depends entirely on the assumption that past data can predict newer data points accurately. However, the user requests in real life are full of uncertainty, and it is non-trivial to justify whether a data set would match well with the prepared inventory.

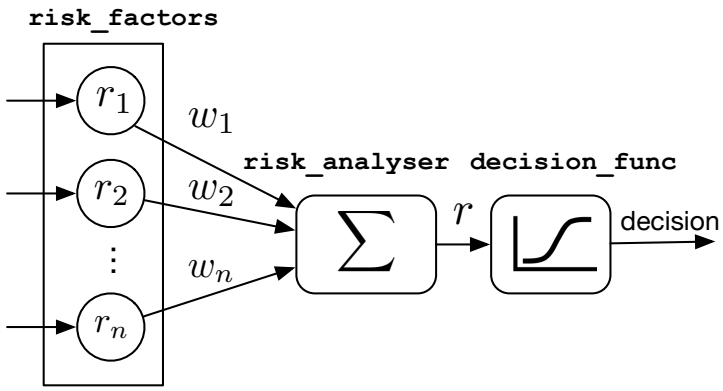
Inspired by the risk-oriented trading strategy in the stock market (35), we take an alternative approach to drive the decision making with risk. We dynamically adjust the inventory by evaluating the risks calculated from both the user requests data and the inventory data.

5.2 Inventory Optimisation and User Request Placement using Risk Analysis

Our broker stocks reserves VMs instances in its inventory and resells them as on-demand instances. The user experience is the same as buying on-demand instances directly from cloud providers, i.e., users can terminate VM at any time. When terminated, users can then get cashback as discounts. The cashback amount depends on the time usage and profit of the broker.

Internally, our broker places user requests in the VMs from the inventory. When there is no more VM available, the broker needs to decide whether to stock reserved instances to fulfil the request or use an on-demand instead. Such a decision-making process is the centre of our broker.

We formulate the decision-making problem as a binary classifying function that taking quantitative risk factors as input. Figure 5.3 gives an overview of our risk-analysis based




```
Input:
// A set of quantitative risk factors
risk_factors: (T -> ℝ) set
risk_analyser: ℝ set -> ℝ
decision_func: ℝ -> bool
Output:
// Define Optimisation with function composition
reinforce_dec =
  decision_func ∘ risk_analyser ∘ map risk_factors
```

Figure 5.3: Risk-analysis based decision making process.

decision-making process, together with an abstract type definition of each part.

There are three parts in the process: quantitative *risk factors*, a *risk analyser* and a *decision function*.

- Each risk factor turns the status of brokers into a quantitative risk measure. The broker status covers the perspectives from the view of user requests and reserved instances. In the abstract definition, a generic data type (`T`) is used to allows a function in `risk_factor` to accept parameters to produce a quantitative result for a risk factor.

- *Risk analyser* normalises and assigns weights to each risk before aggregating them. There is also an additional risk-taking adjustment based on the current revenue level. The output *r* is between [0,1].

- *Decision function* takes the aggregated risk to decide whether to buy a reserved instance or an on-demand one to fulfil a user request.

In the rest of this section, we will give more details of each part of the process shown in Figure 5.3.

5.2.1 Quantitative Risk Factors

We analyse the risks of stocking more reserved instances from the following aspects.

Anomaly user requests: We should consider stocking more reserved instances if and only if the number of requests indicates an increasing trend in VM usage. In this case, we use a mean and standard deviation (*mean-sd*) anomaly detection. Recall that our decision-making process is triggered by a periodic function that checks the pending queue so that we will use the same period as a time unit for our analysis. For the ease of presentation, we name the time of unit as $\mathscr{T}$ and define the user request rate as the number of the user requests over $\mathscr{T}$.

We quantify this factor by comparing the most recent user request rate to the mean of user request rate over the most recent 10% length of a typical contract length of the reserved instance. For example, if the contract length is ten months, the mean is calculated from the most recent one month. The method is also known as z-score anomaly detection (66).

The main point of this process is to reduce the number of outliers in the series data. The extreme values of outliers would artificially inflate the broker's demand, resulting in an unwanted escalation in cost.

The risk threshold is twice the corresponding standard deviation, i.e., the range of anomaly requests is:$[mean, mean + 2 * sd]$. The threshold is configurable.

The quantitative risk factor of the anomaly requests is defined as:

$$\begin{cases} 0 & r(t) < mean \\ \frac{r(t)-mean}{2*sd} & mean \leq r(t) \leq mean + 2 * sd \\ 1 & r(t) > mean + 2 * sd \end{cases} \tag{5.1}$$

where $r(t)$ is the time series function at time t. To further improve the accuracy of risk analysis, on can introduce more anomaly detection to eliminate other types of outliers. We will discuss possible directions in future work.

Total numbers of reserved instances: The current number of reserved instances (the size of the broker inventory) is also a risk factor. This risk factor is similar to anomaly detection but from the inventory data rather than the customers' data. If the current number is significantly higher than the average number, we should rate it as a substantially high risk. Similar to the anomaly request, we also take the mean of reserved instance size of the most recent 10% of the typical contract length of the reserved instance for comparison. The function of calculating this risk factor is the same as (5.1).

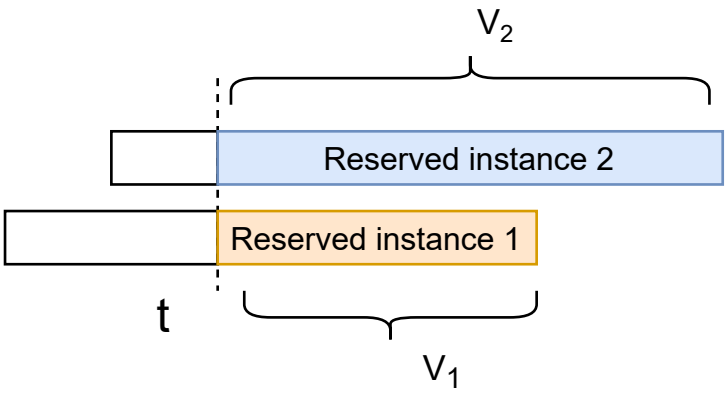


Figure 5.4: Using volume of the inventory as a risk factor: $V_1 + V_2$

Volumes of reserved instances: Another dimension of the risk of the current reserved instance stock is the volume, which is the remaining length of the contract. To illustrate, in the example shown in Figure 5.4, the volume is $V_1 + V_2$.

The function to calculate the risk factor of volume is:

$$\frac{\sum_{i=1}^{N} vol_i}{N * len}$$

Where vol_i is the current volumes of a reserved instance, N is the total number of the instances, and len is the length of the contract. We can easily see that the size of free future volume corresponds to risk.

5.2.2 Normalised Linear Risk Analyser

To combine the risk factors listed, we use a linear risk model to normalise our risk values. The impact of each risk factor on profitability cannot be quantified without pre-existing data. Hence, throughout the experiment, we consider all risk factors to be of equal importance—the range of the sum of all risk factors within $[0, 1]$. So, we assign a weight vector to normalise the range of each factor accordingly, i.e., $\vec{w}^{\intercal} * \vec{r}$, where $\vec{w}$ is the weight vector, and r is a vector of the risk factors. The output value is the sum of normalised risk factors.

Risk-taking risk adjustment: We would also like to take the current revenue level as a 'positive' risk factor to adjust the sum of all the risk factors. The intuition behind this is to allow higher risk to get more profit. This is a trading strategy in the traditional marketplace, called a risk-taking strategy. Here, we adopt a trading strategy in our inventory adjustment. We compare the current revenue in the most recent period of $\mathscr{T}$ with the one in the period before. If the revenue rate is higher, we allow the broker to take more risks by reducing the risk factor by a fixed amount of

0.05 (5 %). Similarly, it increases 0.05 when the revenue rate is lower. The number is arbitrary chosen to be 5% of the maximum risk value. The main reason for allowing risk adjustment based on the money on hand is to identify the effect of opportunity cost and profitability of a broker. Balancing the amount of cash on hand is also one of the financial trade practices. In other words, cash is one of our soft risk factors. Note that the range of the risk factor after adjustment remain [0,1].

5.2.3 Decision-Making Function

Decision making is essentially a predicate that takes some quantitative risk factors to produce a boolean value for a decision. In our case, we would like the decision function to satisfy the following requirements:

(i) Being able to take any value in the range of a sum of the risk factors, i.e. [0,1]

(ii) The likelihood of creating a reserved instance changes continuously with the value of the sum of risk factors.

A non-deterministic function is used to satisfy the requirements, i.e.,

$$S(r) = \begin{cases} 1 & r = 0 \\ 1 - e^{-\frac{rng(0,1)}{r}} & 0 < r \leq 1 \end{cases} \tag{5.2}$$

where $rng(0,1)$ picks a uniformly distributed random number between 0 and 1. When the output value is less and equal than 0.5, the broker will create a reserved instance to accommodate pending user requests, otherwise, an on-domain instance is created. With this function, the higher the risk is, the lower the chance of creating a reserved instance becomes. Also, the broker is strategically more inclined to allocate a reserved instance because the overall likelihood of getting a value below 0.5 is higher than the value above 0.5. Figure 5.5 shows three cases when risk is low (0.1), medium (0.5) and high (0.9). The randomness introduced into the system is inspired by the mutation algorithm.

To decide for each pending user request, the broker will first compute the sum of the normalised risk factors, generate the likelihood of each decision with a curve function, and finally 'rolls a dice' to get a decision.

We have presented all the parts for our risk-analysis-based approach to optimise the broker's inventory and user request placement. A summarised pseudocode for our approach is shown in Figure 5.6.

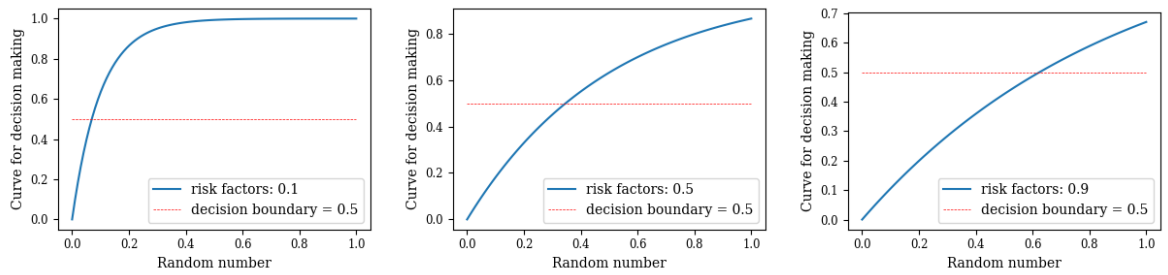


Figure 5.5: Plots showing the decision curve when risk is low (0.1), medium (0.5) and high (0.9). x-axis is a random number generated between [0,1] with equal probability. When $y \leq 0.5$, the broker will create a reserved instance. The length of the arc curve represents the corresponding likelihood. A guideline of $y = 0.5$ is provided for reference.

In the next section, we will give a detailed design and implementation of our broker.

5.3 Detailed Specification & Implementation

To adopt the pricing strategy, we develop a broker system which consists of 7 main components: a user request scheduler, a request database, a pending serving queue, a broker inventory an optimiser, and a provisioner. Figure 5.7 shows different workflows of accommodating a user request. For the ease of clarification, we provide the following high-level abstract type definition of the broker and its component to walk through the functionality of the system:

where `v: T` denotes a variable `v` with type `T`; `t_1 * ... * t_n` denotes the type of a n-dimensional tuple; and `X -> Y` denotes a function type.

A broker is defined as a tuple of the 6 main components: a request database (`rqst_db`) stores user requests as time series data; a broker inventory (`inv_pool`) maintains a collections of VM instances with different pricing schemes, as well as the occupying relationship between the VM instance and the user requests; a pending serving queue (`rqst_srv_queue`) keeps track of the user requests that are yet to be fulfilled due to lacks of reserved instances in the pool; a user request scheduler (`rqst_schdlr`) is a function to coordinate other components for request fulfilment depending on the availability of reserved instances; an optimiser (`optimiser`) is an adjustment function of the stock of VMs in the inventory in order to fulfil the requests in the pending queue; and a provisioner (`provisioner`) is an agent to communicate with the cloud providers to create and terminate VM instances.

Upon the arrival of each user request, the scheduler responds to the request by assigning/releasing the binding between the user request and the VMs in the pool. If there is a query to create a VM and then it cannot be fulfilled by the inventory, the scheduler will put the request in the pending

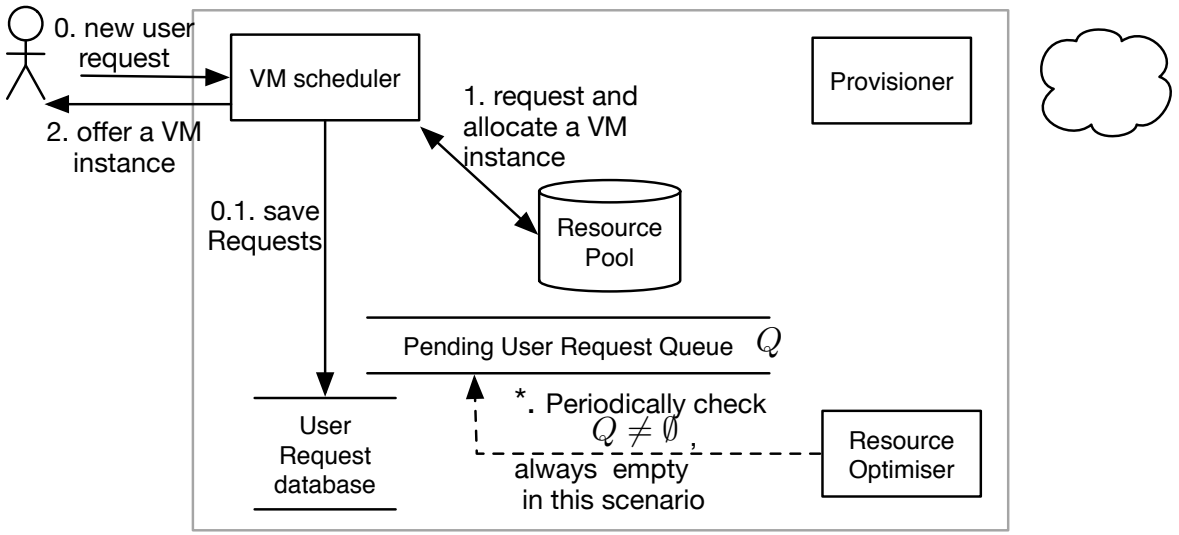
```
Input:
// Functions to compute current risk factors
risk_factors = {F_anomaly_rqst, F_vm_num,
                F_vm_vol, F_profit}
// A linear risk model
risk_model = λw⃗, risks⃗ : w⃗^T * risks⃗
// A non-determined predicate to make decision
decision_func = λr : 1 − e^(−rng(0,1)/r)
// Weights to balance and normalise risk factors
w⃗_risks
/* Data from the broker: request db, inventory broekrand pending request queue
    */
rqst_db, inv_pool, rqst_srv_queue
Output:
/* Decision for whether to create a new reserved instance or an on-demand
    instance for each pending request */
decisions: bool list
Procedure:
decisions = []
risks⃗ = [F_anomaly_rqst(rqst_db), F_vm_num(inv_pool),
         F_vm_vol(inv_pool), F_profit(current)]
for ∀ rqst ∈ rqst_srv_queue do
  x = decision_func(risks⃗, w⃗_risks)
  decisions.append(x)
end
```

Figure 5.6: Detailed specification of the decision making process with risk analysis

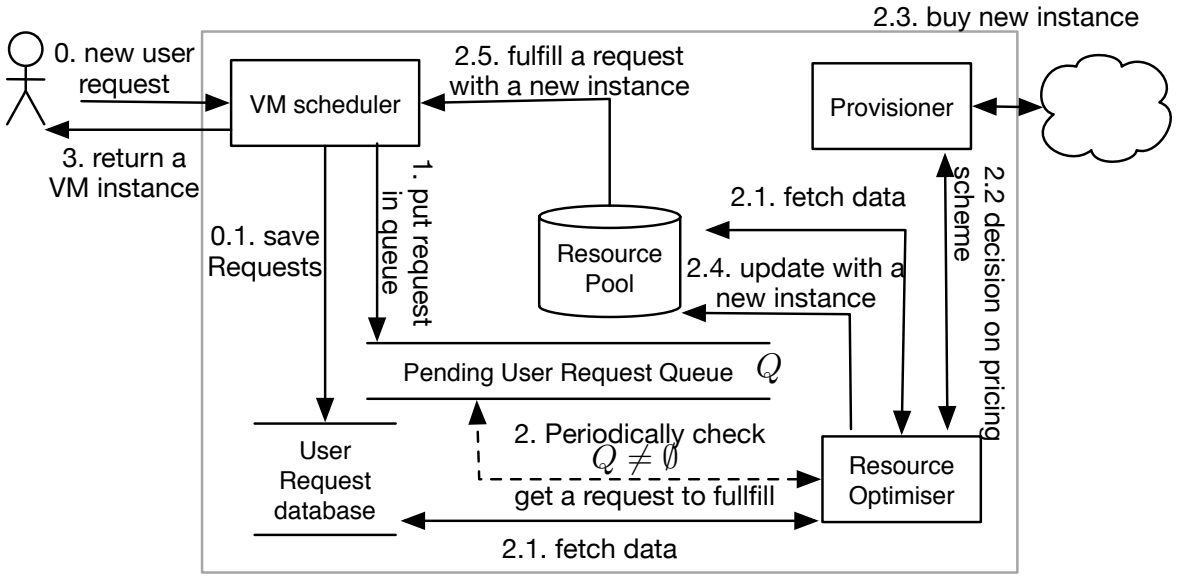
queue. The optimiser periodically checks if there are pending requests. If so, the optimiser will then review the current stocking level in the inventory and a period of most recent user requests to evaluate the risk level. The pending requests must be fulfilled; thus, the optimiser can choose to either stock more reserved instances if the risk is low or buy on-domain instances in the case of high risk. The details of the process of risk analysis will be presented in the next section.

Another critical aspect to keep the broker sustainable is the cashback model. Cashback is generated directly from the price difference between running costs and the income from reselling VMs. For example, if a user rents a VM for a t_u unit duration and a broker made $P\%$ profit during the said duration. Then, the user is entitled to earn **at most** $P\%$ cashback of what was initially spent.

The revenue of a broker is calculated from three main components, the user demands, broker running cost, and the cashback value. Each of the components affects the profitability of the broker system differently and is certainly not trivial. If the broker aims to maximise the profit, it must be able to identify the correlation functions of each sub-component. The task is both



(A) The broker will allocate a reserved instance directly from the inventory to accommodate a user request when a reserved instance is available.



(B) The broker will create either an on-demand or a reserved instance to accommodate a user request on when no reserved instance is available.

Figure 5.7: Workflows illustrating the broker's responses to a user request when a VM is available (A) and when a no VM is available (B).

```
broker = (
 rqst_db: RqstDB_T,
 rqst_srv_queue: RqstQ_T,
 inv_pool: INV_T,
 rqst_schdlr: Rqst_T * (RqstDB_T * RqstQ_T * INV_T) -> RqstDB_T * RqstQ_T *
     INV_T,
 optimiser: RqstQ_T * INV_T -> INV_T,
 provisioner = (
  create_vm: VM_T * PrcSchm_T -> Rsrc_T,
  destroy_vm: Rsrc_T -> unit
 )
)
// Type Price Scheme
type PrcSchm_T = Revered_T | OnDemand_T
// Type for User Requests
type Rqst_T = UID * VM_T * CREATION | TERMINATION
// Type for the User Request DB
type RqstDB_T = (Rqst_T * time) set
// Type for the Request Queue
type RqstQ_T = Rqst_T list
// Type for VM Resource
type Rsrc_T = RID * VM_T * PrcSchm_T
// Type for the Broker inventory
type INV_T = Rsrc_T -> Rqst_T
```

Figure 5.8: Abstract type definition and specification for the broker model

challenging and dynamic. Therefore, in this work, we are only interested in the gross profit from the broker operation without diving too deep into the correlation functions.

The gross profit margin (Ψ) from the time step t_1 to t_2 is calculated using the equation 5.3.

$$\begin{aligned} \Psi(t_1,t_2) &= 100 * \frac{\rho(t_1,t_2) - \omega(t_1,t_2)}{\rho(t_1,t_2)} \\ \rho(t_1,t_2) &= \sum_{i=0}^{n_r^{(t_1,t_2)}} (c_{Ond} u_r) \\ \omega(t_1,t_2) &= \sum_{i=0}^{n_{Re}^{(t_1,t_2)}} c_{Re} + \sum_{i=0}^{n_{Ond}^{(t_1,t_2)}} (c_{Ond} u_{Ond_i}) \end{aligned} \tag{5.3}$$

where t_1 and t_2 is the beginning and the end of the measured duration. $\rho(t_1,t_2)$ is the broker's revenue during the same period with c_{Ond} as the on-demand cost per unit time; u as the time usage per request; and n_r as the total number of requests. The operational cost of the broker (ω) is comprised of the cost of the reserved instances and on-demand instances of the broker during

the same period.

We have explained the details of our risk-based optimisation. In the next section, we will evaluate the broker using simulated user requests and Alibaba public cloud trace (91).

5.4 Evaluation

To evaluate our broker model, we simulate a user request environment using the Alibaba cloud datasets (91). The profit level is used as our performance metrics with given user request data. In the rest of this section, we will present the details of the simulation environment setup and each broker model for comparison, followed by result analysis.

5.4.1 Simulation Environment

In this experiment, we focus on evaluating the effectiveness of user requests placements for the same VM instance type from different price schemes. Therefore, we simplify the scenario settings by assuming that users only request for the same instance type. There are two pricing schemes, reserve instances and on-demand instances.

According to a given user request time series data, as each time unit elapses, our simulation environment feeds user requests to the experimental broker models.

We have prepared two sets of user requests data. The random walk user requests data and the Alibaba server trace. The traces consisted of two versions: Alibaba 2017 and Alibaba 2018 which were released during the corresponding year.

