Imperial College London
Department of Computing

Optimal Dynamic Resource Allocation in Multi-Class Queueing Networks

MEng Individual Project Report

Diagoras Nicolaides

Supervisor: Dr William Knottenbelt
Second Marker: Dr Jeremy Bradley

15^{th} June 2010

Abstract

We consider the problem of dynamic resource allocation in a multi-class system under transient arrival streams. We propose a system consisting of a number of servers, which can be switched on and off dynamically, which is used to service paying customers. Different classes of service level agreements are offered, each with different charges for servicing customers and different penalties, in the form of discounts, for failing to meet agreed quality of service requirements.

A series of different policies to make server allocation decisions, based on different parameters, are proposed, implemented and evaluated by simulation. Our goal is to maximize profits by dynamically changing the number of active servers, subject to financial parameters.

The JINQS library has been extended to support the necessary features required for the defined model, in order to evaluate the policies effectiveness under different scenarios. The results of several simulations are presented and evaluated.

Acknowledgments

I would like to thank my supervisor, Dr. William Knottenbelt, for his support, guidance and enthusiasm throughout this project. It has made a world of difference.

I would also like to thank my second marker, Dr. Jeremy Bradley and Dr. Tony Field, for their assistance and inspiration on how best to move forward.

Finally, I'd like to thank my friends and family who offered me helpful advice throughout the project. A special thank you to Laura for her endless help.

Contents

Chapter 1

Introduction

1.1 Motivation

This project is motivated by the rapid growth of distributed systems and cloud computing and the endless endeavour to maximize profits, but can also be extended to real life situations such as banks, hospitals and many others.

Organizations provide services to customers who expect the service to have a certain standard, which typically translates to short waiting times. This requires organizations to increase costs to keep this standard through more service providers. Considering a popular web service for example, customers could be served faster if more servers were available but this would in turn decrease profits due to the extra server running costs, so a balance must be found between customer satisfaction and service provider costs. Intuitively, this relationship is inversely proportional.

The matter in question involves a service provider who provides a certain service to a large number of customers and how best to dynamically vary the number of resources to maximize profits as well as meet customer demands. Consider the above example of the popular web service where the number of servers which are servicing customers is dynamic and each server incurs a running cost. When demand is high, more servers can be turned on and when demand decreases they can be turned off accordingly. Using this technique, customers can remain satisfied and profits high, given that costs from the active servers will vary according to demand.

These economic factors imply that customers and service providers must agree on a *Service Level Agreement* (SLA) which formalize their obligations. The obligations can vary depending on context and will be customized to our problem later on. The system will distribute the jobs amongst the available resources and attempt to offer the best *Quality of Service* (QoS).

1.2 Approach

The aim of the project is to develop a series of algorithms, or policies, to dynamically allocate number of resources to meet customer QoS demands and maximize profits under transient customer arrival rates. To avoid confusion, 'allocation of resources' will refer to the varying number of servers which are accepting customers, rather than allocation of servers to customer classes as, for example, in [1]. Past work on the subject has mainly at-

tempted to switch a fixed number servers between customer classes, using queueing theory or other decision frameworks such as Markov Decision Process and dynamic programming, to tackle the problem. The approach of dynamically switching servers on and off under the proposed model has not been dealt with before, therefore each policy will provide a different approach to tackling the problem so we can investigate the pros and cons of each technique and how close it is to optimal allocation.

The policies will be evaluated under a software based simulation framework, namely JINQS, through a number of different simulations with varying parameters. JINQS is extended in various different ways to meet our modelling demands as well as provide a graphical user interface to assist with the running of the simulations and interpretation of results.