- Alibaba 2017: Released in 2017, the trace lasted for 12 consecutive hours on 1300 machines. The trace includes a collocation of online services and batch workloads.
- Alibaba 2018: Released in 2018, the trace lasted for eight consecutive days on 4000 machines. The trace also contains the directed acyclic graph information of the batch workloads.

Both periods of the original datasets are not sufficiently long to evaluate the effect of user request placement for 3-month reserved instances. Therefore, we resample the user requests to be a 3-years' time series data according to the method proposed by Moniz et al. (87). Each request is a tuple of the following format:

```
REQUEST_ID * START_TIME * TERMINATION_TIME
```

Each resampled user request time series becomes an independent scenario for simulation. For ease of presentation, we call the user request from Alibaba 2017 as **Dataset 1**, and the one from

Table 5.1: Summary of User requests times series data for simulation

Scenario Ref	Source Data	Original Length	Resampled Length
Dataset 1	Alibaba 2017	12 hours	3 years
Dataset 2	Alibaba 2018	8 days	3 years

Table 5.2: Statistic description of both datasets

Description	Dataset 1	Dataset 2
Data Points	1,298,775	7,324,831,146
μ/σ	77.08/96.98	49.12/370.43
Min/Max	0/5,450	0/129,215
25 %	18	4
50 %	47	14
75 %	108	48

2018 as **Dataset 2**. A summary is given in Table 5.1.

Datasets 1 and 2 differ from each other in some key aspects. Table 5.2 shows statistical description of the datasets. The size of Dataset 1 is smaller and observably less dispersing than dataset 2. Dataset 2 overall has higher volatility, which should directly impact the profit of each system. We expect the result from Dataset 2 to be worse for the pure reserved strategy and a good challenge to the rest of the systems.

5.4.2 Broker Setting

We established two broker systems based on our models, namely *No risk adjustment* and *Risk-taking*. Additionally, we compare these with three other systems: *Pure reserved*, *Best case*, and *Auto-ARIMA*, which serve as baselines and a typical predictive approach, respectively. Below are the details of each system:

- **Risk-taking**: This is the comprehensive version of our broker system as detailed in Section 5.2. It incorporates all risk factors along with adjustments for risk-taking behaviors.
- **No risk adjustment**: A simplified version of the Risk-taking system, this model does not include adjustments for risk-taking, focusing instead on standard risk normalization.
- **Auto-ARIMA**: Utilizes the *Auto-ARIMA* time series prediction method (125) to estimate future user requests. Any demand that exceeds predictions is automatically handled using on-demand instances.

- **Pure reserved**: A baseline system that only uses reserved instances to fulfill all user requests. It stocks up on new reserved instances whenever there is a pending user request, representing a lower boundary in our system comparison.

- **Best case**: An ideal broker system with foresight into future user requests, allowing optimal planning of VM instance stocking. This system represents the upper boundary in our comparison.

Table 5.3: Component usage in each broker strategies

	Broker Inventory		Optimisation		
System	**Reserved**	**On-demand**	**Risk**	**Adaptive Risk**	**Prediction**
Risk-taking	✓	✓	✓	✓	
No risk adjustment	✓	✓	✓		
Auto-ARIMA	✓	✓			✓
Pure reserved	✓				
Best case*	✓	✓			Exact

* The broker makes optimal decisions based on perfectly accurate prediction for the future user requests.

Table 5.3 shows components in each system. All the brokers will have reserved instances as their main resource. Apart from the pure reserved, the rest of the systems use on-demand instances as a buffer when appropriate. The no-risk adjustment strategy employs a fixed risk when deciding, whereas the risk-taking can alter the risk level according to the cash-on-hand level. The Auto-ARIMA is an automatic variable adjustment time series prediction model which places excessive demands from the prediction onto on-demand instances. Lastly, the best case is the system that produces the highest profit for the given data.

5.4.3 Experimental Results

The results of the simulation show comparisons of accumulated quarterly profit. The profit calculation uses the data taken from Alibaba compute type pricing where the reserved instance gives a 60% static discount. Moreover, the profit is calculated from Equation 5.3. Lastly, the simulation does not account for communication time between users, brokers, and providers.

Figure 5.9 shows the profit of each broker system from Dataset 1. Empirically, both of our systems outperform the pure reserved strategy and Auto-ARIMA model for most parts. In some quarters, our risk-based produce profit levels comparable to the best-case scenario. The statistical breakdown of the result is shown in Table 5.4. Compared to the best case, the overall profit of

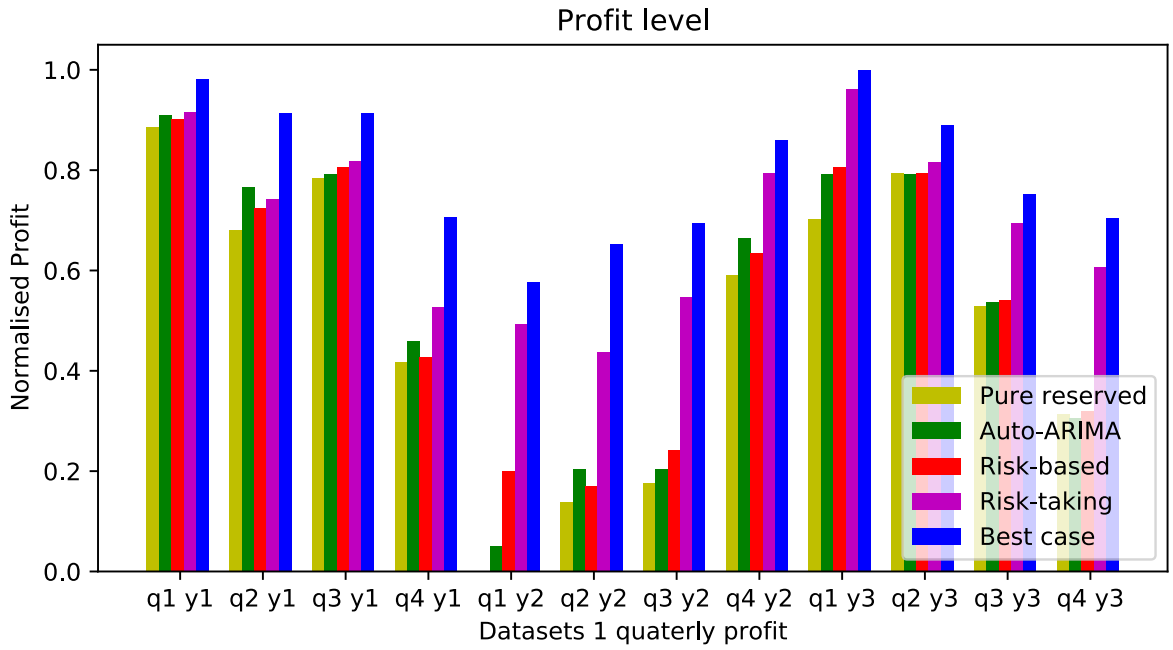


Figure 5.9: The graph shows a comparison of profits between each broker system of the input Dataset 1. The broker system components of each system are shown in Table 5.3. The input is divided into multiple parts of 4 months to illustrate each period's profit level better. Both risk-based systems outperform the pure reserved and reach close to the theoretical maximised profit. The results are consistent throughout the data.

Table 5.4: Comparative values from the base case

Systems	Highest profit(%)	Lowest profit (%)
Risk-taking	96.06	65.85
No risk adjustment	91.51	23.08
Auto-ARIMA	92.50	21.60
Pure reserved	89.85	-4.42

both risk systems has a higher high and higher low than competing systems. Strictly speaking, higher high and low means that it is more likely to earn a higher profit while less likely to lose money.

We have normalised the profit level in a min-max normalisation fashion and compare the results from each system with the best case in Figure 5.11. All systems perform on a similar trend to the best case, with our risk-based outperform the Auto-ARIMA in both Datasets. Without the offloading capability, the reserved only system vastly underperforms the rest of the systems. This is especially pronounced in Dataset 2, where data is volatile.

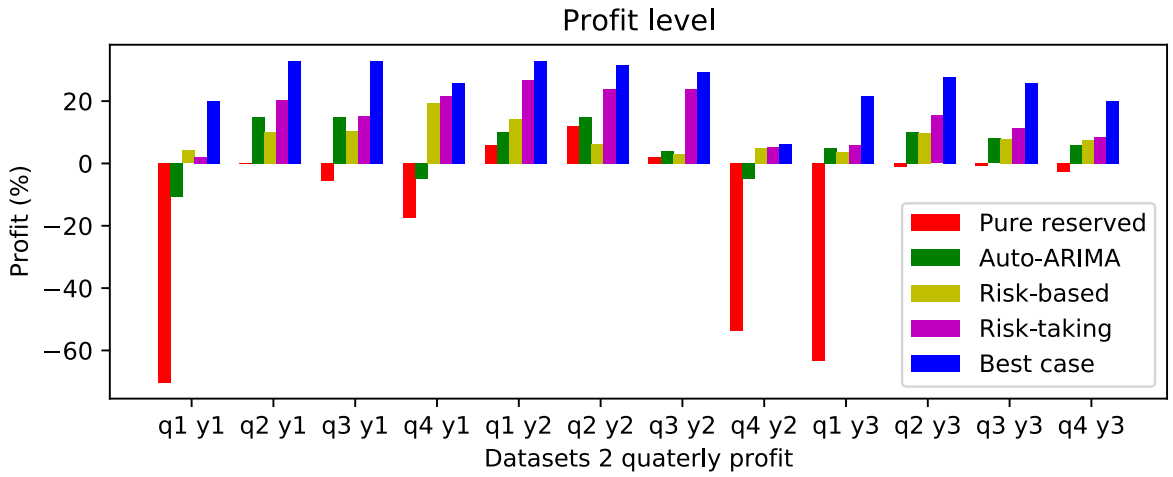


Figure 5.10: The graph illustrates the normalised profit level of input dataset 2. In this dataset, the pure reserved instance struggles to return a profit. On the other hand, the risk-based system manages to stay close to the best case. With the inclusion of risk adjustment feedback, the profit level manages to edge closer to the actual best case values.

Table 5.5: The correlation tests between profit difference of best case and pure reserved and average usage time of users.

Test	Correlation	P-value
Pearson	0.11	0.34
Spearman	0.09	0.45
Kendall	0.06	0.46

Additionally, we also experiment with the risk level such that the broker is taking less risk to allow high utilisation in its inventory. The assumption is that a system with high utilisation of the inventory is likely to have a higher profit. A sample of the first quarter of the second year (q1 y2) period in the Dataset 1 shows that higher average utilisation does not equate to higher profit. The broker employs fewer reserved instances in the inventory; thus, it has to rely more on the on-demand instances, which drive the cost higher. The issue is also known as opportunity cost. To make a good profit and avoid losing money, a broker has to strike a balance between risk and reward, which is the theme of this work.

Apart from the direct relationship between utilization and profit, it might be intuitive to hypothesize that a broker system with a high frequency of short-term usage would better leverage on-demand instances. To examine this hypothesis, we analyze the relationship between profit differences and average cloud usage, as depicted in Figure 5.13. While the correlation

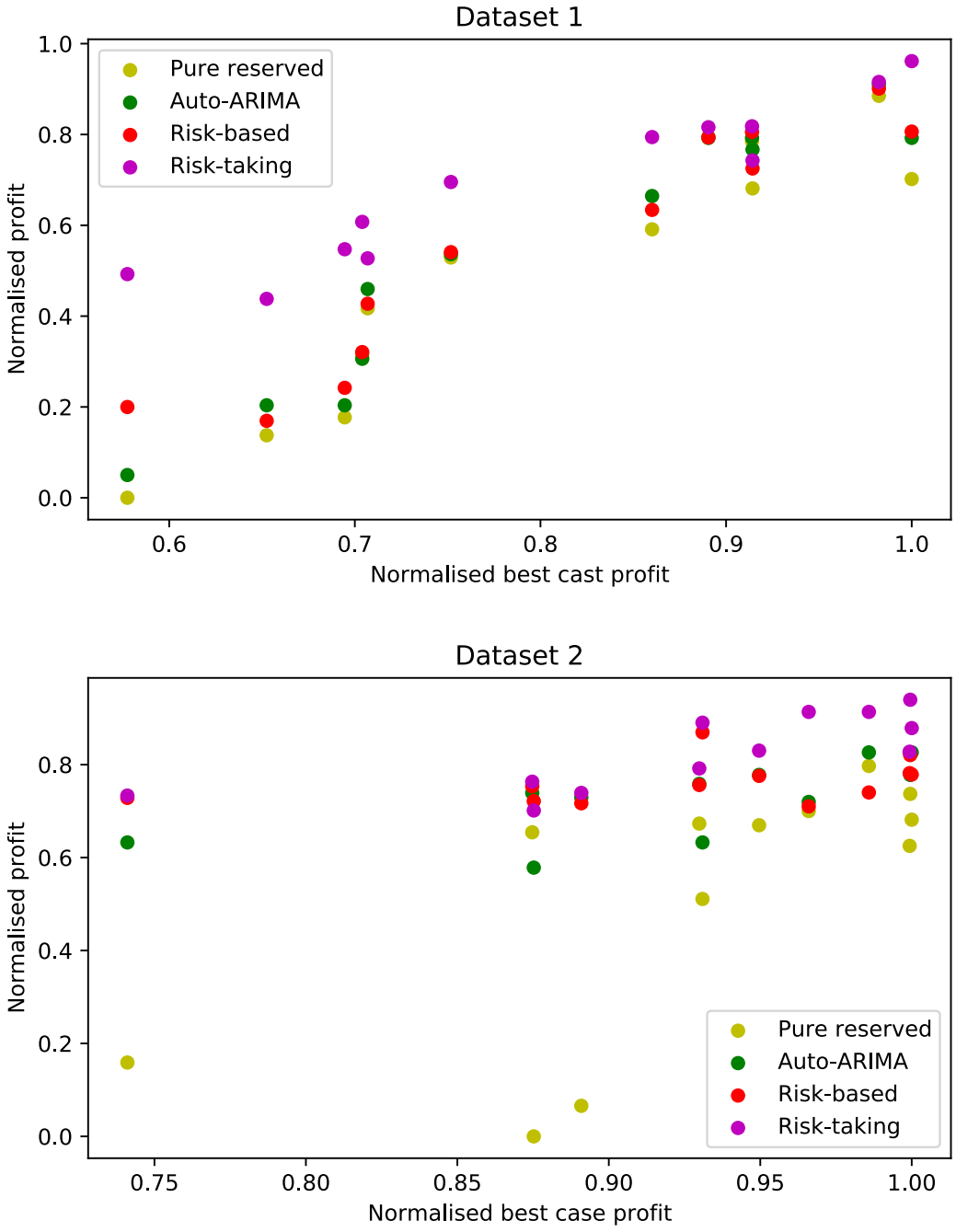


Figure 5.11: (TOP) A scatter plot of a normalised profit level of Dataset 1. The magnitude of the difference between each system is relatively similar throughout the system. Generally, we can see a pattern that the risk-taking outperform the no-risk adjustment system and Auto-ARIMA system for the majority of the period. (BOT) A scatter plot of a normalised profit level of Dataset 2. In Dataset 2, we can see that the Auto-ARIMA and no risk adjustment perform similarly while still trailing behind the risk-taking. Overall, the difference is larger than that of the Dataset 1 result.

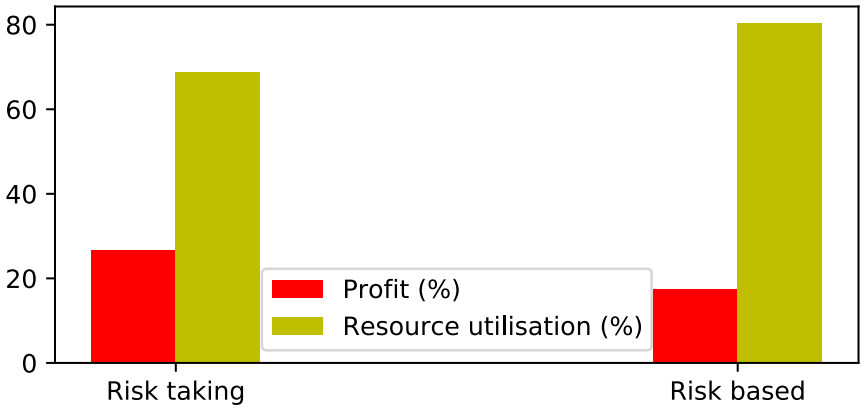


Figure 5.12: The graph shows the profit of the broker in the q1 y2 period in Dataset 1. From the average utilisation of both systems, the risk-based has a higher utilisation lower profit. A higher average reserved instance usage does not always translate to a higher profit.

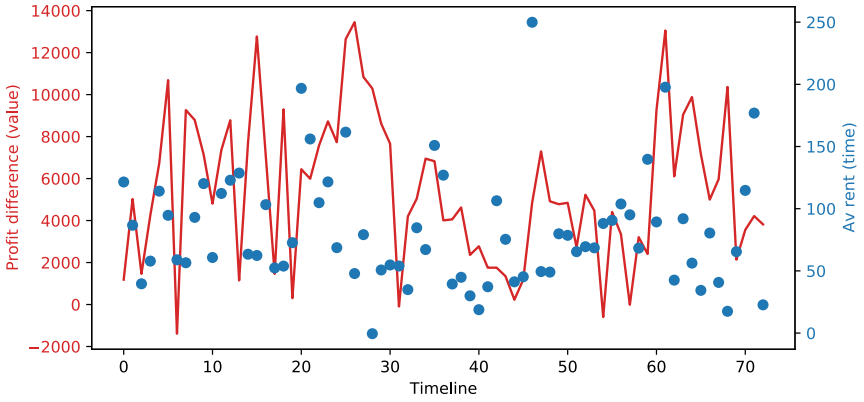


Figure 5.13: The graph of a profit difference between pure reserved and the best case and average usage time of users. A typical assumption for highly frequent small requests data is that it should suit the broker with more on-demand instances rather than the one that relies heavily on the reserved instances. However, from the graph, it does not appear to be the case.

Table 5.6: The correlation tests between profit difference of best case and pure reserved and risk factors.

Test	Correlation	P-value
Pearson	0.71	1.6e-12
Spearman	0.68	1.8e-11
Kendall	0.57	7.8e-13

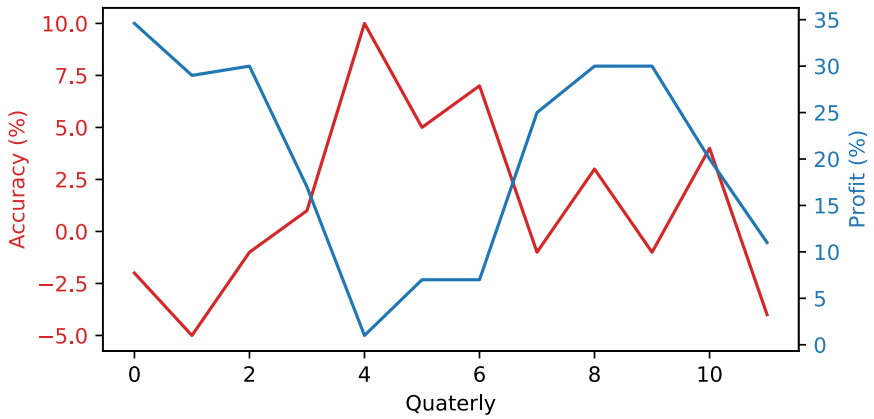


Figure 5.14: The graph compares percentage of over or under estimate the future values of the Auto-ARIMA (RED) and quarterly profit (BLUE). From the result, the profit of a broker seems to be negatively effect by the over-estimation rather than the under-estimation.

appears strong at certain time steps, it is notably weaker at others. The results from correlation tests, detailed in Table 5.5, further substantiate the finding of an exceptionally weak correlation between short-term frequency usage and profit differences across all three tests ([38]). This suggests that average usage duration may not significantly influence the profitability of a broker with a bias towards on-demand instances.

Conversely, the level of risk factors exhibits a more pronounced correlation with profit, as illustrated by the correlation values in Table 5.6. Although this correlation does not conclusively establish risk factors as the definitive indicator of profit, it indicates that they are a significant component of the system's operational success.

The main concern in a cloud broker system that aims toward good profit is the low utilisation of its inventory. The situation occurs when the system overestimates its users' demand and prepares a larger than necessary inventory size. Figure 5.14 shows the over-under estimation of Auto-ARIMA prediction and its respective profit. The overestimation (positive red number) causes the system to lose more money than the underestimation values.

One potential solution to address prediction inaccuracies is to introduce a bias towards underestimation. This bias is already subtly incorporated into the risk factors of our system. By considering the remaining time and size of the inventory, our system is designed to mitigate the effects of overestimation.

However, introducing additional conditions into the cloud broker system can increase its complexity, necessitating a reevaluation of the risk analysis to accommodate these new complexities. Although our current model uses only three risk factors, we have empirically demonstrated that it remains effective, even with this limited number of factors. This effectiveness is achieved without the need for running complex optimization algorithms, which require considerable computational resources and precise regression of user data.

Additionally, while our system is robust in its current form, expanding the number of risk factors could potentially enhance its adaptability and accuracy. This expansion would allow for a more nuanced response to diverse brokerage scenarios that our current model may not fully encompass. By incrementally integrating more risk factors, we can refine our system's predictive capabilities and improve its overall performance in dynamically changing market conditions.

5.5 Related Works

Research on profit maximisation within cloud computing has employed diverse methodologies. Here, we explore several significant studies that align with and inform our own approach.

Resource scheduling in cloud computing has been extensively studied, often assuming that schedulers are privy to both the start and end times of user requests (85; 19; 100). These requests are modeled using probability distributions, with termination times linked directly to their start times. This scheduling strategy is designed to optimise the utilisation of reserved instances, with results verified through mathematical optimisation. However, such assumptions about request timing may not hold in practical scenarios, potentially limiting the real-world applicability of these models.

Amit et al. (28) have examined profit-oriented cloud brokerage with a focus on Quality of Service (QoS) parameters. Their model utilises data communication patterns to enhance utilisation and profitability, similar to our predictive approach. However, they treat resources as fixed costs, whereas our model adapts to variable cost conditions, offering a more flexible cost management approach.

Profit maximization has also been approached through job scheduling and queuing models. Shalmali et al. (4) and others (17) have proposed models that use QoS parameters and service

request distributions to maximize profits, suggesting that effective scheduling can significantly enhance financial outcomes.

Another notable direction in research involves the allocation of cloud resources on reserved instances, with profits optimised based on contract lengths and projected user demand (85; 123). These studies provide mathematical proofs of global optimisation, valuable for validating results under controlled conditions.

5.6 Future Directions

Looking forward, the potential to enhance cloud broker profitability is substantial. Currently, our risk factors are selected manually, which may not be ideal as system complexities increase. Automating the selection of risk factors and considering more dynamic and realistic scenarios, such as multiple tiers of service instances, could significantly improve system responsiveness and accuracy. In this work, risk factors serve as crucial indicators of profit potential; however, their manual selection introduces limitations.

In the forthcoming chapters, we will investigate the application of causal discovery methods, similar to those used in scientific research to discern cause and effect relationships that are impractical to test experimentally. Specifically, we will explore the feasibility of employing an approach akin to the additive noise model within the brokerage context to develop a robust profit model based on the causal relationships between operational parameters and financial outcomes.

5.7 Summary

We have introduced a broker model designed to simplify the selection of VM pricing schemes for cloud users, offering a balance between cost-effectiveness and flexibility. Central to our model is a risk-analysis based decision-making function, optimised through the analysis of a comprehensive real-world dataset from Alibaba. Our findings indicate that the broker's profitability closely mirrors the theoretical optimum, where user demands are perfectly anticipated.

Despite its efficacy in simulations, our current model's reliance on predefined risk factors may not adequately cover all operational scenarios, nor does it fully eliminate the need for human oversight in setting up and adjusting these factors. By leveraging causal relationships, we aim to further refine our profit model, enhancing its predictive accuracy and operational efficiency.

In our next section, we will delve deeper into how causal discovery techniques can be adapted to enhance decision-making in cloud brokerage, potentially revolutionising the way brokers

manage inventory and respond to customer needs.

CHAPTER SIX

AUTOMATIC PROFIT MODEL GENERATOR CLOUD BROKERAGE

From the previous chapter, we have discussed the solution of inventory optimisation using the risk factors. While risk is an intuitive concept, the evaluation of the risk is still a labour-intensive process ([113]; [68]). When the broker system structure becomes convoluted, the risk factor evaluation becomes more complicated and requires more expertise. Here, we would like to design a broker that figure out these *risk* parameters automatically. One approach when dealing with the relation between parameters is to find the correlation. In a sense, a correlation is a connection or relation between multiple parameters with the assumption of their behaviours. Knowing the behaviour is beneficial when the main focus is to understand how each parameter interacts. Our broker system utilises these relations to identify the behaviour of the broker profit under some changes in the associated parameter. Thus, the main focus of the work is to find deeper relations between some parameters and the profit through a causal discovery-like process.

6.1 Automatic Profit Model Generating Approach

The investigation into the underlying relationships within observable data is critical across all research disciplines. The connections discerned between collected data sets can elucidate the behaviour of the system under study. These relationships can be deduced through a variety of methodologies, including correlation and independence tests. In recent years, causal inference has gained traction amongst researchers conducting high-cost experiments. This approach is often employed in cases where repeating experiments to verify results is either prohibitively

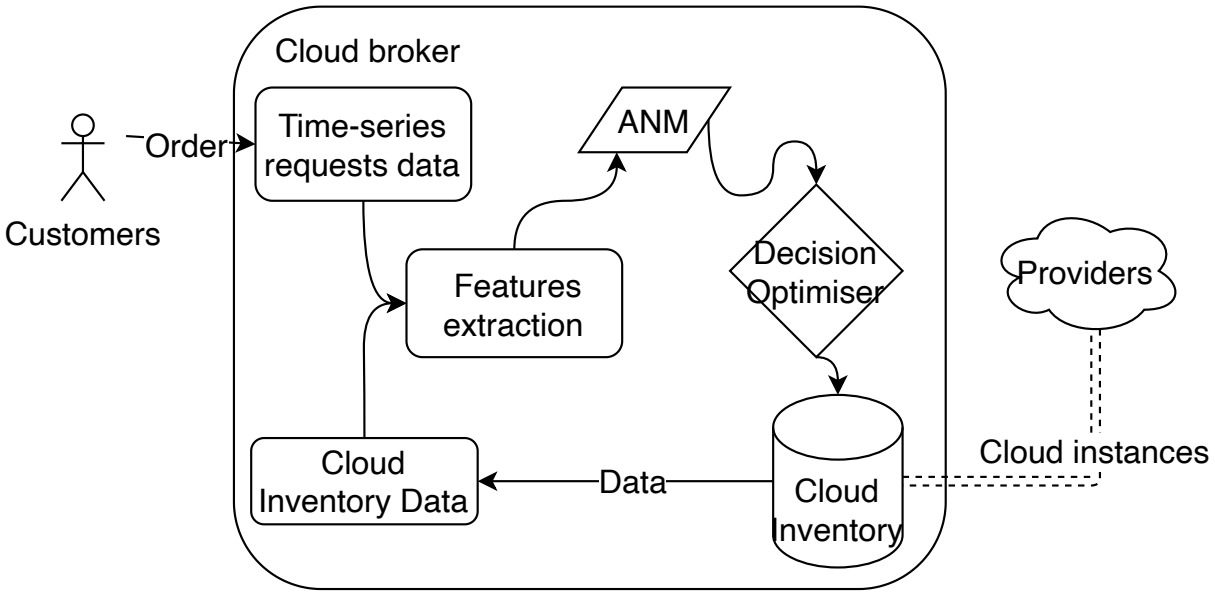


Figure 6.1: A cloud broker operation starts from taking customers orders, process them and then makes adjustments accordingly. The additive noise model broker extracts important data features and selectively builds a model from the associate features. The model is then used in the decision optimiser to make an optimal broker inventory adjustment.

expensive or impossible, facilitating the identification of cause-and-effect relationships between observable data sets (93; 106). A primary advantage of causal inference over other analytical methods is its capacity to discern genuine causal relationships, thereby avoiding the pitfalls of spurious correlations. For instance, although data may show that the average weight of the world's cats has been increasing annually alongside the price of lumber, correlating these trends does not confirm a causal relationship. Identifying genuine causal links between variables is the principal focus of causal discovery.

In this study, we tackle the challenge of optimising broker inventory with a broker model derived from the *Additive Noise Model* (ANM), which serves as the model generator (55). Our broker system employs an ANM-like approach to identify a function that establishes a robust relationship between the generated features from the broker data and the broker's profitability. This relational function then acts as a decision-making tool for broker inventory management. To ensure its accuracy when applied to new data, the validity of this function is confirmed through a comprehensive health check procedure. An overview of this high-level system is depicted in Figure 6.1.

Optimising a broker system with reserved instances at its core requires one of the following. First, predicted data for the optimisation process—alternatively, a system model that estimates an optimised system's behaviour. Figure 6.2 a) illustrates a broker system that uses past data

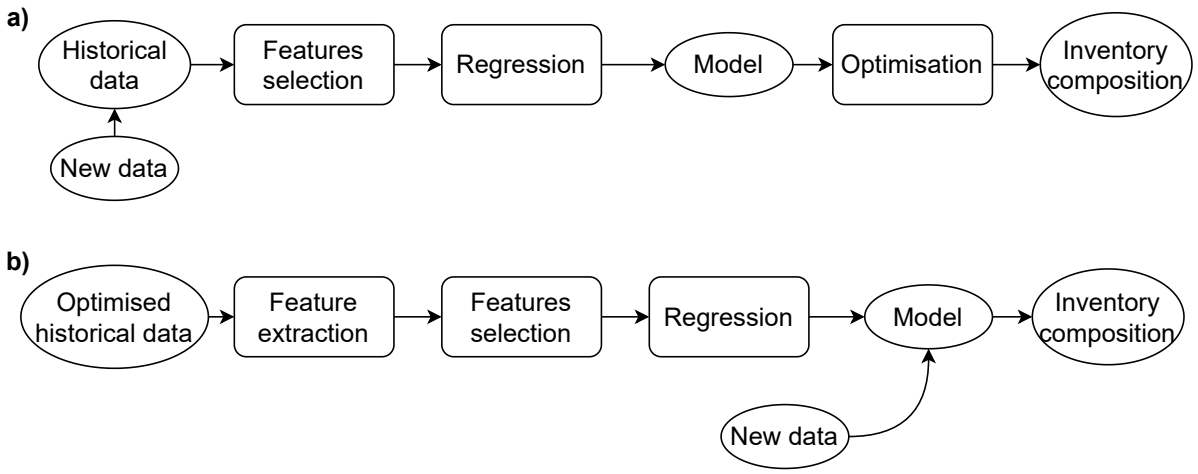


Figure 6.2: Typical workflow of a system that involves future values needs a data estimation process. a) A simple predict-optimised utilised time series prediction and use the predicted data to optimised the broker system. b) A model-based behaviour capture uses a system model built from the data to optimised the system.

to estimate the future values of the desire data and feed the data into an optimisation algorithm. The strength is the simplicity but rely far too much on the estimation. Figure 6.2 b) shows an example of a generated model from optimised past data. The model allows the broker to adjust its inventory at the decision time.

To evaluate our solution, we developed a prototype broker and simulated it with the Alibaba cloud trace dataset (91). The simulation result shows that our ANM-based broker system generates a better result than the typical regression method.

A motivating example of cloud brokerage system with the additive noise model

The additive noise model in open benchmark data is confirmed to be effective on varieties of datasets at finding the causal pair (69). To investigate the effectiveness of the ANM in a broker system, we build a system of inventory composition adjustment function, which act as a decision making process. In this setting, the broker takes only one tier of performance cloud queries. Moreover, the broker employs two types of cloud instances, reserved instances and on-demand.

Let assume that the decision function is

$$(res, ond) := F(ts.mean, ts.value)$$

$$res := ts.mean$$

$$ond := ts.value - ts.mean$$

where ts is a time series of customers query and $ts.value$ and $ts.mean$ is the value of the user time series query and the mean of the sixty previous time steps.

With the simulation using this decision function, the outputs of broker inventory composition, $ond : res$, depends entirely on the customers' query data. Specifically, the ond values is generated from two variables $ts.value$ and $ts.mean$. While, the res values is generated from a single $ts.mean$ variable.

$$res := f_1(ts.mean) \tag{6.1}$$

$$ond := f_2(ts.value, ts.mean) \tag{6.2}$$

With randomly generated time series of customers' queries.

$$ts_t = ts_{t-1} + e$$

where e follow normal distribution, i.e. $N(\mu, \sigma^2)$ with $\mu = 0$, $sigma = 10$, value of the ts never falls below 0. The graphical example of the time series is display in Figure 6.3.

Each query of the customer are terminate after $t_{ter} := N(3,4)$ time step after creation. With the example, we test the causality between parameters using the Additive noise model. If two parameters are cause and effect pairs, then the ANM should produce a causality relationship as the arrow indicated.

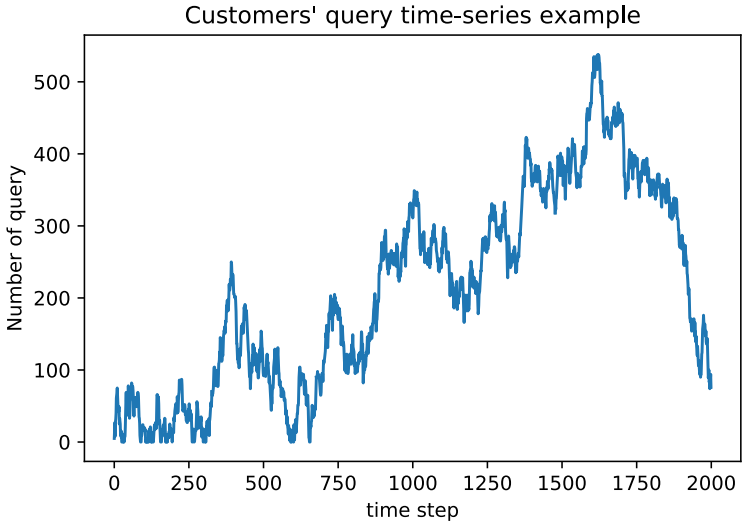


Figure 6.3: The graph shows the customers' demand in this example. The demand follows a random walk model.

The tested pairs are:

- $ts.mean$ and res
- $ts.value$ and res
- $ts.mean$ and ond
- $ts.value$ and ond

From equation 6.1, we can safely assume that if there are relationship functions between parameters of the broker system and the profit. Then, the parameters must include $ts.mean$ and $ts.value$.

Parameters	Causal direction	Parameters	Causal score
$ts.mean$	$\rightarrow$	res	1.041
$ts.value$	$\nrightarrow$	res	0.256
$ts.mean$	$\rightarrow$	ond	0.932
$ts.value$	$\rightarrow$	ond	0.891

Table 6.1: The arrows indicate causal relationship of parameters.

From Table 6.1, three of the pairs show a solid causal score. The positive score indicates the causal direction of the input parameters. If the score is negative, then the causal direction would be reversed. A lower score of closer to zero means that the relationship is inconclusive. The score can go over 1 (or lower than -1) because of the variation of the independent testing. The one-way score could be extremely high, while another direction might be undecided and skew slightly toward the opposite direction, causing the score's overflow. The relationships between variables are illustrated in Figure 6.4.

Next, we are going to add the variable *Profit*, which is calculated from the set of fixed equations below.

$$Profit := revenue - cost$$

$$revenue := f_{re}(ts.value)$$

$$cost := f_{co}(ond, res)$$

substitute with equation 6.1, we have

$$cost := f_{co}(f_2(ts.value, ts.mean), f_1(ts.mean))$$

Assuming that the broker has no control over the customers' queries, we can ignore the *revenue* parameter values as a relationship to the profit for now. With the ANM, we can see that even after

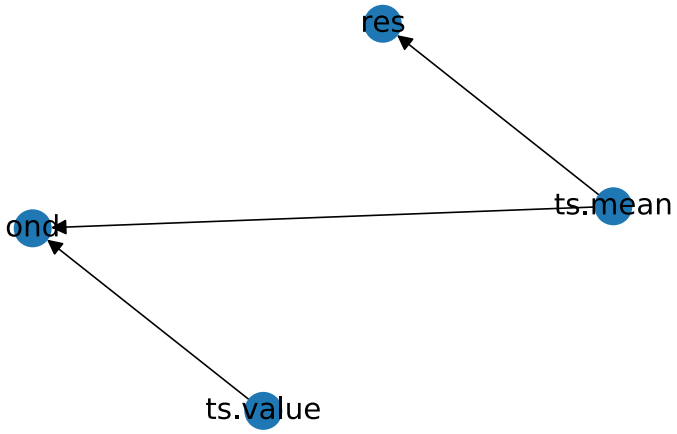


Figure 6.4: The graph shows the overall causal structure of created by the causal discovery between parameters. In this example, *ts.mean* affects both reserved instance and on-demand instance but only the *ts.value* affect the on-demand.

compositing functions. We can still trace back the profit to the original time series of customer queries.

From Causal Discovery with Cascade Nonlinear Additive Noise Models, we can assume that it is possible to infer the intermediate causal variable x_2 from the original model of $x_1 \rightarrow x_3$ to become $x_1 \rightarrow x_2 \rightarrow x_3$.

Thus, from

$$ts.mean \rightarrow Profit$$

$$ts.value \rightarrow Profit$$

We can safely assume that there exist functions which take specify inputs such that they associate with the profit value.

$$f(ts.mean, ts.value) := ond$$

$$g(ts.mean, ts.value) := res$$

The system we have developed deduces functions and links input parameters, which dictate the composition of the inventory, to data reflecting optimised profits. The aim is for the broker to employ a decision function that aligns with historically optimised profits.

In our approach, we draw parallels to Auto-regressive causal discovery methods or Vector Auto-Regressive (VAR) systems. However, rather than utilising the commonly employed Granger causality within VAR frameworks, we incorporate the additive noise model as an alternative due to its robust function estimation capabilities. Traditional Granger causality methods primarily

focus on identifying relationships within time series data through lagged terms. This aspect, however, does not constitute our main area of interest. Instead, our emphasis is on understanding and modeling the functional relationships that directly influence the decision-making process in broker inventory management.

6.2 Stacked Cloud Brokerage Strategy

A common strategy among cloud brokers is to purchase cloud instances in bulk at discounted rates and resell them at a profit. Traditionally, brokers match fixed-type requests with corresponding cloud instances to sidestep issues related to security and performance overhead.

Recent studies suggest that the performance overhead associated with running multiple virtual machines on a single cloud instance is minimal enough not to significantly impair performance (76). On the security front, brokers can implement various security frameworks to address and monitor potential vulnerabilities, thereby maintaining customer trust (81). Consequently, we propose that utilising a single cloud instance for multiple user requests could provide an efficient solution for brokers aiming to enhance the flexibility of their inventory management.

However, a critical concern that a broker must address is ensuring that this shared operation model does not compromise the expected performance levels. The field of performance capture in cloud instances is a vigorously active area of research (111), offering insights that can be leveraged to mitigate potential drawbacks.

One viable approach is to classify virtual machine (VM) specifications into performance tiers. By adopting a tiered system similar to that used by cloud providers, a broker can offer users performance categories rather than specific specifications. This system ensures a minimum performance standard while facilitating more efficient inventory management. Each performance tier is designed as a linear combination of lesser tiers, allowing for various configurations of cloud instance management. This flexible setup enables the broker to optimize configurations to meet higher target indices such as utilization and profitability. Utilizing a comprehensive benchmark database, brokers can accurately quantify the performance levels of cloud instances, thereby ensuring a consistent cloud experience for customers.

Assuming that there are the following performance tiers:

$$[p_1,\ p_2,\ p_3]$$

where the performance level of each tier is equivalent to

$$p_3 \equiv p_2 + p_1$$

$$p_2 \equiv 2p_1$$

where p_2 is about two times as powerful as the p_1. If the customers demand a lot of p_1, the broker can employ a higher performance tier and substitute two of the p_1s with one p_2. Since the price per performance is cheaper at higher tiers. Therefore, in a certain scenario, a broker using a higher performance tier can benefit the profit. On the other hand, if the demand of the p_1 does not continue, then the broker lose an opportunity on the already purchased p_2 instance. This creates a decision-making scenario on which is the best configuration of the cloud inventory.

Shared virtual machines business model

There is ample evidence to suggest that sharing hardware resources is a common practice in cloud computing businesses. Time-sharing, for instance, has been proven to be an effective strategy, even when considering the quality of service constraints (98). Thus, it is feasible to offer a brokerage service where reserved cloud instances are subdivided and shared among multiple customers.

While the use of shared instances can raise security concerns, these can be adequately addressed through the application of established security frameworks (18). Additionally, the performance issues associated with sharing cloud instances can be effectively managed to ensure compliance with guaranteed performance levels or service level agreements (29).

Consequently, the practice of hosting multiple tenants on a single piece of hardware or a single cloud instance does not inherently pose insurmountable challenges. This approach can offer both economic and operational efficiencies, making it a viable option in the strategic toolkit of cloud brokers.

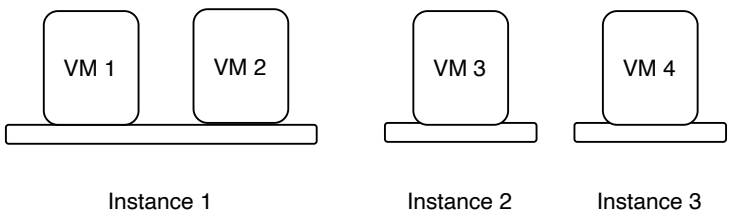


Figure 6.5: Instance 1 of a cloud broker is a more powerful cloud instance. Therefore, it is able to accommodate more virtual machine images than instance 2 or 3. If we assume that all virtual machine is of the same performance level, then instance 1 can do the same amount work as instance 2 and 3 put together.

Stacked Hosting of Instances and Virtual Machines

As illustrated in Figure 6.5, a more potent cloud instance (Instance 1) can accommodate a greater number of virtual machines with similar performance requirements, provided all service requirements are met. Typically, higher-performance instances offer better value for money, potentially increasing the broker's profit margins. However, the initial higher costs associated with these powerful instances must be recuperated, which may not always be guaranteed.

Offloading Customer Queries

Assuming that a single reserved instance can be divided and shared among multiple customers, the subsequent challenge is to capitalize on this arrangement.

Financially, an idle reserved instance represents a misallocation of resources within the broker's inventory. Minimizing the number of these idle instances is crucial for enhancing profitability. One effective strategy employed by brokers involves offloading some customer queries to more expensive instances (84; 123). This approach has been demonstrated to be successful, optimizing the utilization of available resources and improving overall financial outcomes.

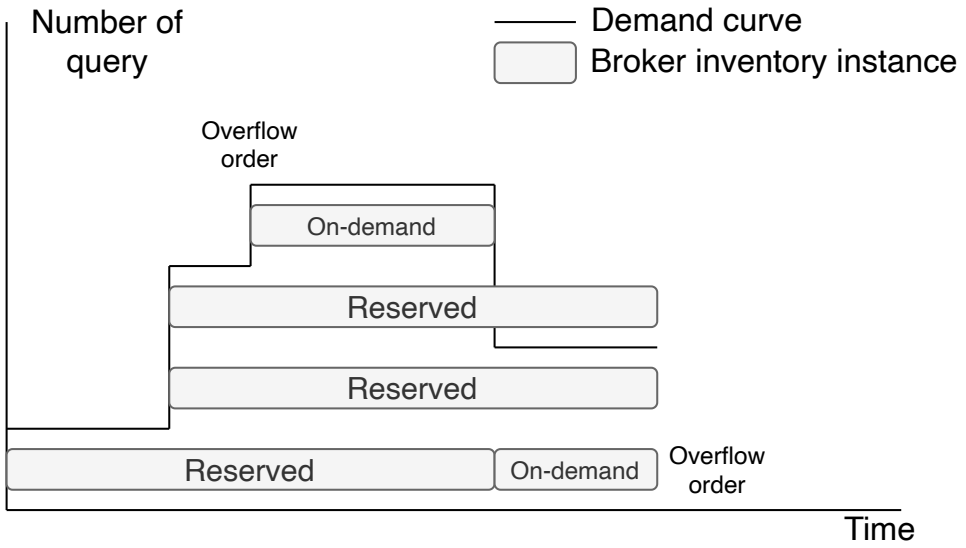


Figure 6.6: Some queries that cannot be accommodate by the existing cloud instances in the inventory can be offloaded to another type of cloud instance.