1.3 Report Structure

Before discussing existing work and the planned work on this project, an introduction to theoretical concepts used throughout the project is presented (Chapter 2). This will provide a clear overview on queueing theory and the economic factors relating to it, as well as help with analysis of existing work which follows. We continue to define the model parameters and SLAs under which we evaluate the algorithms (Chapter 3). The implementation details concerning the framework and the policies implemented are described in the Chapters 4 and 5 respectively, with appropriate diagrams for the ease of the reader. Chapter 6 describes the simulations and the outputted results, followed by individual evaluations of each policy. Finally the report ends with an analysis of the policies (Chapter 7) and a summary and conclusions section, with future work which could be done on the underlying system.

1.4 Contributions

We aim to answer the question for service providers *'how can I change the vary number of servers I have switched on to earn more money, without sacrificing customer satisfaction?'*. The novelty behind the project is the fact we will use transient arrival streams to model multi-class customer arrivals. We examine the effects of different strategies and if predictive planning is worthwhile in the face of uncertainty.

Chapter 2

Background To Queueing Systems

2.1 Queueing systems

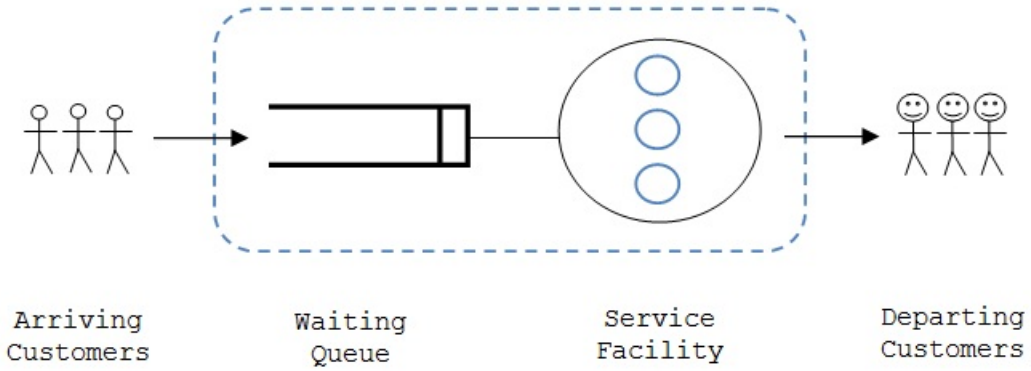


Figure 2.1: Queueing system schematic

A queueing system can be summarized as a system where customers arrive according to an 'arrival process' to be serviced by the service facility, which can have one or more servers. If a customer arrives to be serviced but finds all servers are busy, he/she joins a waiting queue to be serviced later. This service facility serves customers as they arrive or, if customers are waiting in the waiting queue, serves them first according to some service discipline. The customer then leaves the service facility upon completion of his/her service. This process is depicted in Figure 2.1. To avoid confusion, throughout the project we will use generic terms to describe the above system. We will use *customers* in a general sense to refer to processes which require some service (not necessarily a human) and *servers* or *instances* for the service facility which provides this service. The *system* will be used to refer to the queue along with the service facility (blue box above). For example, in a bank, the customers are the people entering the bank to be served, the tellers are the servers and the system is the building itself. A queueing system can be broken down into three main components; *the input process*, *the system structure* and *the output process*.

2.1.1 Input Process

The input process refers to customer arrival into the system. With reference to the schematic, it is the eclipse and arrow on the left which points into the system. The arrival process can be broken down into three aspects:

1. **The size of the arriving population**
   This is the group of customers who wish to use the service being provided. There are two types of arriving customer population sizes, *finite* and *infinite*. Finite customer population size involves small numbers of customers arriving into the system (which consequently affects the arrival rate). Infinite customer population in fact means a large number of customers, which for mathematical and analytical purposes is treated as infinite.