Figure 6.6 illustrates the query offload to on-demand instances. The overflow queries are offloaded to the on-demand instance on some conditions, i.e. outliers.

A broker must choose the best combination (cloud instance performance level, pricing schemes) of cloud instances for the given customer queries data from the two strategies. We start formulating the problem by explaining the structure of cloud inventory and its definition.

6.3 Brokerage System Profit Model

We simplify the inventory optimisation process within the broker system into a straightforward equation involving profit and inventory variables for ease of discussion. Let P represent the profit function of a broker system, defined as:

$$P : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$$

Here, $\mathbb{R} \times \mathbb{R}$ represents the revenue and cost respectively, with the function returning a real number that indicates the profit value.

To maximise profit, it is crucial to understand the specifics of each component involved. Assume n and m represent the number of user requests and the number of cloud instances in the broker's inventory during a calculation period $T = [t, t_k]$, respectively. We analyze these within a specified window of time to focus the calculations on a particular period of interest. The period T serves as both a reference point and a boundary for comparing results.

The broker's revenue can be calculated using the equation:

$$x = \sum_{i=1}^{n} p_i \cdot U_i$$

where U_i denotes a user request i and p_i is the payment received from the respective user. The cost is given by:

$$y = \sum_{i=1}^{n_t} ResvC_i + \sum_{j=1}^{m_t} OndC_j$$

Here, n_t and m_t represent the number of new instances added to the broker's inventory at time t, with $ResvC$ and $OndC$ being the average costs of reserved and on-demand instances at that time, respectively. Alternatively, the cost can be generalised as:

$$y = \sum_{i=1}^{m} (c_i \cdot INST_i)$$

where $c_i \cdot INST$ is the cost of each instance in the inventory during the time T.

Optimising profit fundamentally boils down to the configuration of cloud instances in the inventory. However, as previously noted, forecasting far into the future requires accurate predictions of demand well beyond the current data points. Therefore, this work introduces a broker model that infers profit relationships and makes just-in-time decisions based on dynamic data.

Constructing a broker profit model begins by structuring relevant data: customer order data, inventory data, and profit are denoted as **x**, **y**, and **z** respectively. Assuming an optimised system, a function $G(\mathbf{x}) = \mathbf{y}$ exists such that the profit function $P(\mathbf{x}, \mathbf{y}) = \mathbf{z}$ yields maximum profit. Thus, we can integrate the inventory generating function G into the profit equation:

$$P(\mathbf{x}, G(\mathbf{x})) = \mathbf{z} \tag{6.3}$$

Assuming that G and its inputs are identifiable, the broker should then operate at an optimal profit level, with **z** presumed optimal from the identification of G.

To validate this claim, we explore the broker system dynamics. At any given time t_k where $t_k \in T := [1, K]$, **y** is computed from G as follows:

$$\mathbf{y}(t_k) = G(\mathbf{x}(t_{k-a}, \ldots, t_{k-1}))$$

The decision to adjust broker inventory is expressed as:

$$\arg\max_{y} AGG_k P_{t+k}(\mathbf{y}_{t+k}, \mathbf{x}_{t+k})$$

where $\arg\max_y AGG_k$ represents the values of **y** from t to $t+k$ that maximise the broker's profit, given **x** as inputs.

Several challenges arise in this optimisation process. Firstly, the $\arg\max_y$ operation demands highly accurate regression data extending far into the forecast horizon. Secondly, the regression process itself requires meticulous preprocessing of input data.

Even with a high degree of accuracy, the system might yield suboptimal profits since the regression pairs between **x** and **y** assume the task of matching inventory parameters with demand. Other, hidden parameters could also influence profits, indicating potential disconnects within the system.

Alternatively, employing correlation tests to identify relationships between profit and system data (**x, y**) may lead to redundant insights since **z** is derived directly from **x** and **y**. For instance:

$$\text{start}[t, t_k] \propto \mathbf{z}$$

$$\text{res}[t, t_k] \propto \mathbf{z}$$

Correlations such as the number of starting queries and reserved instances are likely to correlate

with profit. Consequently, extracting features directly from the data may prove more effective (21; 67). Thus, constructing a profit model of the brokerage system emerges as a viable solution.

Typically, model generation and selection are manually conducted, informed by prior knowledge of the system. Chosen models are evaluated using a fitness metric to confirm their accuracy.

Furthermore, causal discovery aims to solve the identifiability issue, i.e., model selection, within a brokerage context where multiple variables interact and influence profitability. Understanding these relationships is crucial for comprehending broker behaviour and variable interactions. Consequently, in the forthcoming section, we introduce a broker system that utilises causal discovery and feature synthesis to pinpoint connections to the profitability of a cloud broker.

6.4 System Workflow & Architecture

This section elucidates the foundational concepts and methodologies underpinning our automatic model generator brokerage system, centered around the Additive Noise Model (ANM). This model plays a crucial role in identifying key data entities that influence the broker's target outcomes—primarily profit—while constructing broker inventory composition functions. We begin with an overview of the broker system, outlining the high-level concept and the mechanisms involved.

6.4.1 System Workflow & Components

The system initiates from the data perspective. Broadly, our system manages two types of data: optimised (historical) data and actual (operational) data. Optimised data comprises past data with precise measurements used to "figure out" solutions, whereas actual data pertains to the information currently employed in system operations.

ANM serves as both a model selector and a parameters filter. This method enhances traditional regression by incorporating independence noise and bi-directional hypothesis independence testing, yielding directional *causal* pairs and their behavioral models. We deploy ANM to sift through and filter out irrelevant features generated from raw operational data, subsequently building models from the relevant feature pairs aligned with the broker's target, i.e., profit. The workflow of our broker system is depicted in Figure 6.7.

We will now describe the main components of the system:

- **Feature extraction**: As a preprocessing step before auto-model generation, feature extraction calculates multiple data features to capture the data's characteristics. These features, rather

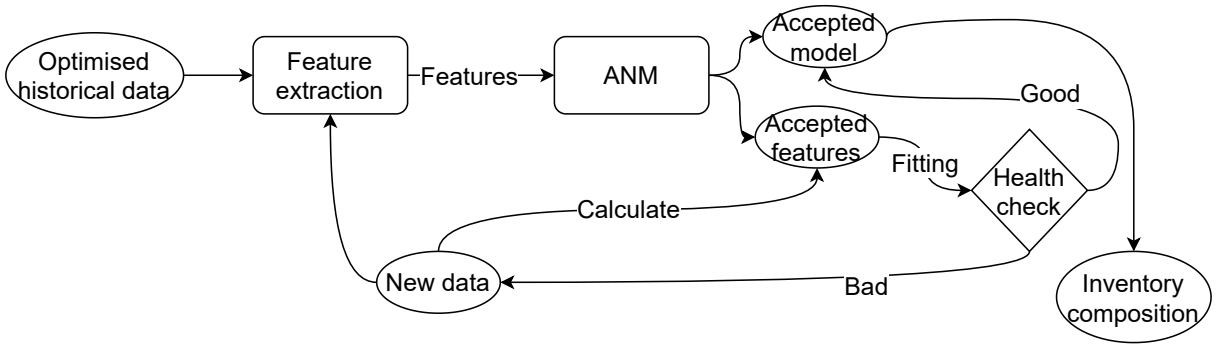


Figure 6.7: The automatic model system introduces an additional layer of hypothesis testing from the generated features to automate feature selection and filtering processes.

than raw data, are used in the additive noise model's regression method to prevent mere linear combinations of data. Models trained with data features are generally more accurate than those trained with raw data.

- **Auto-model generator**: This component employs the causal inference method (ANM), starting with a model estimator, in this case, the Gaussian process. The Gaussian process, augmented with added noise and hypothesis testing, forms the crux of the ANM. Additive noise, independently generated and added to the data (i.e., feature entities), facilitates the verification of bi-directional hypothesis testing for causal direction. Hypothesis testing determines whether two datasets are independent of each other. The broker system's model—a generated function—mimics the system behavior based on validated relationships between feature entities and profit targets. This model informs the broker's inventory composition decisions for incoming data and can be adjusted or replaced if health check criteria are not met.

- **Inventory composition**: This component generates corresponding inventory data in accordance with the model, dictating the broker's procurement decisions regarding which cloud instances to include in the inventory. Essentially, the inventory composition represents the system's output.

- **Model health check**: This critical step ensures the model's continued accuracy following inevitable shifts in the data distribution used for model construction. When data distributions change, maintaining model accuracy becomes challenging. Thus, an additional step is implemented to verify that the deployed model in the broker system remains correct.

6.4.2 Feature Synthesis

Machine learning algorithms are a method of taking inputs and producing desirable output results. Several improvements can be made to help train the algorithm, one of which is the feature engineering process. We will not dive too deep into discussing the benefit of feature engineering or how tedious the process can become. In this work, the feature engineering process is assumed by the feature synthesis calculation of the data.

The feature synthesis is an essential step in providing the model generator with candidates. Figure 6.8 shows the sliding time series corresponds to the change from raw time series to aggregated features.

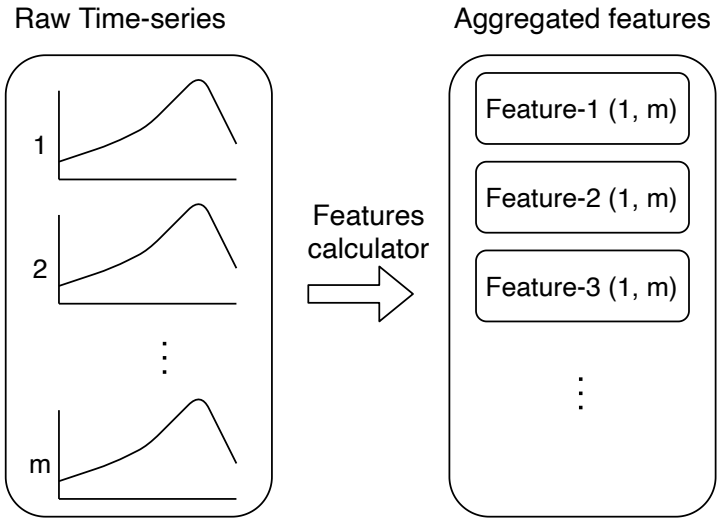


Figure 6.8: Each moving window of a raw time series is calculated to a single data point in multiple aggregated features.

We can split the real data into two, the customers' data and broker data. Customer's data are a collection of order requests that are processed chronologically. The broker data is the inventory data, i.e. number of reserved instances or performance level. The inventory data is the data that directly calculate the running cost of the broker.

The broker defines the frequency of the process. For example, if the broker process customers order every second. Then, the data are collected and process every second. Therefore, the working time-step of a broker is one second. A shorter time-step leads to shorter responds time to the customer, lower data to process per time, more missing data etc. On the other hand, a longer time-step produces a larger chunk of data but a longer delay in customer response time and decision making. There is no definite rule for the duration of time step. Nevertheless, we can reference approaches that deals with the bin size setting, such as Rice's rule and Sturge's

rule (130).

Each customer data can be represented as below:

```
USR = {ID,Start,Spec,TIME} | {ID,Terminate,TIME}
```

Then, multiple time series data can be obtained for the system, e.g., the number of starting requests, the number of each specification etc.

We can divide the customers time series into a tier of specifications:

$$TS_{p1} = (ts_i, ts_i = \sum_{i=0}^{n_t} Start_i)$$

$$TS_{p1} = (ts_i, ts_i = \sum_{i=0}^{m_t} Terminate_i)$$

$$\dots$$

$$TS_{pn} = (ts_i, ts_i = \sum_{i=0}^{n_t} Start_i)$$

$$TS_{pn} = (ts_i, ts_i = \sum_{i=0}^{m_t} Terminate_i)$$

Similarly, the inventory time series can be constructed from the snapshot of the inventory itself, i.e. the composition, the availability etc. The time series of the broker inventory are:

$$ITS_{p1}^{on} = (its_i, its_i = \sum_{i=0}^{n_{on}} on)$$

$$ITS_{p1}^{resv} = (its_i, its_i = \sum_{i=0}^{n_{resv}} resv)$$

$$\dots$$

$$ITS_{pn}^{on} = (its_i, its_i = \sum_{i=0}^{n_{on}} on)$$

$$ITS_{pn}^{resv} = (its_i, its_i = \sum_{i=0}^{n_{resv}} resv)$$

Each of the time series is pass through feature extraction processes to create aggregated time series features. The features used in this work are shown in Table 6.2. All of the features used in the system are parameters independent, meaning that they do not require additional user inputs. This is to eliminate the unnecessary parameter adjustment in the system. However, any custom features can be used in the system as long as they can be aggregated to the same length as the target. For example, time series

$$TS_{p1}(t) = (x_i, x_i = \sum_{i=0}^{n_t} Start_i(i))$$

Feature	Description
`absolute energy`	The absolute energy of the time series which is the sum over the squared values
`max`	Highest value of a time series
`mean`	Mean value of a time series
`min`	Lowest value of a time series
`kurtosis`	the kurtosis of x (calculated with the adjusted Fisher-Pearson standardised moment coefficient)
`root mean square`	the root mean square (rms) of a time series
`sample entropy`	sample entropy of a time series
`standard deviation`	The standard deviation of a time series

Table 6.2: A set of extracted time series features

$$TS_{p1}(t+1) = (x_i, x_i = \sum_{i=0}^{n_t} Start_i(i))$$

...

are converted into an aggregated feature time series

$$TS_{mean} = (mean(TS_{p1}(t)), mean(TS_{p1}(t+1)), ...)$$

The calculated features set is the starting parameters for the auto-model generator.

6.4.3 Model Generation Based on the Additive Noise Model

In this work, we adjust the inventory based on the model generated from the Gaussian process within the additive noise framework. As previously mentioned, directly matching regression pairs can be arduous and may sometimes fail to meet the system's objectives. Consequently, we opt to use substitute parameters in the form of data features, selecting some associated features via bi-directional hypothesis testing.

Rather than employing raw data, we utilise data features. Feature extraction is pivotal in any machine learning algorithm as it reduces data redundancy and generally enhances both the speed and accuracy of the algorithm (71). Furthermore, we employ feature extraction to prevent the linear combination of regression data in our system, which is essential as the Gaussian process cannot be performed with data that are mere combinations of other data. We utilise the *identification* property of the additive noise model to refine the generated feature space, ensuring only pertinent features are included in the model.

Consider a feature set E:

$$E = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$$

which comprises entities containing features of our broker data from the feature extraction process. Each $\mathbf{e}$ represents an individual feature derived from $\mathbf{x}$ and $\mathbf{y}$. For each $\mathbf{e}$, there is a corresponding feature function k_i such that:

$$k_i(\mathbf{ts}) := \mathbf{e}_i \tag{6.4}$$

where $\mathbf{ts}$ is the time series data of customers and broker $(\mathbf{x}, \mathbf{y})$.

We postulate the existence of a function F_i which associates features with profit:

$$\mathbf{z} := F_i([\mathbf{e}_m, \ldots, \mathbf{e}_k]) + n_i \tag{6.5}$$

such that if a single directional confidentiality test from $e_i \in E$ to $\mathbf{z}$ is accepted, then the target $(\mathbf{z})$ should behave in accordance with model F_i.

The feature filtering and model selection process is executed concurrently. The function F_i is estimated using the Gaussian process, and the additive noise n_i is independently generated from the Gaussian distribution. The additive noise model approves the functional model F_i if the independence test between $\mathbf{z}$ and $\{\mathbf{e}_m, \ldots, \mathbf{e}_k\}$ is unidirectional. Dependent models and features are discarded. Once all feature entities are evaluated, the model described in equation 6.5 becomes the profit model for our broker system.

Although capturing the profit behaviour from all possible parameters might be challenging, the profit value is calculated from a fixed function of two sources of variables, the customers' variables $(\mathbf{x})$ and broker inventory variables $(\mathbf{y})$:

$$\mathbf{z} = P(\mathbf{x}, \mathbf{y})$$

Since P is a fixed function, $\mathbf{y}$ can be derived from the common target variables $\mathbf{z}$ following the causal model of cascade causality (15). In our broker model, the cascade causal is linked with the connected profit function P. Therefore, if $\mathbf{z}$ is optimised and behaves according to the feature entities chosen with the additive noise model process, then for each y in $\mathbf{y}$ there exists a function from the regression process such that:

$$f_i([\mathbf{e}_m, \ldots, \mathbf{e}_k]) = y$$

where y is the inventory composition component of the system, i.e., the number of reserved

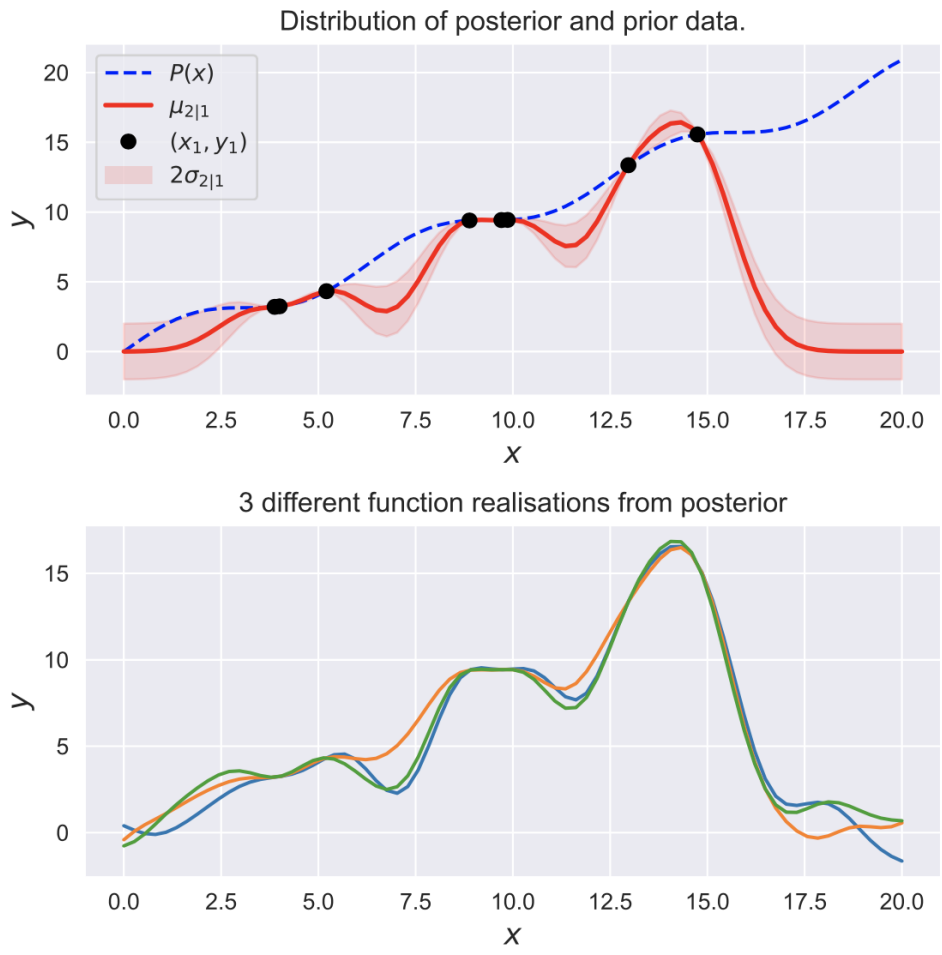


Figure 6.9: (Top) $P(x)$ represents the profit data, with black dots illustrating the samples for the Gaussian process. The Gaussian process function, depicted in red, is defined by mean and standard deviation. (Bottom) This graph displays three sample functions derived from the distribution, serving as single-point "prediction" models.

instances running in a broker. Thus:

$$\mathbf{z} = P(\mathbf{x}, f_i([\mathbf{e}_m, \ldots, \mathbf{e}_k]))$$

This equation is an extension of the profit model built from the additive noise model.

It is important to acknowledge that feature extraction functions are not inherently bijective, which implies that the uniqueness of the optimal solution cannot always be guaranteed. Nonetheless, given the existing initial and boundary conditions, the solutions are generally sufficient for practical applications. These boundary conditions are essential for managing reserved instances. For instance, the number of reserved instances in the inventory at any given step should not decrease compared to the previous step unless they are due to expire.

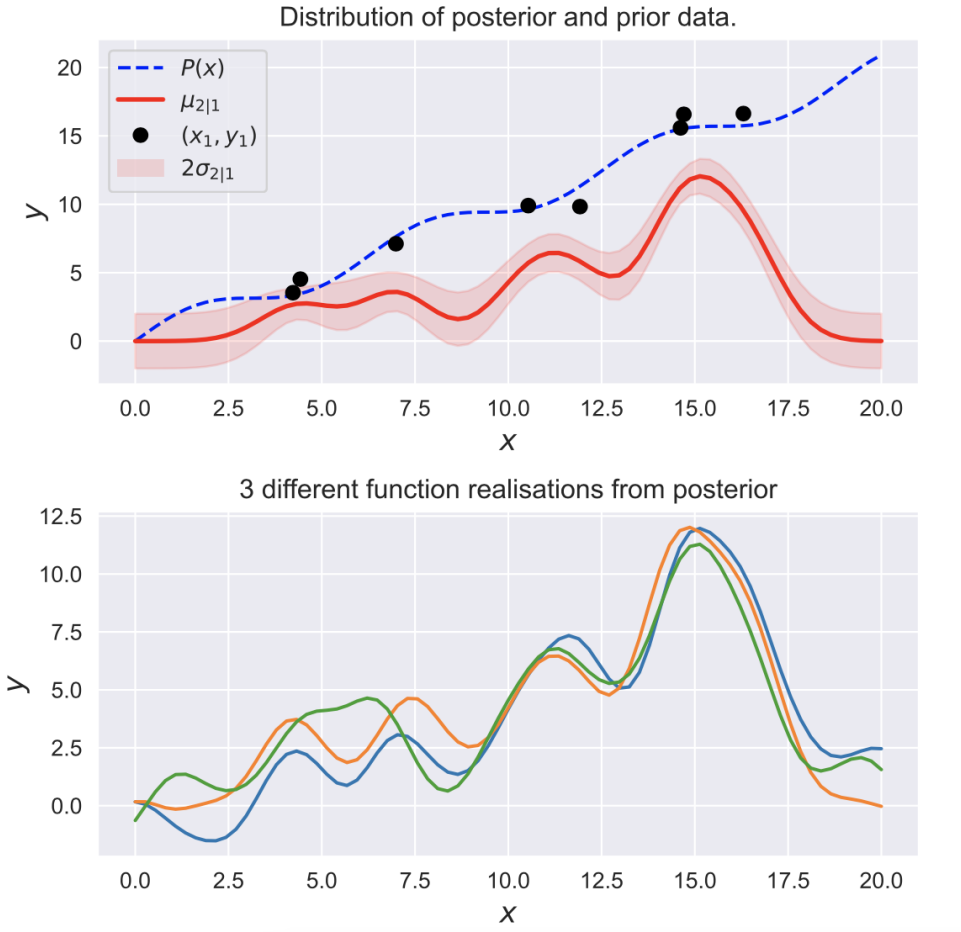


Figure 6.10: (Top) $P(x)$ represents the profit data, with black dots indicating the observation sample with added noise. (Bottom) Similar to a normal Gaussian process, the sample functions are depicted as single-point predictions.

From Figure 6.9, it is evident that prediction functions derived from the Gaussian process are not unique. Various approaches exist for selecting the prediction value. A common method involves averaging all sample values, which tends to converge towards the mean function. Alternatively, the expected loss can be minimized, typically accomplished via a loss function $\mathscr{L}(y_{\text{true}}, y_{\text{predict}})$, as shown below:

$$R_{\mathscr{L}}(y_{\text{predict}}|\mathbf{x}_*) = \int \mathscr{L}(y_*, y_{\text{predict}}) p(y_*|\mathbf{x}_*, \mathscr{D})\, dy_*$$

The optimal prediction value minimizes this expected loss:

$$y_{\text{best}}|\mathbf{x}_* = \underset{y_{\text{predict}}}{\operatorname{argmin}} R_{\mathscr{L}}(y_{\text{predict}}|\mathbf{x}_*)$$

where y_* represents a single point prediction from the test input $\mathbf{x}_*$, and y_{predict} is the prediction from the sample functions.

Added noise in observations deviates the prediction functions, thus altering the inverse functions (if they exist) from their normal counterparts. The introduction of noise is a key component of the additive noise model for causal discovery. In this work, we apply similar principles to uncover relationships between parameters and profit, avoiding accidental correlations that do not genuinely influence profit outcomes.

It should be noted, as depicted in Figure 6.10, that the variance at observation points is non-zero, indicating that the sample functions do not pass through the observation points. In predictive scenarios, noisy observations, or inaccurate measurements, can significantly impair accuracy. However, in our system, the noise variable is reintroduced only after Gaussian function approximation during the relationship inference phase, thus preventing any loss in accuracy due to noise.

6.4.4 Independence Criterion & Health Check

A generated broker profit model cannot always stay accurate or effective once newer data are received. Thus, it is helpful to adapt or replace the outdated model during the broker operation.

If F_i is the current broker model from the model generator at time step t, in the next time step, $t+1$, there is new data $\mathbf{x}_{new}$ which is also encoded into new feature entities.

The health of the model can be measured or evaluate using an accuracy metric. In the case of the additive noise model, however, the model's health can be measured with the same independence criterion test.

With new data, there are new time series data set which then be encoded into feature entities. For each e_i in E from the original model, the values of E are updated.

$$ind(e_i, \mathbf{z}) = \text{Independence score}$$

The *ind* score ranges from 0 to 1. For simplicity, let x, y be the two parameters which we want to measure the HSIC. If the model is from $x \to y$, then $y_{predict}$ is the prediction value from the Gaussian process model of x. The independent score is calculated from $HSIC(y_{predict} - y, x)$. And the bi-directional score are calculated as $ind(y,x) - ind(x,y)$ (Value : 1 if $x \to y$ and -1 if $y \to x$)

If the bi-directional *HSIC* score concludes differently from the original model, i.e. anything but acceptance in the direction from e_i to $\mathbf{z}$, then the model is no longer valid. Thus, the system triggers the model generator to find a more suitable model and feature entities to replace the old set.

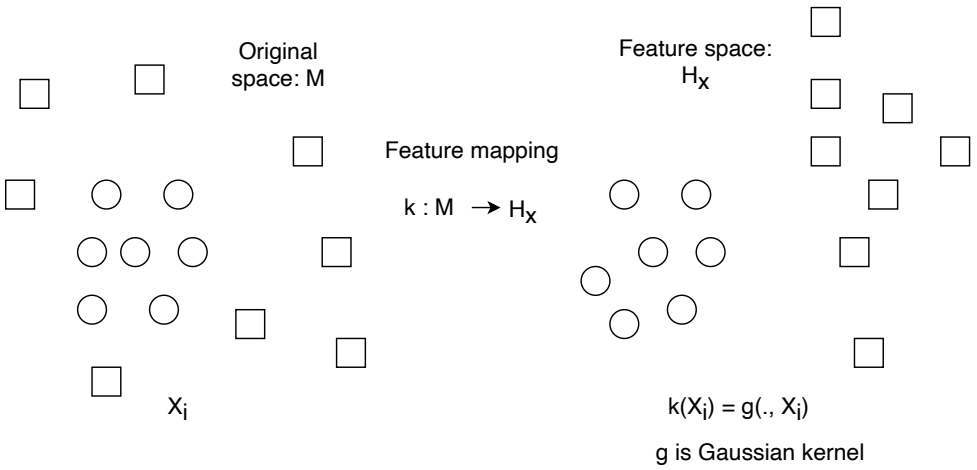


Figure 6.11: Hilbert-Schmidt independence criterion use a kernel method to transform data from its original space to a feature space which is easier to identify the independence property.

The model checking has a relatively low computational complexity compared to the optimisation or regression. Therefore, if the longevity of the model is high, then the system's overall complexity is also low, which is beneficial to the system.

We have now introduced all the main components of the system. The pseudo-code of the broker system is provided in the Algorithm 1.

6.5 Experimental Results

Operating a cloud broker involves substantial financial implications, making direct experimentation costly. Consequently, simulating the broker system serves as the preferred method for testing its effectiveness. This section details and discusses the experimental results of the automatic profit model broker system using a broker simulation.

6.5.1 Time Steps & the Period of Calculation

The period considered for the calculation is the specific length of data utilised by the system, often referred to as a working window. In the context of a broker system, this period is akin to a training period, where calculations based on the data influence subsequent time steps.

Typical contracts for reserved instances last between one to three years, requiring considerably lengthy training data. However, to mitigate the extensive data requirements, we can truncate the data collection period by making certain assumptions. Customer behaviours often exhibit monthly and weekly cyclical patterns, which some might term as seasonal periods. By leveraging

Algorithm 1: Automatic model generator cloud brokerage

Data: Individual customer cloud request data
Result: Broker cloud inventory composition
x:= customers requests data;
y:= inventory composition;
acp: = Accepted level of the model from *HSIC* score
TS_x := time series(**x**);
TS_y := time series(**y**);
E := Fe_E(TS_x) $\cup$ Fe_E(TS_y);
profit := P(**x**, **y**);
F ,$\{e_i\}$ = Auto-func(E, profit);
while ***x*** *exists* **do**
  $\mathbf{x}_{new}$
  TS_x := time series(**x** + $\mathbf{x}_{new}$);
  TS_y := time series(**y**);
  From Fe_E: $k_i([TS_x, TS_y])$:= $\{e_i\}$
  FIND $\mathbf{y}_{new}$ from F($[e_i]$) := profit
  if $F([e_i])$ *accepted* $>$ ***acp*** **then**
    KEEP F, $\{e_i\}$;
  else
    TS_x := time series(**x**);
    TS_y := time series(**y**);
    E := Fe_E(TS_x) $\cup$ Fe_E(TS_y);
    profit := P(**x**, **y**);
    F ,$\{e_i\}$ = Auto-func(E, profit);
  end
end

these cycles, we reduce calculation time and circumvent constraints on data availability. Given that the only publicly reliable data sources are the cloud server traces from Google and Alibaba ([91; 99]), and considering that Alibaba's data trace is more recent, it forms the basis of our data selection. For this experiment, the auto-model employs 28 days' worth of backed data.

The time step represents the operational cycle of our broker system, spanning from data analysis to decision-making regarding inventory. The duration of a time step is crucial yet subject to user preference. Given that cloud instances are typically billed hourly and responses to customers must be timely, it is recommended that time steps be shorter than one hour. In this system, time steps are set at five-minute intervals.

We denote the current time step as t, the subsequent time step as $t+1$, and the preceding one as $t-1$.

6.5.2 Data

To mimic the real cloud workloads, we are using a server trace from Alibaba cloud (91). The trace consisted of multiple attributes. In this experiment, we are using the start and stop time of the work and the number of CPU cores assigned to the work as a performance indicator. Unfortunately, there is no available data that cover a multi-year duration. Hence, we over-sample the original data to increase the number of data points with the same distribution.

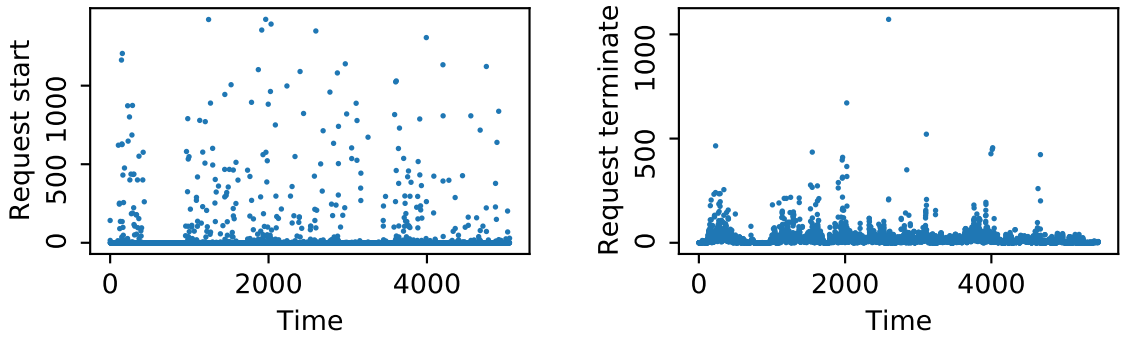


Figure 6.12: Example of the data distribution of the users requests. (Left) The starting requests per time step (Right) the termination requests per time step.

The simulator takes individual request orders chronologically and processes them at the same simulated time, i.e. if the requests start at time t, then it is processed at t.

A raw data format of a simple broker are formatted as follow:

```
USR = {ID,Start,Spec,TIME} | {ID,Terminate,TIME}

OND = {ID,Start,Spec,TIME} | {ID,Terminate,TIME}

RESV = {ID,Start,Spec,Terminate,TIME}
```

where `USR`, `OND`, `RESV` are a user request, on-demand instance in a broker inventory and reserved instance in an inventory respectively.

User requests and on-demand instances have an active and inactive state. The tuple after | is the inactive part which will be activated after the `TIME` is reached.

6.5.3 Results

The verification of the auto-model system is to verify that the generated model is a causal model. The identification of the causal model is done using the “intervene” method on the observed data.

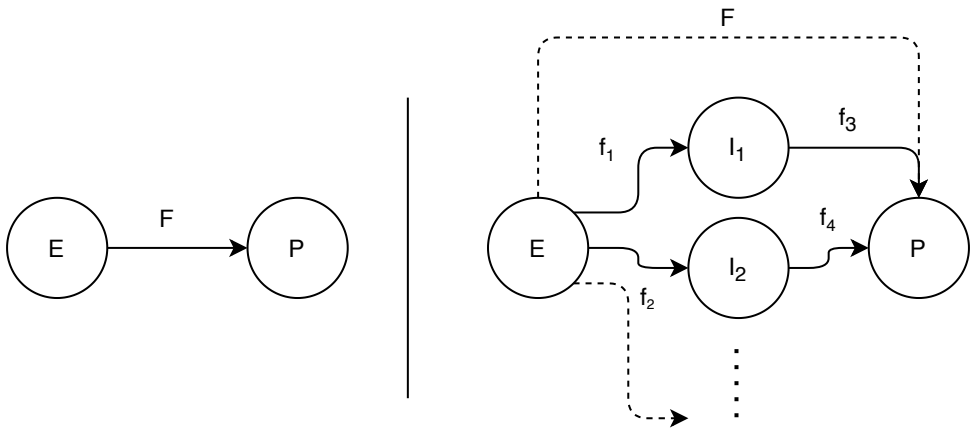


Figure 6.13: (LEFT) The figure shows typical causal model on the left-hand side. The causal relation function (F) is the behaviour of the effect parameter when the independent parameter change. (RIGHT) This figure shows causal model with hidden parameters (Is) i.e. inventory composition. f_3 and f_4 are fixed functions which contribute to the profit values while f_1 and f_2 are the regressed models from the independent parameter to the inventory composition.

The intervention is the middle step in the total of four steps of causality, association, intervention and counterfactual (93; 94). The association is about observing the change in parameters. The intervention is to do something and observe the change in parameters. Lastly, the counterfactual consider other causal relationship that might affect the observed parameters. All three of the steps must be fulfilled to identify the true causal relationship. However, for most data, including this work, the only intervention process is sufficient for the verification of the model (49).

Two sets of data points generated from two different models (functions) might have the same distribution. Hence, it is impossible to distinguish the generated data alone to infer the relationship. The behaviour of the function can behave differently.

Assuming that the causal diagram represents how the system behaves, we can verify that there is no relation between parameters by applying counter assumption. With intervention on the independent parameters, we can observe the change in dependent parameters and compare it to the original causal diagram.

From Figure 6.13, the original causal model is to find the parameter which affects the profit. Thus, intervention on the said parameter should change the profit distribution. Figure 6.14 shows the change in profit distribution after the intervention. Thus, we can see that the additive noise model effectively generates the causal structure from selected parameters.

Similarly, the right-hand side of Figure 6.13 shows the hidden parameters of the causal model, i.e. inventory composition. By the causal diagram direction, intervention on the independence

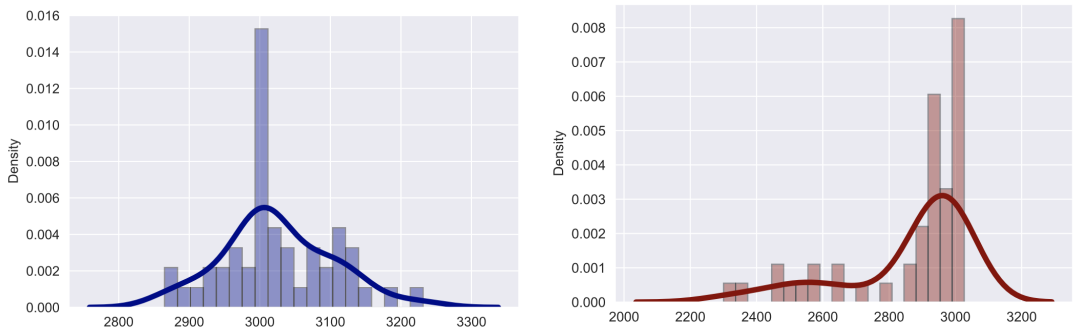


Figure 6.14: The figure shows the changes in profit data distribution after intervention on the independent parameter.

Data section	Mean	Variant
1	16.868	375.195
2	18.412	243.995
3	6.184	308.770
4	13.462	400.252
5	**23.008**	**1459.765**
6	7.444	254.965
7	13.72	760.497
⋮	⋮	⋮

Table 6.3: Mean and variant of each section of the data

parameter changes the distribution of the hidden parameters and then the profit distribution.

Volatility is widely accepted to be the main cause of profit management difficulty (62). We can safely assume that by eliminating the volatility in the data, a broker system would have an easy time managing its inventory. However, the elimination of the uncertainty is not easy or, in some cases, impossible. Thus, we wanted to see if the additive noise model can pick up on the volatile parameter in this case variant. We picked the data section in **bold** from Table 6.3, where the variant is considerably higher than the neighbouring data sections. The DAG shows that the standard deviation and mean of the data also influence the profit of the broker together with the size of the inventory, as shown in Figure 6.15.

To investigate the effectiveness of the auto-model building function, we force the decision function of the broker to be fixed functions. We use only one performance tier, which means the broker has to decide between two choices, an on-demand or a reserved instance, when the broker adjusts the inventory composition. If we fixed the decision model and auto-model from the additive noise model works properly. Then, we should be able to build a causal structure from the data generated by the fixed decision-making function.

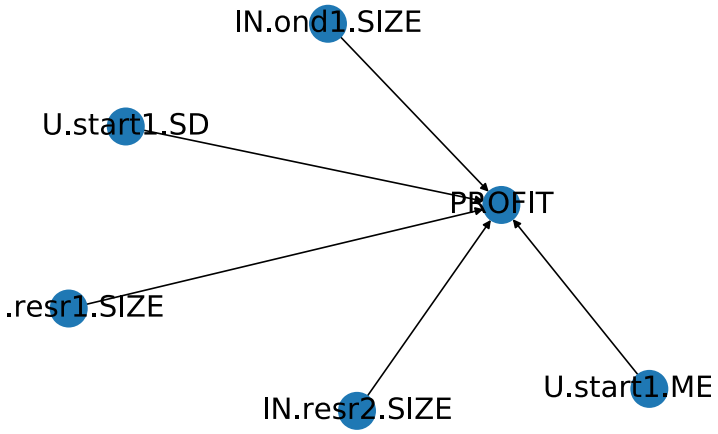


Figure 6.15: The DAG of the broker system when the data is volatile. The hypothesis testing from automatic model generation select a model with `SD` and `MEAN` parameter of the starting data that associate with the profit.

For the first model, we fix the decision function such that at the start there is a 50% chance of choosing either one of the choices. Then, if the profit is uptrend i.e. linear trend is positive the chance of reserved instance is increased by 1% wise versa.

$$f_{on}(x,a) = (1 - f_{trend}((\mathbf{z},a))x$$

$$f_{resv}(x,a) = (f_{trend}((\mathbf{z},a))x$$

$$f_{trend}(\mathbf{z},a) = a + (linear.trend.sign(\mathbf{z}))$$

where a is a size ratio of reserved instances and f_{trend} is a linear trend function of a profit. Notice that, the broker system is built in such a way that it does not use any parameter on our list that relates to the profit. The model and feature sets that associate with the profit change often with weak relation. Thus, it is not reliable or practical to follow any of the models.

Next, we added the mean of the data and its relation with the profit.

$$f_{on}(x,a) = (1 - f_{mean}(x,\mathbf{z},a))x$$

$$f_{resv}(x,a) = (f_{mean}(x,\mathbf{z},a))x$$

$$f_{mean}(x,\mathbf{z},a) = a + -\mathbf{z} + mean(x)\mathbf{z}^{0.5}/5$$

The accepted model from the ANM does include the $mean(x)$ of the data as one of the accepted features as expected.

Figure 6.16: The figure shows the movements of profit over time.

On the other hand, if the feature entity $mean(x)$ is added into a non-profit related function.

$$f_{on}(x,a) = (1 - f_{trend}((\mathbf{z},a))x + mean(x)$$

$$f_{resv}(x,a) = (f_{trend}((\mathbf{z},a))x - mean(x)$$

$$f_{trend}(\mathbf{z},a) = a + (linear.trend.sign(\mathbf{z}))$$

The auto-model cannot recognise the $mean(x)$ as a related pair to the profit. The auto-model can recognise the relation pair between a chosen parameter and the target from the example above.

Additionally, similar to many machine learning algorithms, the additive noise model effectiveness can be varied due to the parameter setting. For example, the DAG of the system would have a smaller number of nodes if the accepted threshold is too high. Unfortunately, there is still no set rule on parameter fine-tuning. For this work, we set the threshold to be high on purpose, as the main reason is to find a good indicator of the profit rather than a complete model.