2. **Arriving patterns**
   Customers arriving into the system come at random, unpredictable rates. For our modelling purposes, we will fit a statistical distribution to model the inter arrival distribution. The arrival process can be characterized as *steady state* if the average rate of arrival remains constant for a sufficient period of time, otherwise it is characterized as *transient* [2]. Some probability distributions which are commonly used and will be explained in more detail later are listed below:

   M: Markovian/Memoryless (Poisson process)
   D: Deterministic (constant inter-arrival times)
   E_k: Erlang distribution of order K of inter-arrival times
   SMPP: Sine Modulated Poisson Process
   MMPP: Markov Modulated Poisson Process

3. **Customer behavior**
   Customers who arrive into the system may find that all servers are currently busy servicing other customers and will be required to join the waiting queue. In some systems, called *blocking systems*, the queue will have a maximum limit by which all customers trying to join the queue when it is full will be declined service. *Non-blocking systems* imply that queues can grow infinitely long.

2.1.2 System Structure

The system structure refers to the physical number and properties of the server and the queue capacity. Servers in the service facility can be in *parallel* where all servers provide the same type of service and a customer only needs to pass through one server to complete service, or *serial* whereby a customer must pass through several servers before completing service [3]. As already mentioned, queue capacity can be limited or infinite.

2.1.3 Output Process

The output process refers to departing customers. This depends on the service behavior, namely the following two factors:

1. **Queueing discipline**
   This factor is also called *serving discipline* and refers to the way customers in the

waiting queue are chosen for service. The most common such discipline is the *first-come-first-served* (FCFS) with lots of other variations. Priority queueing discipline is also an frequently encountered queueing discipline of interest by which multiple classes of customer can join the same system. There are two sub-classes of the priority queueing discipline, namely *preemptive* and *non-preemptive*. Preemptive discipline implies that if a customer of a certain class is being serviced and a customer of higher priority arrives at the system, the customer with the lower priority will stop being serviced to service the higher priority customer. This can further sub-classed to *preemptive resume* and *preemptive restart* which refer to whether or not the customer who had his/her service preempted will continue processing from where it left off or the service will start from the beginning. Non-preemption is the opposite of preemption strategy which means that once a customer is being serviced, his/her service cannot be interrupted, even if a higher priority customer arrives.

2. **Service-time distribution**
   Different customers have different service time requirements. In other words, some customers take longer being serviced than others. This variation in service time must be modelled using a probability distribution for analytical purposes. The most commonly assumed service time distribution is the negative exponential distribution with many others available:

   M: Markovian/Memoryless (exponentially distributed)
   D: Deterministic (constant service times)
   E_k: Erlang distribution of order K service time
   G: General service times distribution.

2.2 Kendall Notation

The Kendall Notation is the standard notation used for showing the five elements which completely describe a queueing system and will be used throughout this project. The Kendall Notation is as follows:

A / B / X / Y / Z

where:
A: Customer arrival rate
B: Service rate
X: Number of parallel servers
Y: Queue capacity
Z: queueing discipline

If a multi-class customer base is used, the factors which are dependent on customer class are assigned a subscript i. This only applies to the customer arrival rate (A) and the service rate (B). For example, M_i/M/1 would indicate that each class of customer has different interarrival times whilst their service times are identical regardless of class.

2.3 Resource Utilization and System Stability

$$\rho = \frac{\lambda}{m\mu} \tag{2.1}$$

To assist with further discussion on queueing theory, we must introduce the notion of resource utilization and system stability. Equation (2.1) is the utilization formula for m servers and should be less the one in order for the system to be stable. In other words, for a stable queueing system, the rate of customers arriving (λ) must be less than the service rate ($m\mu$) to cope with demand, otherwise customers are arriving faster than the system can deal with. Therefore, if $\lambda > m\mu$ then there will be steady increase of customers in the system, whilst $\lambda < m\mu$ is impossible. $\lambda = m\mu$ is known as the *flow conservation law* and gives rise a measure called *traffic intensity*. Traffic intensity shares the same equation as utilization but is measured in *erlangs*. The ceiling value ($\lceil \lambda \,/\, m\mu \rceil$)is used in many queueing systems to determine the minimum number of servers required to support demand. For example, 4.5 erlangs means that a minimum of five servers is required to keep the system stable.