A profit function with correspondent feature inferred from the Gaussian process serves as a profit model. Notice, profit function in Figure 6.16 is a time series function. On the other hands, profit model in Figure 6.17 (TOP) is a function with feature parameter (X) dependent. From the function, we can infer that the profit behaves in relation to the feature X. Figure 6.17 (BOTTOM) shows function realisations from the distribution function. These functions are used as a point prediction. The point prediction is the single solution that is needed in the decision making

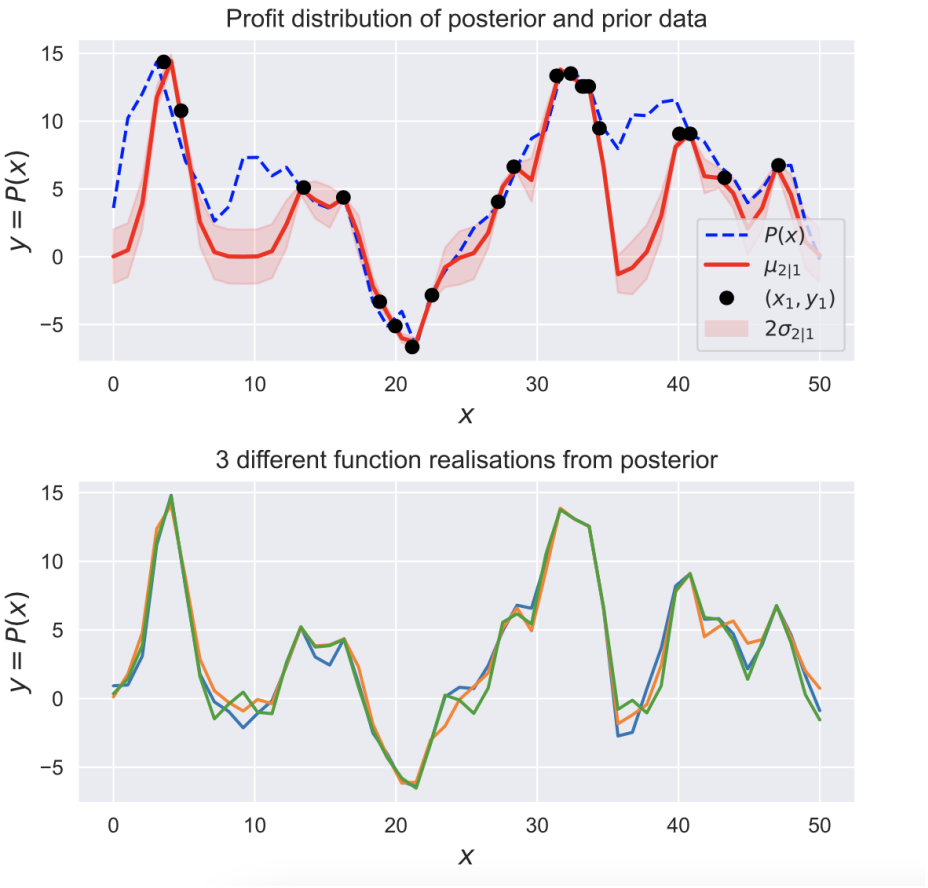


Figure 6.17: (TOP) The graph shows profit function infer from the Gaussian process with a correspondence parameters (feature extracted from the data). (BOTTOM) A function realisation for a point prediction.

process.

The number of sampling points contributes to the accuracy of the model to the real data. Since the real data is observed rather than a continuous function thus if the computing power allows increasing the sample number should result in higher accuracy. However, that is not always the case in Figure 6.18. Over-fitting is another issue that still needs further study. It does not only use more computing power, but the generated function behaves badly enough to cause an inaccurate area. Fortunately, over-fitting and parameter selection problems are the main strength of the causal discovery (additive noise model) handled by the HSIC. The over-fitting would fail the bi-directional independence test while parameter selection is handled using both independent and Gaussian processes.

Finally, to compare the system with other prediction methods, we use the Auto-ARIMA, which represents the regression type method (96). The Auto-ARIMA automatically choose the best fit ARIMA model for the given data. Thus, the Auto-ARIMA gives the best representation of

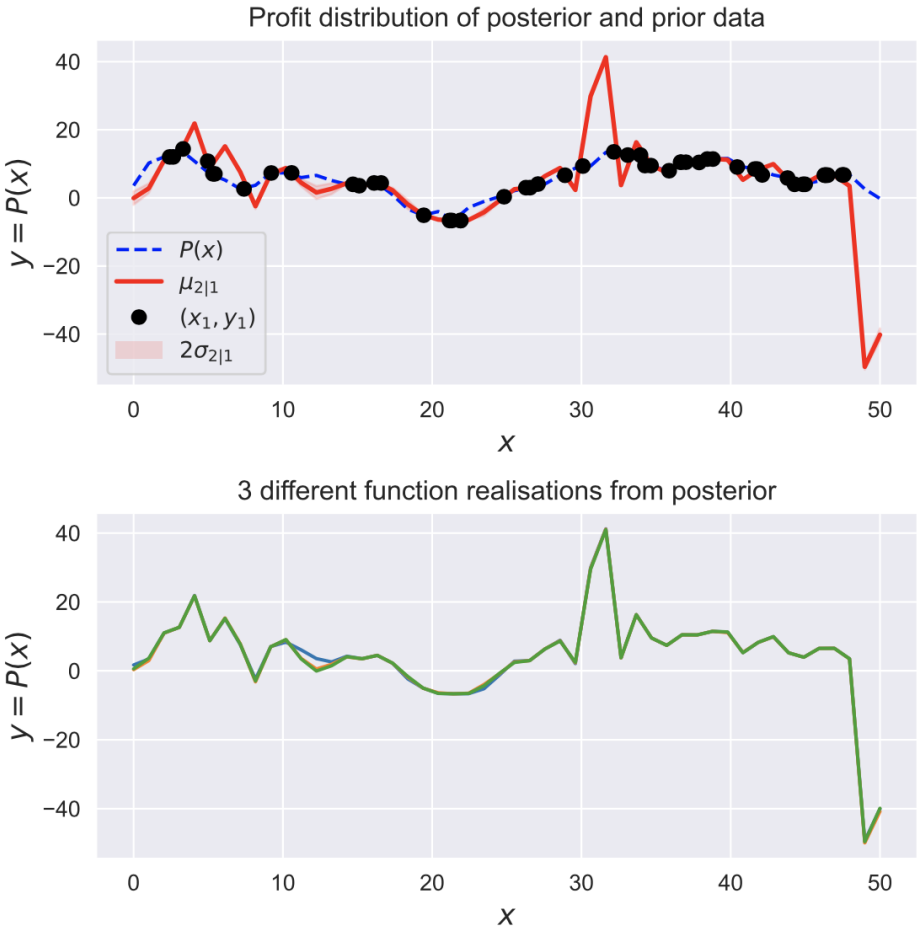


Figure 6.18: (TOP) The graph shows profit function infer from the Gaussian process with a correspondence parameters (feature extracted from the data). (BOTTOM) A function realisation for a point prediction. In this figure, we can see the overfitting problem and high error function approximation.

a general ARIMA model. Another system in the comparison is the artificial neural network, specifically, the short-term-long-term memory neural network. This network is suitable for the prediction of time series data (82; 11). Thus, it is also a good benchmark for the neural network prediction algorithm. We use these systems to verify that the model works as intended, i.e. generate profit in the same direction.

1. The regression system: We use a time series prediction (Auto-ARIMA) to determine the inventory composition. The Auto-ARIMA eliminate the parameter tuning, which makes it a fair comparison. The Auto-ARIMA predicts the demand from the historical data of the customers' data. If the real value data exceed the predicted values, then the overflow orders are placed onto on-demand instances instead of reserved instances.

2. Short-term-long-term memory neural network: the network was trained using the input from

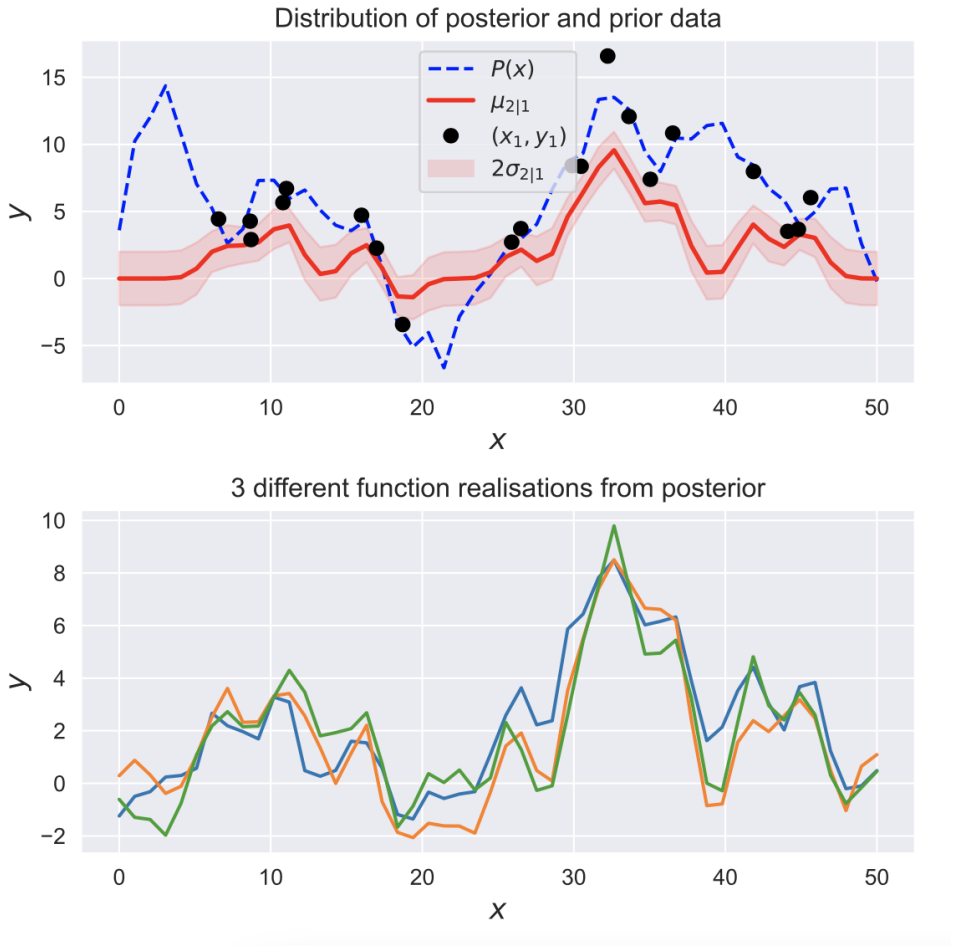


Figure 6.19: (TOP) The graph shows profit function infer from the Gaussian process with correspondence parameters (feature extracted from the data). (BOTTOM) A function realisation for a point prediction.

customers time series queries and inventory data with the optimal profit value.

3. The auto-generated model system: The proposed system utilises the feature extraction and the additive noise model like method. The feature set is used as an input to the model generator as candidates that could influence the profit. The hypothesis testing and additive noise filter out the unrelated features and build a relation model from the accepted ones.

Comparing the effectiveness of the three broker systems, we show the utilisation of the inventory and profit. The utilisation is not the main target of the system. However, it does reflex the effectiveness of the decision making somewhat. The number of reserved instances vs utilisation is shown in Figure 6.20. Overall, the total number of reserved instances in the auto-model is slightly lower than the Auto-ARIMA. However, utilisation is considerably higher. The ANN, on the other hand, employ more reserved instances. Difference strategies create different inventory compositions. However, generally, the behaviour of the profit is similar with minor differences

in values.

Figure 6.21 shows the raw profit comparison of the Auto-ARIMA, the short-term-long-term memory artificial neural network, and the model generator. The blue line (model generator) is in multiple periods higher than the other two solutions. In the big picture, the profit of the three systems behaving similarly. This indicates that all of them are effective at managing the broker inventory.

6.6 Related Works & Future Directions

The concept of creating a detailed description of a system using mathematical equations has been one of the most important topics in science and engineering. However, the discussion of methodologies behind the building blocks of the system is still rarely mentioned or mostly avoid due to the nature of conjuring something out of thin air (data) without doing a confirmation experiment is not popular among scientists. Nevertheless, the concept of building relationships plays a vital role in understanding the development of the system.

Building a system of equations that try to understand the mechanics from existing data can be useful in many fields of science (34). The additive noise model is one such algorithm from a family of causal discovery methods which can identify and infer the underlying system interaction. There are systems such as bio-diversity and the relation between predator and prey which utilised the hypothesis testing to understand the interaction between predators and preys (10). The group uses the inferring method of causality to discover inter-species interaction with good performance comparing with the well-studied model. Additionally, in a related field of cloud computing, performance anomaly detection can be improved using feature selection based on causality mining. The causal discovery method attempts to identify the root cause of the anomaly, which can be taken care of by the system (97). Another usage of the causality is to utilising it as a feature selection method (127). Several feature selection algorithms are utilising the causality that shows a better performance compared to the traditional methods. Lastly, the additive noise model used in this chapter also provides a strong performance of causal structure learning from time series, which is the data structure used in our system (118).

As for a cloud broker system, an approach to optimise the system using the additive noise model is a novel approach. The usage of causality application stops at the inference and the identification of causal data pairs. In this work, we have expanded the usage of the additive noise model further to build a profit model. The model helps with the adjustment of the inventory that benefits the profit of our broker system.

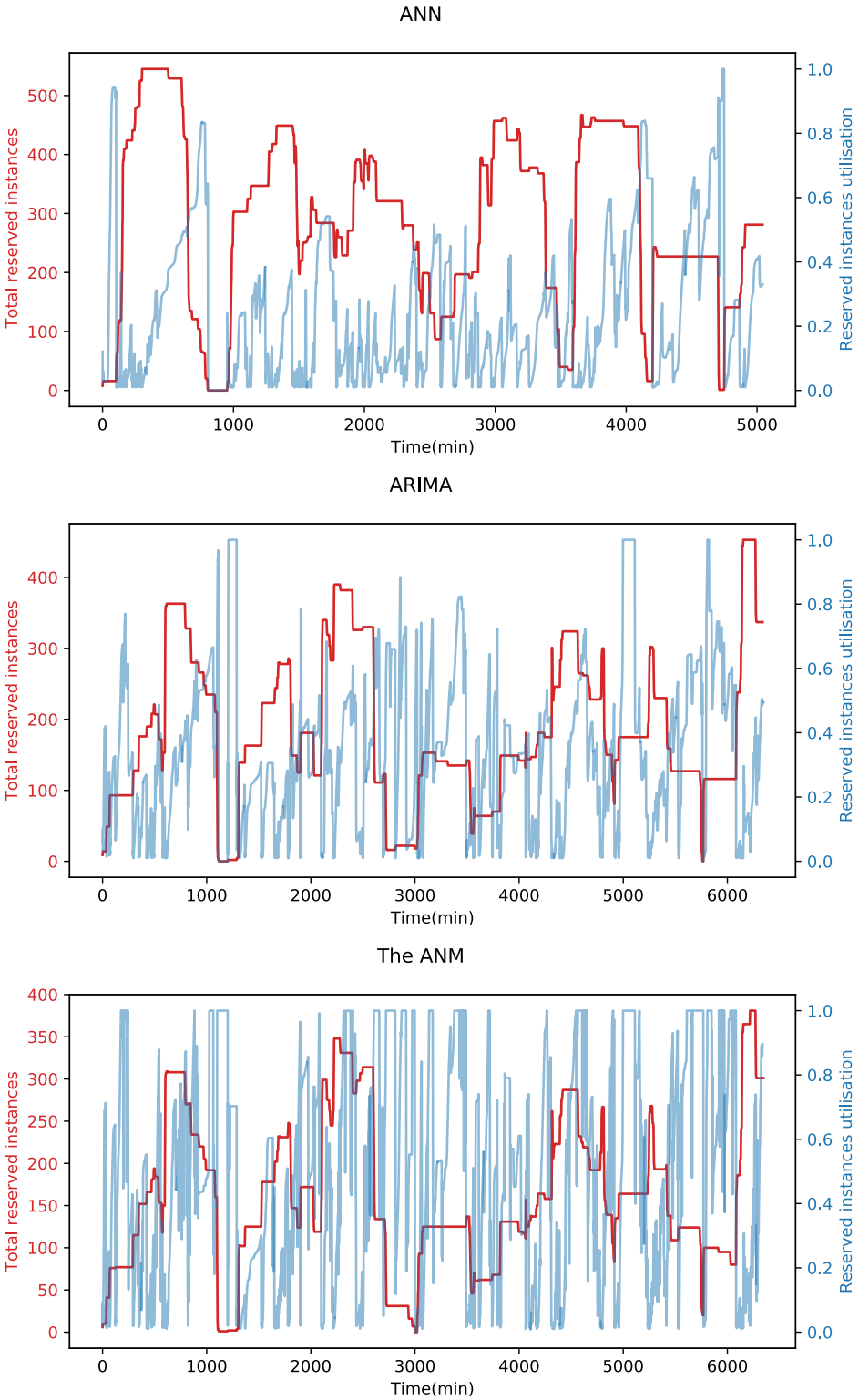


Figure 6.20: (TOP) The short-term-long-term memory artificial neural network time series prediction brokerage system result is shown. The solid red line is the number of reserved instance while the solid blue line is the average reserved instance usage. (MID) The figure shows the number of reserved instances in the broker inventory and average utilisation of the reserved instances with the Auto-ARIMA. (BOTTOM) The result from the auto-model system shows overall lower number of reserved instances but higher utilisation.

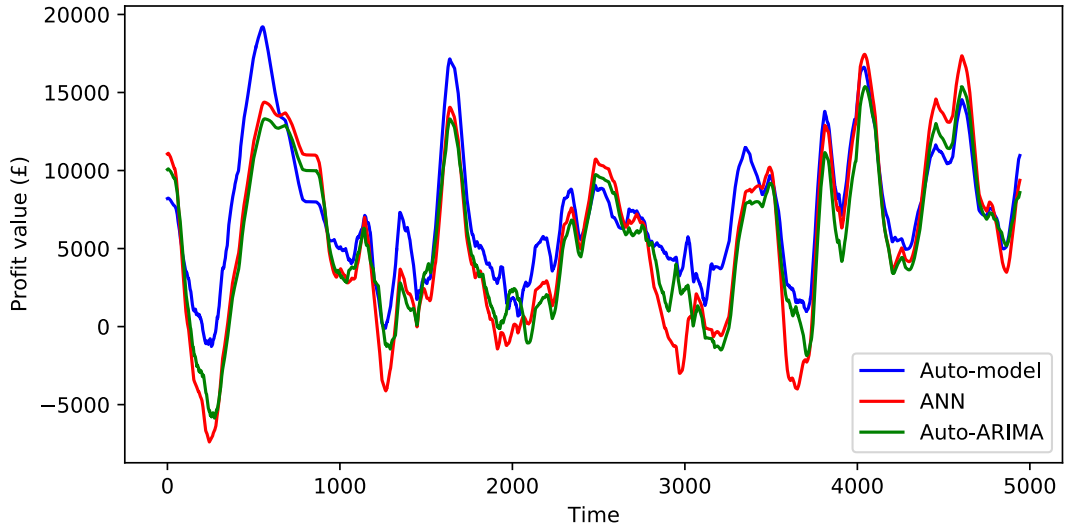


Figure 6.21: The graph shows profit comparisons between the auto-model, Auto-ARIMA and the artificial neural network system. Generally, the profit trend is going in the same direction as expected from all the working algorithms. However, there are some areas in which the auto-model generate a higher level of profit. While there are some areas that the other two systems that perform better. However, overall, the auto-model can generate a higher profit level than the other two competing systems.

However, the automatic model generator brokerage, while showing a promising result, can still be improved. The algorithm suffers similar flaws as traditional hypothesis testing, i.e. curse of dimensional. Additionally, the feature extractions can be improved to represent the insight characteristic of the data better, i.e. the cross over feature generator (61). A complex set of features could provide the system with a better understanding of its behaviour.

6.7 Summary

Matching the size and configuration of the broker inventory is the most important aspect of having a high-profit broker. However, achieving the said goal is not trivial. Since there could be multiple factors that influence profit, in this chapter, we have proposed a resource management system for a cloud broker based on the feature extraction process and the model generator from the additive noise model. The critical component in the ANM identifies variables from the feature extracted time series data that affect the broker's profit in real-time. By establishing these variables, the broker can adjust its inventory of multiple cloud instance types to suits the incoming data. From the simulation, our cloud broker system was able to infer the underlying interaction model between variables and achieves higher profit than the standard time series regression (Auto-ARIMA) and short-term-long-term memory artificial neural network broker.

CHAPTER SEVEN

FUTURE WORKS & CONCLUSION

The development of the cloud broker is relatively new as the cloud market itself. Thus, there are still many problems that have yet to be solved. The existence of the cloud brokerage system alone solved many of the inherited problems of the cloud, i.e. performance, cost, simplicity. In this work, we focus mainly on the broker inventory system, which deals directly with the cost of the cloud. The decision-making system for the broker inventory influences the profit of the broker directly. Getting the correct configuration of the inventory comes with many challenges. Two approaches are proposed and evaluated using both synthetic data and cloud trace data. However, several aspects can be improved, which will be discussed in this chapter.

7.1 Contributions

We discuss our major contributions in this work by considering the four components of our hypothesis (section 1.5) and how we addressed the related challenges during the research and development.

7.1.1 Risks in Cloud Brokerage Operation

The operation of a cloud brokerage service that employs reserved instances poses some risks.

> *HYP 1 - A bulk buying cloud broker operates under some risks from the uncertainty of the user requirements. These risks can be managed by identification of the risk factors and functional risk management.*

We have addressed the bulk buying hypothesis with risk factors and accompany decision functions. From the evaluation in chapter 5, the decision-making from the risk factors reduces the effect of uncertainty and, therefore, can produce a good profit level. Additionally, the risk level is allowed to change depending on the cash-on-hand of the broker. This is to reduce the opportunity cost that might happen.

Furthermore, the broker system utilised two sources of data. The benefit of using two sources of data is that we get feedback from both sources. By getting feedback from both sources, the broker does not have to rely solely on the prediction of the customers' data and includes the state of the broker inventory.

Additionally, the inclusion of the broker data helps with the inventory adjustment, which will be discussed in the solutions in this thesis.

7.1.2 Hidden Parameters in Random Process

We have mentioned a random process in previous chapters. Randomness is one of the main issues that we are trying to solve. From the hypothesis,

> *HYP 2 - A random process in a structural system contains hidden causal parameters and relationships that can be inferred.*

Within an organized system, we hypothesize that it is possible to infer relationships among some parameters from seemingly random data. To test this hypothesis, we establish a fixed decision function system within a broker framework. The data input into the system is generated using a random walk model, which provides a robust simulation of pseudo-random movements, thereby approximating real-life data dynamics.

By configuring the fixed-function system to utilize specific parameter values extracted from the random input data as thresholds for decision-making, we investigate the potential to infer the relationships of these parameters to the outcomes. The inference process employs the additive noise model to ascertain whether the derived parameter values influence the decision outcomes in a predictable manner, thus validating or refuting the hypothesis.

7.1.3 Behaviour of Parameters in The Brokerage System

After the work on the risk factors, we have developed a parameter identification system based on the additive noise model. The main idea is to find confirmation on some of the risk assumptions, i.e. higher remaining volume in the inventory poses more risk. In this system, the assumption is

inferred by the relations from the additive noise model causal discovery. Therefore, this allows for the automatic model generator.

HYP 3 - In an optimal system, parameters behave under certain underlying relations.

In chapter 6, we have tested that under optimal setting, the additive noise model can select parameters that we generate using the feature extraction process from both customers and broker data. With the pair, if we make a change on one parameter, then the distribution of another parameter also change. This behaviour is verified through a process called intervention. The intervention is one of the verification methods of causality. We are not going too deep into the true causality. We focus on the effect of the parameter relations in our system, which will be used in the decision-making process of our brokerage system.

7.1.4 Decision System For Cloud Broker Inventory

The decision system for our brokerage system is built upon the extracted parameters from both sources of data. From the hypothesis,

HYP 4 - We can use the behaviour of the parameters to build a decision system for the cloud broker inventory.

We developed decision systems for cloud broker inventory management based on models of parameter behavior. These systems were devised from two distinct approaches: a risk-based model and an auto-model generator. The risk-based model relies on a human-curated list of risk factors, whereas the auto-model generator employs inferred relationships derived from the additive noise model. Both systems are predicated on the assumption that changes in parameters can significantly influence profit behavior.

In the risk-based approach, profit is influenced by adjustments to the inventory composition, guided by identified risk factors. This adjustment directly impacts the profit levels. Conversely, in the auto-model approach, the relationships between extracted parameters and profit are determined through the additive noise model. The cascading effect of this model enables the system to deduce relationships from selected parameters to inventory composition and from inventory numbers to profit. Consequently, this model informs decisions about which cloud instances should be added to the inventory, adhering to the parameter-to-profit relationships initially inferred.

Our evaluations, detailed in Chapters 5 and 6, confirm the effectiveness of both approaches. The findings indicate that we have successfully validated our hypothesis, demonstrating that parameter behavior can indeed be harnessed to construct a decision-making framework for cloud broker inventory management.

7.2 Cloud Brokerage System Research & Future Work

The specific future direction of each proposed solution has already been discussed in chapter 5 and 6. This section is dedicated to the direction and future work of cloud brokerage systems in general.

Up until here, the work has been focused on the broker inventory and the management of cloud instances. This area relates closely to the performance aspect of the cloud. After all, a performance indicator is one of the quantifying factors of a value of a computer. Thus, a good starting point of future work can be found in a cloud performance-related topic.

7.2.1 Performance Related Cloud Brokerage System

It is easy to see that the performance is the best way to quantify the value of cloud instances or any computers, i.e. faster and cheaper is most likely have a better value. However, quantifying the performance of a cloud instance is complex. As of recently, general computing power measurements are done through a general benchmark suite where it tests multiple aspects of computer calculation power. The final score is calculated from the weighted performance of each curated task. All of the tasks, weights, and scales are decided by the developer of the benchmark suite. Therefore, there is always going to be an argument on the method. Furthermore, specialised workloads and specialised cloud instances are seeing more traction as of late. Special hardware for machine learning acceleration, for example, cannot be measured using traditional methods. Thus, the benchmark suite has to keep up with the change and update the tests. The update makes the old benchmark score obsolete and cannot be referenced. Additionally, benchmark score is not the sole indicator of how well the application would run on a computer. To solve the issue, cloud users could perform a cloud workload characteristic mapping (126; 107).

From Figure 7.1, many of the broker components are subject of an active ongoing research. Starting with the cloud application itself, the characteristic of an application, if not specified by the customer, is challenging to model after without a test run. At this stage, a broker can employ a testing instance that test runs the application while capturing the characteristic in real-time since monitoring and data collection is a routine procedure of system administrators to keep an

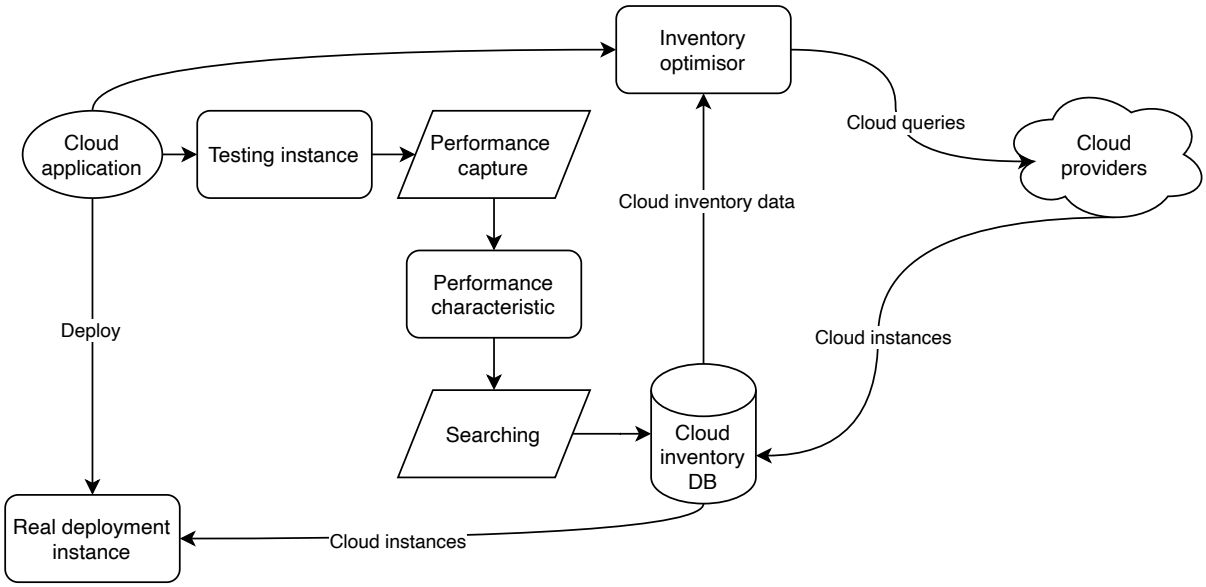


Figure 7.1: A map of cloud brokerage components

operation running smoothly. If the monitoring data and cloud characteristics can be matched up, choosing the right cloud should be easier.

Another main component that we can use to improve the work in this thesis is the cloud inventory database. The cloud inventory database collects and records the performance and crucially characteristics of cloud instances. The database helps select the right cloud for the job and interchange one cloud instance for another if necessary. For example, if cloud instance 1 is occupied, but there is a performance equivalent slot on a bigger cloud instance available, then it can be used as a substitution without altering the cloud experience on the customer's end. The swappable property gives flexibility to the decision-making process of the system. For example, if the broker knows that cloud instance `A1` from Amazon perform close to the instances `small` from Microsoft Azure but cheaper, then these two are interchangeable. On the other hand, with the possibility of instance sharing, i.e. one instance hosts more than one customer virtual machine image, bigger quantified instances are equivalent to multiple smaller instances. Again, this further gives the system more choice and opportunity to reduce cost and make more profit.

7.2.2 Matching Application Characteristics

After creating the cloud application testing instance and cloud inventory performance database, the broker needs to match the application with the right cloud instance for the best performance.

From Figure 7.2, cloud applications perform differently depending on the specification requirements. For example, some applications scale well with multi-core cloud instances,

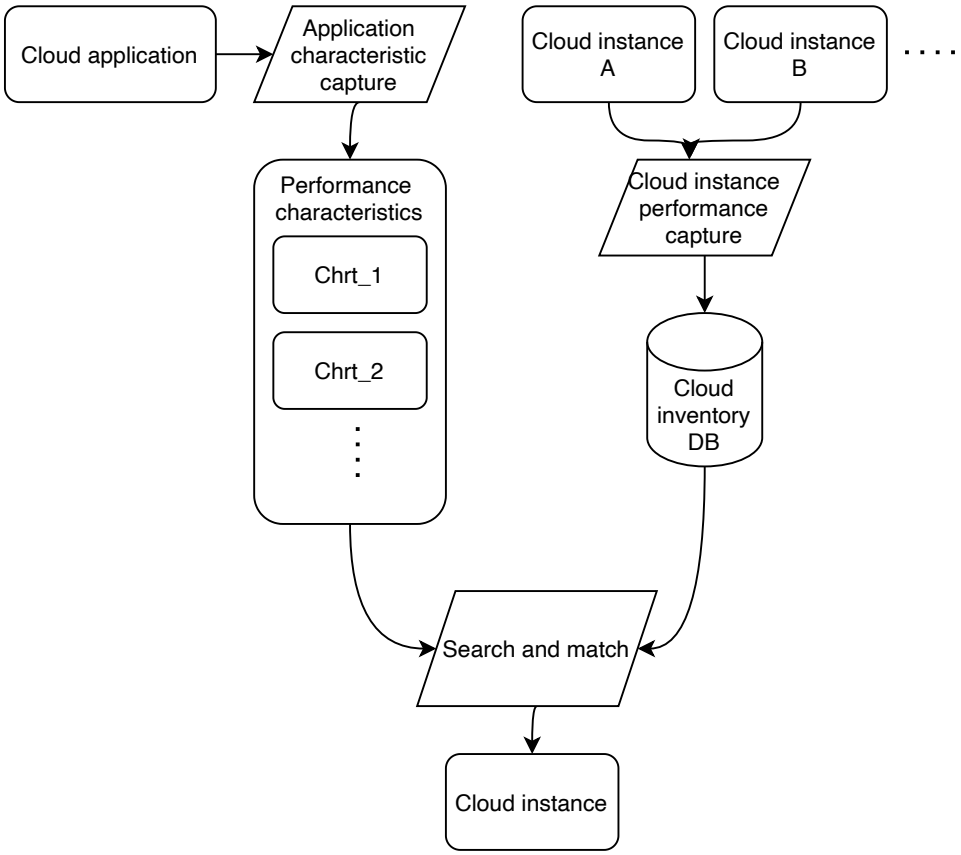


Figure 7.2: The diagram shows a performance matching workflow and components. Cloud applications and cloud instances go through a similar key feature capturing process. After that, the matching algorithm between cloud application characteristics and cloud instance performance from the database takes over. The result is the best cloud instance for the application.

but some perform well with high capacity memory. These characteristics need to be defined, and measured (75).

The method of measurements is one of the active research areas. This includes monitoring, stress testing etc. (37; 88). These characteristics data from the measurement should benefit choosing the right cloud instance.

Cloud instance performance characteristics are more complex than the straightforward performance specifications typically advertised for computers, such as CPU and memory capacities. Unlike physical hardware, cloud providers often list these specifications in the form of *virtual* specifications. This distinction means that direct comparisons between different providers or even different instance types are not feasible, as the virtual specifications do not correspond directly to physical hardware capabilities.

A standardized performance indicator becomes crucial in this context. Such an indicator would

greatly benefit cloud customers by providing a clear and consistent basis for comparing cloud instances across different providers. This standardization would simplify the decision-making process for customers shopping for cloud instances, enabling them to make more informed choices based on comparable performance metrics.

7.2.3 Optimisation of Broker Inventory

The principal focus of this research is the optimisation of broker inventory, particularly within the context of varying pricing schemes for cloud instances. This study proposes two distinct methodologies for addressing inventory optimisation. The first method, a risk-based system, employs predefined risk factors to facilitate decision-making. The second method involves an additive noise model broker that utilises a causal discovery process to generate and select decision models. Both approaches yield favourable outcomes; the risk-based system is independent of data, whereas the auto-model generator depends on data. The latter has the capability to adjust its model in response to new data. Theoretically, once the broker has processed sufficient data, it may attain data independence. Valid models can be archived within a model bank and subsequently retrieved for use under appropriate conditions.

The selection of previously applied models and their associated variables necessitates a model storage and selection method, as illustrated in Figure 7.3. This additional layer can be integrated into the system to enable the application of a previously stored model to a recurring pattern of input data. Furthermore, both optimisation and Gaussian processes are known for their computational demands. Enhancements in these areas would significantly expedite the selection process, particularly in scenarios requiring frequent decision-making.

It is pertinent to note that the model bank bears resemblance to a deep learning model, where an increased number of neurons facilitates the learning and storage of a greater variety of solutions. However, the scalability of training in a model bank is limited. The auto-model generator is trained using a predetermined number of data points to identify a single valid model, whereas a deep learning network iteratively adjusts until it determines the optimal model capable of classifying 'n' patterns.

As the terminology suggests, the model selector is an algorithm specifically devised to choose an appropriate model from the storage for use by the broker. The criteria for model selection can vary; however, the core principle involves aligning the characteristics of the application data with the intended outcome.

In summary, a critical component of cloud brokerage aimed at profit generation involves amalgamating the aforementioned steps with an effective decision-making or optimisation

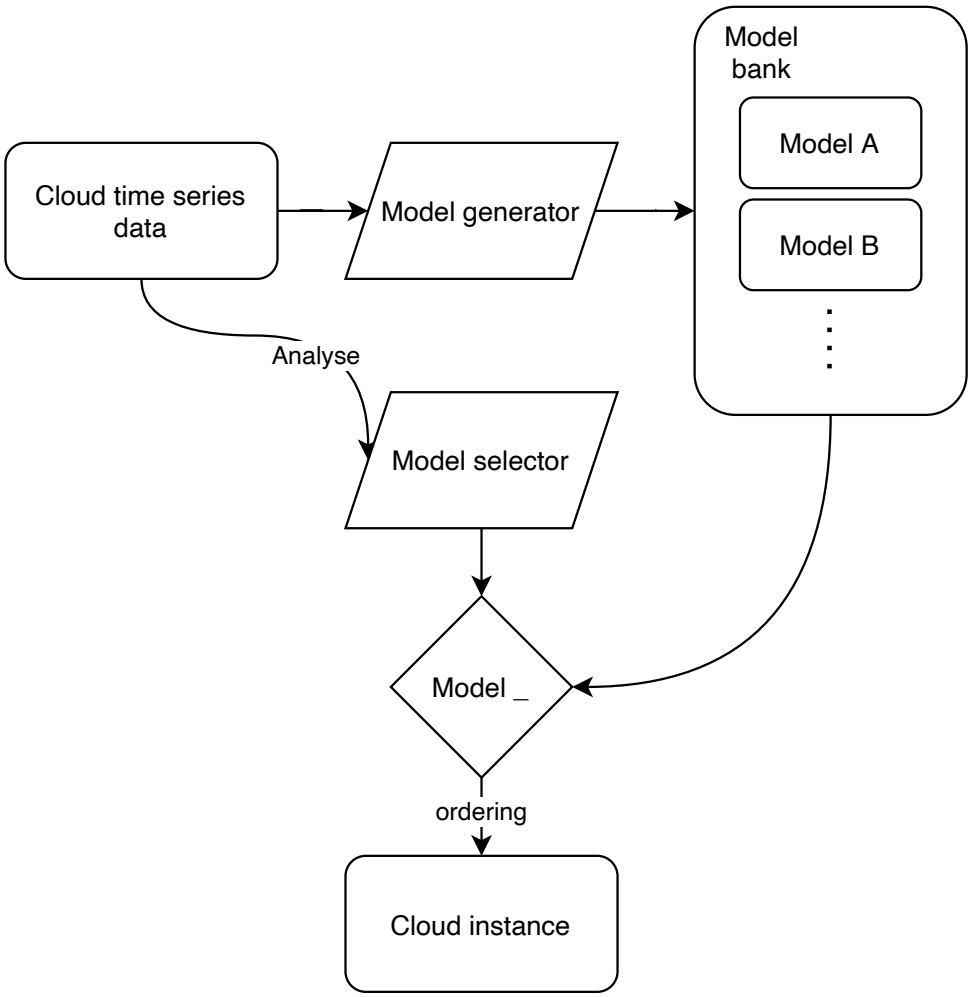


Figure 7.3: Causal discovery methods are used to infer relationships and causal parameters from a dataset. Should the population (or its distribution) undergo changes, the inferred data may become outdated. To address this, a causal model storage system is implemented. This system archives models so that they can be retrieved and reused when similar data patterns are encountered in the future.

process to forge a comprehensive broker system.

Causality and System Modelling

Causality is a longstanding concept extensively explored and employed in scientific research. A key utility of causality is in identifying the causes behind observed effects or outcomes within experiments. In contemporary applications, causality techniques are deployed to decipher complex systems. For instance, business-related data often leverages causal discovery methods to enhance system efficiency, as highlighted in existing literature ([20]). In the context of a well-structured system like a cloud broker, the application of causality can be extended to various processes, such as the queuing of queries or in areas adversely impacting user experience, such

as the time taken to initialize a virtual machine. By accurately identifying the underlying causes, cloud brokers or providers can significantly improve user experiences by mitigating delays.

7.2.4 Additional Research Domains in Cloud Brokerage Systems

There are several burgeoning areas within the realm of cloud brokerage systems that merit further exploration:

Security is paramount, particularly in cloud computing where customers often store sensitive information remotely. Physical security is a notable advantage of cloud computing, as data centers maintain rigorous protection standards. Conversely, software security remains a significant concern for cloud clients (102). Given that cloud computing environments typically accommodate multiple tenants in shared physical spaces, ensuring secure isolation of tenant domains is imperative.

Service Level Agreements (SLAs) are critical clauses that cloud providers are obligated to honor with their customers. These agreements are designed to ensure satisfactory service across all aspects of cloud operations. Brokers, assuming the role of cloud providers, are also required to uphold these SLAs. This responsibility is challenging since brokers often do not control the underlying cloud infrastructure. Although SLAs do not directly correlate with a broker's profitability, there is potential to optimize profits based on service quality within an e-commerce framework (78). Strategies such as employing queuing theory and fixed-point optimization can be advantageous for the cost model.

Furthermore, brokers can leverage SLA negotiation tools to transfer any discounts or compensation for disruptions from the providers to the customers (124). Numerous approaches exist that address SLAs, potentially enhancing the service quality of the brokerage system.

User Interface and API: The interface is the initial point of interaction between cloud customers and brokerage services. Currently, only commercial brokers offer user-friendly, web-based interfaces. The focus has predominantly been on the backend functionalities of brokers. However, enhancing user accessibility remains a critical area for development. Features such as a high-level display of performance metrics and pricing are essential to improving user experience and should be integrated into brokerage services.

7.3 Conclusion

Research into cloud brokerage systems is both multifaceted and intricate. This work has extensively explored the potential of employing cloud brokers as comprehensive service providers

for cloud customers. Through bulk purchasing, brokers offer customers the flexibility of on-demand cloud services at reduced costs. Moreover, by managing multiple cloud instances, brokers can ensure a certain level of performance, enhancing user experience as discussed in Chapter 4 and supported by (105). This model not only helps familiarize customers with cloud services but also contributes to the growth of the user base, ultimately benefiting the entire industry.

Our approach utilizes an inventory-based broker model, primarily focusing on reserved instances as the primary product. These instances, secured through long-term contracts of one to three years, offer a more cost-effective solution compared to on-demand options due to their lower price per unit of time. The broker system is tasked with accurately forecasting demand to optimize the number of reserved instances purchased for profitability. Additionally, to minimize the idle time of reserved instances, our broker strategically incorporates on-demand instances into the inventory.

The decision-making process for selecting specific cloud instances employs various techniques aimed at optimizing profitability. However, predictions are crucial as they inform the optimization process. We observed that mere predictions are insufficient for profitability due to potential overestimations and underestimations, particularly when predictions deviate significantly from the training data.

To address these challenges, we proposed two innovative solutions. The first, a risk-based solution, employs predefined risk factors and corresponding decision functions to manage the cloud inventory. This model simulates one-to-one deployment—matching one customer query to one inventory instance. It strategically adjusts risk exposure based on the broker's financial position, showing promising profitability in simulations using the Alibaba dataset. Despite its effectiveness, this system has limitations, including the manual selection of risk factors and constraints in broker model configuration.

The second solution, an automatic profit model generator, is inspired by causal inference. It identifies relevant parameters that influence profitability and generates a corresponding model. This system offers greater flexibility than the risk-based solution but still relies on the availability of comprehensive customer data. For optimal performance, it requires extensive data to learn various potential models, which may not always be readily available.

In conclusion, this study presents novel methods for making informed inventory decisions aimed at profit optimization in cloud brokerage. By developing and testing two distinct models, we have demonstrated that it is possible to effectively manage the uncertainties inherent in cloud services, a challenge common to various sectors such as stock trading and airline ticket pricing.

Our mathematical models provide a deeper understanding of complex systems, and with further research, they could be adapted to address a broader spectrum of challenges across different industries.

REFERENCES

[1] *Announcing amazon elastic compute cloud amazon ec2*, 2006.

[2] A. Agrawal, C. Catalini, and A. Goldfarb, *Some simple economics of crowdfunding*, Innovation policy and the economy, 14 (2014), pp. 63–97.

[3] F. Allen and R. Karjalainen, *Using genetic algorithms to find technical trading rules*, Journal of financial Economics, 51 (1999), pp. 245–271.

[4] S. Ambike, D. Bhansali, J. Kshirsagar, and J. Bansiwal, *An optimistic differentiated job scheduling system for cloud computing*, in International journal of engineering research and application, Mar. 2012.

[5] A. Anwar, Y. Cheng, and A. R. Butt, *Towards managing variability in the cloud*, in 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2016, pp. 1081–1084.

[6] N. Arinaminpathy, S. Kapadia, and R. M. May, *Size and complexity in model financial systems*, Proceedings of the National Academy of Sciences, 109 (2012), pp. 18338–18343.

[7] V. Arutyunov, *Cloud computing: Its history of development, modern state, and future considerations*, Scientific and Technical Information Processing, 39 (2012), pp. 173–178.

[8] K. Assaad, E. Devijver, E. Gaussier, and A. Ait-Bachir, *Scaling causal inference in additive noise models*, in The 2019 ACM SIGKDD Workshop on Causal Discovery, PMLR, 2019, pp. 22–33.

[9] A. Barker, B. Varghese, J. S. Ward, and I. Sommerville, *Academic cloud computing research: Five pitfalls and five opportunities*, in 6th {USENIX} HotCloud 14, 2014.

[10] F. Barraquand, C. Picoche, M. Detto, and F. Hartig, *Inferring species interactions using granger causality and convergent cross mapping*, Theoretical Ecology, (2020), pp. 1–19.