2.4 Little's Theorem

$$N = \lambda W \tag{2.2}$$

Otherwise known as Little's Law or Little's Lemma, John Little proved that the long-term average number of customers (N), in a stable non-preemptive system, is equal to the long-term average arrival rate (λ) multiplied by the long-term average time a customer spent in the system (W) (Equation 2.2). What is important about the theorem is that it shows that this behavior is independent of the distribution of the arrival rate, the queueing discipline and the number or servers in the system. Therefore it can be applied universally to all queueing systems with proper allocation of N, λ and W.

2.5 Service Level Agreement (SLA) and Quality of Service (QoS)

The problem we wish to address is how to make the system as profitable as possible through the optimal allocation of available resources and keeping customer satisfaction high. A SLA will act as a binding contract between customer and service provider to formalize the economic parameters of the service. Typical values in the SLA include the charge on the customer for using the service and a penalty on the service provider if a maximum waiting time is exceeded. The QoS is the measure by which a customer is considered happy with the service. When the QoS deteriorates, the service provider will usually end up paying a penalty. It is important to note that SLA and QoS can be defined in various different ways and we will proceed to define them later in our model.

2.6 Previous Work

2.6.1 Existing Commercial Systems

Several commercial cloud computing platforms exist but perhaps the most famous cloud computing platform is Amazon's EC2 [4]. Users can determine the number and type of instances they need to meet their requirements. The feature of interest is the Auto Scaling feature, which allows the number of instances to dynamically increase automatically according to the users criteria. Users use an API to specify the criteria for which instances are switched on and off resulting in a probable use of a sub-optimal strategy with room for improvement. The most common criteria used for switching instances on and off is the CPU utilization whereby the user sets an upper and lower bound percentage for which instances can be switched on and off in order to keep CPU utilization within the bounds. Dynamic allocation policies can be applied to guarantee maximum revenue for the provider as well as a minimization of costs for the user since no more than the required number of instances will be switched on. [5] provides a good insight into several more cloud computing infrastructures as well as an overview of their structure and business model.

2.6.2 Dynamic Allocation Policies

Focus on Revenue Maximization

Past work, such as [6, 7] and the initial work done by last year's Imperial graduate, also define SLA and QoS contracts to attempt to find algorithms and heuristic policies to maximize revenue. They attempt to tackle the problem of dynamic allocation of resources by implementing a search algorithm to find the local maximum of revenue using the admission limit and number of servers through dynamic programming. An optimal allocation and associated admission limit pair is derived from the algorithm to maximize revenue. It is important to note that no mathematical proof of this proposition is provided, but is only confirmed through multiple numerical experiments. In the experiments, [8] uses only a single class of customers whilst [6, 7] provide simulations for up to 2 types of customer. A Poisson distribution is used to model all arrivals. [6, 7] prove that the search for optimal values is of linear complexity but for large enough values a heuristic sub-optimal policy is required for improved efficiency. By varying the system load, the heuristic strategy is compared to the optimal strategy to confirm its reliability. [8] develops multiple algorithms which are evaluated through simulation. It is shown that the algorithm which incorporates a long term predictive spanning tree may provide better results, rather than the short-term counterpart. Problems relating to fast switching of servers due to transient nature of the arrival streams are also seen in the simulations but are not taken into consideration by this algorithm. Extensions to [8] involve developing an algorithm for optimal allocation for a more complex model by adding support for multi-class customers and linear rewards and penalties. Dynamic resource allocation using the model used in this project does not seem to have been studied before.

[9] introduces two dynamic forms of system control. The first is service rate control, whereby an average cost optimal stationary policy is generated through the changing of service rates of the M/M/1 queue under the Poisson arrival model. Each of the available service rates has a cost associated to it and the problem is analyzed under different scenarios on when to best change service rate. Essentially this policy can be translated into a dynamic allocation system by converting the service rates to 'number of servers' with

constant service rates. The second dynamic system control involves a M/M/K queue with a dynamic service pool where the average cost optimization of continuous time processes is analysed and an optimal dynamic allocation policy described. For a Poisson arrival rate of a single class queueing system, the optimal pool size for any number of customers in the system and current active servers is found. The optimal policy for a service pool with maximum K servers and associated running and holding costs is shown for 7 scenarios in the format of [*minimum number of servers to be switched on*](*number of customers to indicate when it is optimal to switch servers on/off*). For example, [2](4,6,8,4,2,1) would indicate that a minimum of 2 servers must be switched on. When the queue length increases to 4, another server must be switched on. The same applies for when the queue length reaches 6 and finally when it reaches 8 customers, all servers should be switched on. Similarly when it reaches 4 customers, a server should be turned off. When it decreases to 2, then the fourth server should be switched on until finally the queue length reaches 1 by which the third server should be turned off leaving the minimum number of servers switch on. It is clear from this example that the optimal policy will result in the system state being transient. In our model of transient arrival streams it is possible that the process starts in a transient state and so (i.e in middle of the second term of the above stated format) it is suggested for the optimal policy to be effective, the controller must immediately switch all servers on until the queue is empty and then switch the required number of servers off to remain at the minimum number of servers required with queue length equal to 0. This algorithm can be altered to match our model and its performance evaluated against other algorithms developed.

Focus on Variable QoS and SLA

Contrary to our model, it is possible for users to negotiate SLAs and QoS metrics with the providers. The users can consider the notion of a utility function to determine the best possible combination of services as a function of various SLAs. [5] looks into the use of these utility functions to determine the optimal mix of SLAs in cloud computing. The problem is tackled using MS Excel to selected optimal values of SLAs to maximize the utility function subject to SLA and cost constraints as a non-linear constraint optimization. From a provider point of view, autonomic techniques such as control theory, machine learning and combinatorial search methods combined with queueing network models are proposed to minimize costs (in fact, the latter set of methods have been successfully applied by the authors and their colleagues in a variety of settings). This work gives insight to a different model and approach to our problem.

[10, 11] attempt to formalize the resource allocation problem for SLA constrained environment and try to find the optimal resource allocation that minimizes total cost. [11] uses a multi-resource model which makes the problem NP-hard and tackles it through a framework for designing heuristics. The model used to solve the problem makes this technique unfit for our purposes. On the other hand, [10] uses different application environments running on a grid and using combinatorial search technique of all possible configuration vectors, decides the optimal allocation of resources to these environments in a way that maximizes the utility function. This technique is similar to what is used in [6, 7] but under a different model.

Dynamic vs Static Policies

[12] also attempts to tackle the problem of dynamic server allocation using statistical metrics analysis to compare static server allocations to dynamic server allocation. Using a constant ratio $\alpha = \frac{numberofcustomersinsystem}{numberofserversonduty}$ to determine the number of active servers, the advantages of dynamic server allocation are realised by showing that dynamic server systems can advertise that a customer will be served in less than 3 minutes with 99.6%-99% confidence (depending on α) whilst with fixed server systems, this can only be guaranteed with 10 active servers and with only 95.5% confidence. However, economic factors are not taken into consideration and this strategy will be far from optimal.

2.6.3 Transient Arrival Rates

Previous work typically assume a Poisson arrival rate but this simple arrival model is unrealistic given that usually traffic arrival processes are often correlated. Therefore, correlated arrival rates are modelled using MMPP. Analyzing long term statistical values of a MMPP system can provide good starting points for algorithm development. However, [13, 14] both warn of the complexity of trying to analytically reason about such systems since it is a computationally intensive task. For this reason, [13] deems analytical reasoning of such systems unfit for QoS oriented design and instead attempts to approximate a MMPP/M/1 queue as a weighted superposition of different M/M/1 queues. Using GRID server analysis to model its usage as an MMPP, results show that the error associated with average response time and queue length computed through the unbiased approximation, never grows larger than 3%. The analysis of MMPP could lead to more informed decisions to be made and improve long run profits. [14] provides numerical techniques for solving the Kolmogorov differential equations, which provide transient solutions. The iterative numerical methods techniques provided may take too long to provide solutions and will end up providing an approximation which could lead to a malinformed decision being made.