[11] F. Bellas and R. J. Duro, *Introducing long term memory in an ann based multilevel darwinist brain*, in International Work-Conference on Artificial Neural Networks, Springer, 2003, pp. 590–597.

[12] D. Bernstein, *Containers and cloud: From LXC to docker to kubernetes*, IEEE Cloud Computing, 1 (2014), pp. 81–84.

[13] J. Bley and M. Saad, *An analysis of technical trading rules: The case of mena markets*, Finance Research Letters, 33 (2020), p. 101182.

[14] M. Bunge, *Causality and modern science*, Routledge, 2017.

[15] R. Cai, J. Qiao, K. Zhang, Z. Zhang, and Z. Hao, *Causal discovery with cascade nonlinear additive noise models*, arXiv preprint arXiv:1905.09442, (2019).

[16] M. Campbell-Kelly and D. D. Garcia-Swartz, *Economic perspectives on the history of the computer time-sharing industry, 1965-1985*, IEEE Annals of the History of Computing, 30 (2008), pp. 16–36.

[17] J. Cao, K. Hwang, K. Li, and A. Y. Zomaya, *Optimal multiserver configuration for profit maximization in cloud computing*, ieee transactions on parallel and distributed systems, 24 (2012), pp. 1087–1096.

[18] M. Carroll, P. Kotzé, and A. Van Der Merwe, *Securing virtual and cloud environments*, in International Conference on Cloud Computing and Services Science, Springer, 2011, pp. 73–90.

[19] S. Chaisiri, B. sung Lee, and D. Niyato, *Optimization of resource provisioning cost in cloud computing*, IEEE TSC, 5 (2012), pp. 164–177.

[20] S. Chalyi, I. Levykin, A. Petrychenko, and I. Bogatov, *Causality-based model checking in business process management tasks*, in 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT), IEEE, 2018, pp. 453–458.

[21] G. Chandrashekar and F. Sahin, *A survey on feature selection methods*, Computers & Electrical Engineering, 40 (2014), pp. 16–28.

[22] Y. Chen and X. Wang, *A hybrid stock trading system using genetic network programming and mean conditional value-at-risk*, European Journal of Operational Research, 240 (2015), pp. 861–871.

[23] I.-H. Cheng and W. Xiong, *Financialization of commodity markets*, Annu. Rev. Financ. Econ., 6 (2014), pp. 419–441.

[24] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, *Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)*, Neurocomputing, 307 (2018), pp. 72–77.

[25] C. Y. Chung, S. Kang, and D. Ryu, *Does institutional monitoring matter? Evidence from insider trading by information risk level*, Investment Analysts Journal, 47 (2018), pp. 48–64.

[26] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, *Benchmarking cloud serving systems with YCSB*, in Proceedings of the 1st ACM symposium on Cloud computing, 2010, pp. 143–154.

[27] D. J. Daly and D. J. Daly, *Economics 2: Ec2*.

[28] A. K. Das, T. A. M. A. Razzaque, E. J. Cho, and C. S. Hong, *A QoS and profit aware cloud confederation model for IaaS service providers*, in Proceedings of the 8th international conferences on ubiquitous information management and communication, Jan. 2014.

[29] Y. Deng, S. Shen, Z. Huang, A. Iosup, and R. Lau, *Dynamic resource management in cloud-based distributed virtual environments*, in Proceedings of the 22nd ACM international conference on Multimedia, 2014, pp. 1209–1212.

[30] C. L. Devasena et al., *Impact study of cloud computing on business development*, Operations Research and Applications: An International Journal (ORAJ), 1 (2014), pp. 1–7.

[31] P. R. Dickson and J. L. Ginter, *Market segmentation, product differentiation, and marketing strategy*, Journal of marketing, 51 (1987), pp. 1–10.

[32] J. Du, R. Liu, and R. R. Issa, *BIM cloud score: benchmarking BIM performance*, Journal of Construction Engineering and Management, 140 (2014), p. 04014054.

[33] A. Elhabbash, F. Samreen, J. Hadley, and Y. Elkhatib, *Cloud brokerage: A systematic survey*, ACM Computing Surveys (CSUR), 51 (2019), pp. 1–28.

[34] J. Elinger, *Information Theoretic Causality Measures For Parameter Estimation and System Identification*, PhD thesis, Georgia Institute of Technology, 2020.

[35] E. Erkko, *Broker-dealer risk appetite and commodity returns*, Journal of financial econometrics, 11 (2013), pp. 486–521.

[36] A. Esfahanipour and S. Mousavi, *A genetic programming model to generate risk-adjusted technical trading rules in stock markets*, Expert Systems with Applications, 38 (2011), pp. 8438–8445.

[37] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, *A survey of cloud monitoring tools: Taxonomy, capabilities and objectives*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 2918–2933.

[38] F. Faul, E. Erdfelder, A. Buchner, and A.-G. Lang, *Statistical power analyses using g* power 3.1: Tests for correlation and regression analyses*, Behavior research methods, 41 (2009), pp. 1149–1160.

[39] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, et al., *Optimis: A holistic approach to cloud service provisioning*, Future Generation Computer Systems, 28 (2012), pp. 66–77.

[40] N. Ferrier and C. E. Haque, *Hazards risk assessment methodology for emergency managers: A standardized framework for application*, Natural hazards, 28 (2003), pp. 271–290.

[41] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg, *Cloudmf: applying MDE to tame the complexity of managing multi-cloud applications*, in 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, IEEE, 2014, pp. 269–277.

[42] J. A. Fonollosa, *Conditional distribution variability measures for causality detection*, in Cause Effect Pairs in Machine Learning, Springer, 2019, pp. 339–347.

[43] D. Franceschelli, D. Ardagna, M. Ciavotta, and E. Di Nitto, *Space4cloud: A tool for system performance and costevaluation of cloud systems*, in Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds, 2013, pp. 27–34.

[44] S. K. Garg, S. Versteeg, and R. Buyya, *Smicloud: A framework for comparing and ranking cloud services*, in 2011 Fourth IEEE International Conference on Utility and Cloud Computing, IEEE, 2011, pp. 210–218.

[45] C. Glymour, K. Zhang, and P. Spirtes, *Review of causal discovery methods based on graphical models*, Frontiers in genetics, 10 (2019), p. 524.

[46] O. Goudet, D. Kalainathan, P. Caillou, I. Guyon, D. Lopez-Paz, and M. Sebag, *Learning functional causal models with generative neural networks*, in Explainable and Interpretable Models in Computer Vision and Machine Learning, Springer, 2018, pp. 39–80.

[47] A. Gretton, K. Fukumizu, C. H. Teo, L. Song, B. Schölkopf, and A. J. Smola, *A kernel statistical test of independence*, in Advances in neural information processing systems, 2008, pp. 585–592.

[48] P. Gupta, A. Seetharaman, and J. R. Raj, *The usage and adoption of cloud computing by small and medium businesses*, International journal of information management, 33 (2013), pp. 861–874.

[49] Y. Hagmayer, S. A. Sloman, D. A. Lagnado, and M. R. Waldmann, *Causal reasoning through intervention*, Causal learning: Psychology, philosophy, and computation, (2007), pp. 86–100.

[50] J. D. Hamilton, *Time series analysis*, vol. 2, Princeton New Jersey, 1994.

[51] A. Harvey, *Time series forecasting based on the logistic curve*, Journal of the Operational Research Society, 35 (1984), pp. 641–646.

[52] J. L. Henning, *Spec cpu2006 benchmark descriptions*, ACM SIGARCH Computer Architecture News, 34 (2006), pp. 1–17.

[53] B. Hirchoua, B. Ouhbi, and B. Frikh, *Deep reinforcement learning based trading agents: Risk curiosity driven learning for financial rules-based policy*, Expert Systems with Applications, 170 (2021), p. 114553.

[54] S. L. Ho and M. Xie, *The use of ARIMA models for reliability forecasting and analysis*, Computers & industrial engineering, 35 (1998), pp. 213–216.

[55] P. O. Hoyer, D. Janzing, J. M. Mooij, J. Peters, and B. Schölkopf, *Nonlinear causal discovery with additive noise models*, in Advances in neural information processing systems, 2009, pp. 689–696.

[56] N. Huber, M. von Quast, M. Hauck, and S. Kounev, *Evaluating and modeling virtualization performance overhead for cloud environments*, CLOSER, 11 (2011), pp. 563–573.

[57] L. N. Hyseni and A. Ibrahimi, *Comparison of the cloud computing platforms provided by Amazon and Google*, in 2017 Computing Conference, IEEE, 2017, pp. 236–243.

[58] A. Iosup, R. Prodan, and D. Epema, *IaaS cloud benchmarking: approaches, challenges, and experience*, in Cloud Computing for Data-Intensive Applications, Springer, 2014, pp. 83–104.

[59] I. Jacobs, L.-N. Lee, A. Viterbi, R. Binder, R. Bressler, N.-T. Hsu, and R. Weissler, *CPODA-a demand assignment protocol for SATNET*, in Proceedings of the fifth symposium on Data communications, 1977, pp. 2–5.

[60] R. D. Jones, Y.-C. Lee, C. Barnes, G. W. Flake, K. Lee, P. Lewis, and S. Qian, *Function approximation and time series prediction with neural networks*, in 1990 IJCNN International Joint Conference on Neural Networks, IEEE, 1990, pp. 649–665.

[61] J. M. Kanter and K. Veeramachaneni, *Deep feature synthesis: Towards automating data science endeavors*, in 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2015, pp. 1–10.

[62] T. Kayhan, T. Kayhan, and E. Yarbaşi, *Profit management in the case of financial distress and global volatile market behaviour: Evidence from borsa istanbul stock exchange*, Theoretical & Applied Economics, 26 (2019).

[63] I. M. Keseler, A. Mackie, M. Peralta-Gil, A. Santos-Zavaleta, S. Gama-Castro, C. Bonavides-Martínez, C. Fulcher, A. M. Huerta, A. Kothari, M. Krummenacker, et al., *Ecocyc: fusing model organism databases with systems biology*, Nucleic acids research, 41 (2013), pp. D605–D612.

[64] S. Khalid, T. Khalil, and S. Nasreen, *A survey of feature selection and feature extraction techniques in machine learning*, in 2014 science and information conference, IEEE, 2014, pp. 372–378.

[65] A. Khayer, M. S. Talukder, Y. Bao, and M. N. Hossain, *Cloud computing adoption and its impact on SMEs' performance for cloud supported operations: A dual-stage analytical approach*, Technology in Society, 60 (2020), p. 101225.

[66] K. S. Killourhy and R. A. Maxion, *Comparing anomaly-detection algorithms for keystroke dynamics*, in 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, IEEE, 2009, pp. 125–134.

[67] K. Kira and L. A. Rendell, *A practical approach to feature selection*, in Machine learning proceedings 1992, Elsevier, 1992, pp. 249–256.

[68] A. Klinke and O. Renn, *A new approach to risk evaluation and management: Risk-based, precaution-based, and discourse-based strategies 1*, Risk Analysis: An International Journal, 22 (2002), pp. 1071–1094.

[69] S. Kpotufe, E. Sgouritsa, D. Janzing, and B. Schölkopf, *Consistency of causal inference under the additive noise model*, in International Conference on Machine Learning, PMLR, 2014, pp. 478–486.

[70] C. Laaber, J. Scheuner, and P. Leitner, *Software microbenchmarking in the cloud. How bad is it really?*, Empirical Software Engineering, 24 (2019), pp. 2469–2508.

[71] C. Lee and D. A. Landgrebe, *Feature extraction based on decision boundaries*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 15 (1993), pp. 388–400.

[72] A. Lee-Post and R. Pakath, *Cloud computing: a comprehensive introduction*, in Security, Trust, and Regulatory Aspects of Cloud Computing in Business Environments, IGI Global, 2014, pp. 1–23.

[73] P. Leitner and J. Cito, *Patterns in the chaos—a study of performance variation and predictability in public IaaS clouds*, ACM Transactions on Internet Technology (TOIT), 16 (2016), pp. 1–23.

[74] A. Li, X. Zong, S. Kandula, X. Yang, and M. Zhang, *Cloudprophet: towards application performance prediction in cloud*, ACM SIGCOMM Computer Communication Review, 41 (2011), pp. 426–427.

[75] X. Li, X. Li, Y. Tan, H. Zhu, and S. Tan, *Multi-resource workload mapping with minimum cost in cloud environment*, Concurrency and Computation: Practice and Experience, 31 (2019), p. e5167.

[76] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, *Performance overhead comparison between hypervisor and container based virtualization*, in 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), IEEE, 2017, pp. 955–962.

[77] D. Lin, A. C. Squicciarini, V. N. Dondapati, and S. Sundareswaran, *A cloud brokerage architecture for efficient cloud service selection*, IEEE Transactions on Services Computing, 12 (2016), pp. 144–157.

[78] Z. Liu, M. S. Squillante, and J. L. Wolf, *On maximizing service-level-agreement profits*, in Proceedings of the 3rd ACM conference on Electronic Commerce, 2001, pp. 213–223.

[79] S. N. Luko, *Risk management terminology*, Quality Engineering, 25 (2013), pp. 292–297.

[80] M. Lungarella, K. Ishiguro, Y. Kuniyoshi, and N. Otsu, *Methods for quantifying the causal structure of bivariate time series*, International journal of bifurcation and chaos, 17 (2007), pp. 903–921.

[81] S. Luo, Z. Lin, X. Chen, Z. Yang, and J. Chen, *Virtualization security for cloud computing service*, in 2011 International Conference on Cloud and Service Computing, IEEE, 2011, pp. 174–179.

[82] X. Ma, Z. Tao, Y. Wang, H. Yu, and Y. Wang, *Long short-term memory neural network for traffic speed prediction using remote microwave sensor data*, Transportation Research Part C: Emerging Technologies, 54 (2015), pp. 187–197.

[83] A. Maydeu-Olivares, D. Shi, and A. J. Fairchild, *Estimating causal effects in linear regression models with observational data: The instrumental variables regression model*, Psychological methods, 25 (2020), p. 243.

[84] J. Mei, K. Li, A. Ouyang, and K. Li, *A profit maximization scheme with guaranteed quality of service in cloud computing*, IEEE Transactions on Computers, 64 (2015), pp. 3064–3078.

[85] J. Mei, K. Li, Z. Tong, Q. Li, and K. Li, *Profit maximization for cloud brokers in cloud computing*, IEEE TPDS, (2018).

[86] P. Mell, T. Grance, et al., *The NIST definition of cloud computing*, (2011).

[87] N. Moniz, P. Branco, and L. Torgo, *Resampling strategies for imbalanced time series forecasting*, International Journal of Data Science and Analytics, 3 (2017), pp. 161–181.

[88] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, *Gmone: A complete approach to cloud monitoring*, Future Generation Computer Systems, 29 (2013), pp. 2026–2040.

[89] R. Nemani, *The journey from computer time-sharing to cloud computing: A literature review*, International Journal of Computer Science Engineering & Technology, 1 (2011).

[90] NIST Cloud Computing Security Working Group, *Nist cloud computing security reference architecture*, tech. rep., National Institute of Standards and Technology, 2013.

[91] Paper Resource Webpage, *Alibaba cluster data*, 2017.

[92] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, *Introducing stratos: A cloud broker service*, in 2012 IEEE fifth international conference on cloud computing, IEEE, 2012, pp. 891–898.

[93] J. Pearl, *Causality*, Cambridge university press, 2009.

[94] J. Pearl and D. Mackenzie, *The book of why: the new science of cause and effect*, Basic books, 2018.

[95] M. J. D. Powell, *Restart procedures for the conjugate gradient method*, Mathematical programming, 12 (1977), pp. 241–254.

[96] S. Pravilovic, A. Appice, and D. Malerba, *Integrating cluster analysis to the arima model for forecasting geosensor data*, in International Symposium on Methodologies for Intelligent Systems, Springer, 2014, pp. 234–243.

[97] J. Qiu, Q. Du, K. Yin, S.-L. Zhang, and C. Qian, *A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications*, Applied Sciences, 10 (2020), p. 2166.

[98] M. M. Rahman, R. Thulasiram, and P. Graham, *Differential time-shared virtual machine multiplexing for handling QoS variation in clouds*, in Proceedings of the 1st ACM multimedia international workshop on Cloud-based multimedia applications and services for e-health, 2012, pp. 3–8.

[99] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, *Heterogeneity and dynamicity of clouds at scale: Google trace analysis*, in Proceedings of the third ACM symposium on cloud computing, 2012, pp. 1–13.

[100] O. Roger and D. Cliff, *A finance brokerage model for cloud computing*, JoCCASA., 1 (2012).

[101] M. Silva, M. R. Hines, D. Gallo, Q. Liu, K. D. Ryu, and D. Da Silva, *Cloudbench: Experiment automation for cloud environments*, in 2013 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2013, pp. 302–311.

[102] A. Singh and K. Chatterjee, *Cloud security issues and challenges: A survey*, Journal of Network and Computer Applications, 79 (2017), pp. 88–115.

[103] P. J. Smith, A. Firag, P. A. Dmochowski, and M. Shafi, *Analysis of the M/M/N/N queue with two types of arrival process: applications to future mobile radio systems*, Journal of Applied Mathematics, 2012 (2012).

[104] W. R. Smith, *Product differentiation and market segmentation as alternative marketing strategies*, Journal of marketing, 21 (1956), pp. 3–8.

[105] S. Son, G. Jung, and S. C. Jun, *An sla-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud provider*, The Journal of Supercomputing, 64 (2013), pp. 606–637.

[106] P. Spirtes and K. Zhang, *Causal discovery and inference: concepts and recent methodological advances*, in Applied informatics, vol. 3, Springer, 2016, p. 3.

[107] A. N. Tantawi, *Solution biasing for optimized cloud workload placement*, in 2016 IEEE International Conference on Autonomic Computing (ICAC), IEEE, 2016, pp. 105–110.

[108] A. Thomann, *Appendix to'is trading indicator performance robust? Evidence from semi-parametric scenario building'*, Evidence from Semi-Parametric Scenario Building'(January 2, 2019), (2019).

[109] J. N. Tsitsiklis and B. Van Roy, *Analysis of temporal-diffference learning with function approximation*, in Advances in neural information processing systems, 1997, pp. 1075–1081.

[110] Z. ur Rehman, F. K. Hussain, and O. K. Hussain, *Towards multi-criteria cloud service selection*, in 2011 fifth international conference on innovative mobile and internet services in ubiquitous computing, Ieee, 2011, pp. 44–48.

[111] B. Varghese, O. Akgun, I. Miguel, L. Thai, and A. Barker, *Cloud benchmarking for performance*, in 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, IEEE, 2014, pp. 535–540.

[112] T. Velte, A. Velte, and R. Elsenpeter, *Cloud computing, a practical approach*, McGraw-Hill, Inc., 2009.

[113] J. Vrijling, W. Van Hengel, and R. Houben, *A framework for risk evaluation*, Journal of hazardous materials, 43 (1995), pp. 245–261.

[114] S. S. Wagle, M. Guzek, P. Bouvry, and R. Bisdorff, *An evaluation model for selecting cloud services from commercially available cloud providers*, in 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2015, pp. 107–114.

[115] X. Wang, S. Wu, K. Wang, S. Di, H. Jin, K. Yang, and S. Ou, *Maximizing the profit of cloud broker with priority aware pricing*, in 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2017, pp. 511–518.

[116] Wang W and Niu D and Li B and Liang B, *Dynamic cloud resource reservation via cloud brokerage*, in 2013 IEEE 33rd ICDCS, 2013.

[117] W. W. Wei, *Time series analysis*, in The Oxford Handbook of Quantitative Methods in Psychology: Vol. 2, 2006.

[118] S. Weichwald, M. E. Jakobsen, P. B. Mogensen, L. Petersen, N. Thams, and G. Varando, *Causal structure learning from time series: Large regression coefficients may predict causal links better in practice than small p-values*, in NeurIPS 2019 Competition and Demonstration Track, PMLR, 2020, pp. 27–36.

[119] A. S. Weigend, *Time series prediction: forecasting the future and understanding the past*, Routledge, 2018.

[120] J. E. White, *Rfc0105: Network specifications for remote job entry and remote job output retrieval at ucsb*, 1971.

[121] P. J. Williamson, *Cost innovation: preparing for a 'value-for-money'revolution*, Long Range Planning, 43 (2010), pp. 343–353.

[122] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. Phan, I. Lee, and O. Sokolsky, *RT-Open stack: CPU resource management for real-time cloud computing*, in 2015 IEEE 8th International Conference on Cloud Computing, IEEE, 2015, pp. 179–186.

[123] J. Xiao and Z. Want, *A priority based scheduling strategy for virtual machine allocations in cloud computing environment*, 2012 CSC, (2012), pp. 50–55.

[124] J. Yan, R. Kowalczyk, J. Lin, M. B. Chhetri, S. K. Goh, and J. Zhang, *Autonomous service level agreement negotiation for service composition provision*, Future Generation Computer Systems, 23 (2007), pp. 748–759.

[125] L. Yermal and P. Balasubramanian, *Application of auto ARIMA model for forecasting returns on minute wise amalgamated data in nse*, in 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), IEEE, 2017, pp. 1–5.

[126] S. Yousif and A. Al-Dulaimy, *Clustering cloud workload traces to improve the performance of cloud data centers*, in Proceedings of the World Congress on Engineering, vol. 1, 2017, pp. 7–10.

[127] K. Yu, X. Guo, L. Liu, J. Li, H. Wang, Z. Ling, and X. Wu, *Causality-based feature selection: Methods and evaluations*, ACM Computing Surveys (CSUR), 53 (2020), pp. 1–36.

[128] S. L. Zeger, *A regression model for time series of counts*, Biometrika, 75 (1988), pp. 621–629.

[129] J. Zhang, N. Xie, X. Zhang, and W. Li, *An online auction mechanism for cloud computing resource allocation and pricing based on user evaluation and cost*, Future Generation Computer Systems, 89 (2018), pp. 286–299.

[130] I. Zlateva, N. Nikolov, M. Alexandrova, and V. Raykov, *Constructing an algorithm for selecting the number of histogram bins in statistical hypothesis testing for normal distribution of sample data*.