Chapter 3

The Model

In this section we define the various parameters of our model. They are the parameters which will be later on implemented for simulation and evaluation of the various algorithms. In our model we will assume multi-class customers which, for simplicity, we will limit to a maximum of three classes ($i = 0,1,2$). We use the usual convention to number the priority classes so that the smaller the number, the higher the priority [14].

Our focus will be a discrete time controlled stochastic system where the controller only has partial system information (we assume that only expected service times are available to us). More precisely, we use a *Markov Decision Process*. At each time step, the system is in a state with a list of possible decisions it can choose from which will take it to the next state. Note that the current state and chosen action, is conditionally independent of all previous states and actions. Stochastic dynamic programming will be used for the implementation of the algorithms.

3.1 Queue System Model

3.1.1 Input Process

The input process will consist of a multi-class infinite arriving customer population joining a single infinite capacity queue. For this project we are only interested in transient systems with variable arrival rates therefore we will only focus on a subset of the arrival processes already mentioned which display this transient nature.

By understanding typical arrival patterns, we can analytically derive performance measures which can directly affect the allocation of resources. It is expected that an algorithm will be developed using these measures. It is important to note that the algorithm itself is unaware of the arrival pattern which will be used for each customer class.

Most models adopt a simple arrival model (i.e [6, 7]), namely the Poisson process (M in Kendall Notation). Its Markovian characteristic, that arrivals occur independently of one another, is not always adequate for modelling complex systems such as multi-media traffic or use of a web service. Furthermore, the constant arrival rate make the Poisson process adequate for modelling 'non-bursty' arrival, but unrealistic for modelling complex systems where the arrival rate changes throughout the operation of the system. We will use the two most common input models with transient behaviour, which are used in most queueing simulations due to their close correlation to real world situations. For example, [13] uses a MMPP model for GRID computing usage, while [15] uses a MMPP to model the empirical evidence gathered from 1025 computers users using with World Wide Web

over a 24 hour period. The second transient model which will be used due to its wide use for approximating individual IP-flows and web services usage, is the *non-homogeneous Poisson process.*

1. **Poisson Process**
   The Poisson Process plays a pivotal role in classical queueing theory. When the inter-arrival times are assumed to be exponentially distributed the arrival process is Poisson. It is commonly used due to the fact it closely resembles many physical phenomenon and is considered to be a good model for an arriving process that involves a large number of similar and independent users [2]. To characterise an arrival process as Poisson, only the arrival rate λ is required.

$$P[X = k] = \frac{\lambda^k}{k!} e^{-\lambda} \tag{3.1}$$

2. **Markov Modulated Poisson Process (MMPP)**

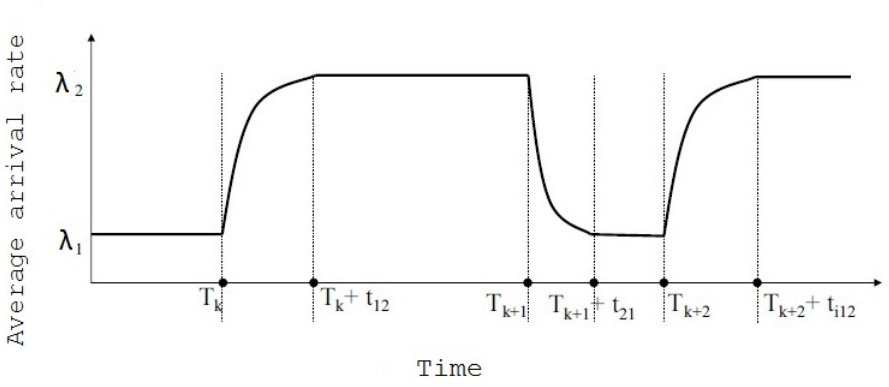


Figure 3.1: 2 state MMPP extracted from [13]

A Markov Modulated Poisson Process (MMPP) is a Poisson process whose rate varies according to a Markov process. The arrivals are therefore generated by a source whose stochastic behaviour is governed by an *m-state* irreducible continuous time Markov process, which is independent of the arrival process [2]. The MMPP can be fully characterized by three parameters. The initial state of the MMPP, the *infinitesimal generator* ($\bar{Q}$), which is also known as the transition-rate matrix and the Poisson arrival rates ($\bar{\Lambda}$).
$\bar{Q}$ defines the probabilities of changing state, given the current state. It is a square matrix of size m for a *m-state* system. With reference to Figure (3.1), the infinitesimal generator would be a 2-by-2 matrix with the rows representing current states and the columns representing the other possible states. Therefore, matrix position (i,j) represents the probability of transitioning to state j whilst in state i. For our purposes, we assume a homogeneous $\bar{Q}$, meaning that transitions do not vary with time.

$$\bar{Q} = \begin{pmatrix} -q_1 & q_{12} & \cdots & q_{1m} \\ q_{21} & -q_2 & \cdots & q_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m1} & q_{m2} & \cdots & -q_m \end{pmatrix}$$

where

$$q_i = \sum_{j=1 j \neq i}^{m} q_{ij} \tag{3.2}$$

The Poisson arrival rates linked to each state are defined as a diagonal matrix $\bar{\Lambda}$ or a vector $\bar{\lambda}$ such that when the Markov chain is in state j, arrivals occur according to a Poisson process of rate λ_j.

$$\bar{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_m \end{pmatrix}$$

$$\bar{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_m)^T \tag{3.3}$$

A common MMPP used to model bursty arrival of packets [16] is the Interrupted Poisson Process (IPP). It is defined as a two state MMPP where one state has an arrival rate of 0.

3. **Non-homogeneous Poisson Process**

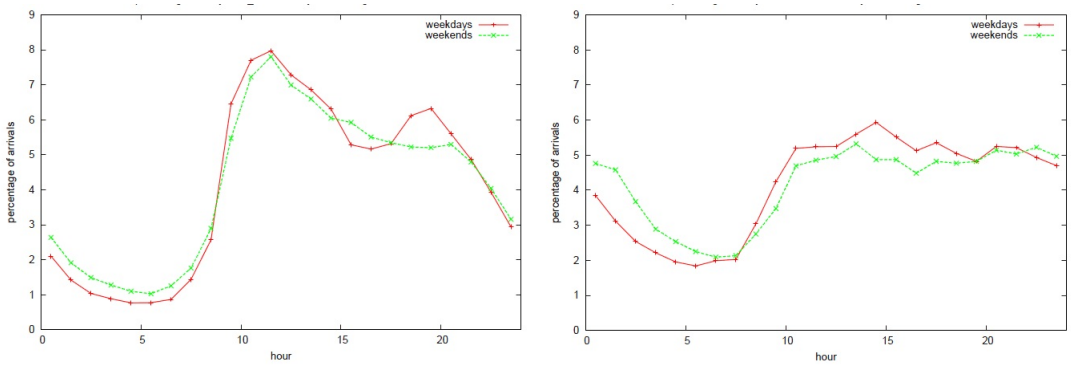


Figure 3.2: Plots of the percentage of walk-in (left) and ambulance (right) arrivals by hour over weekdays and weekends for 2002-2007, extracted from [17]

The final transient arrival pattern we will use will follow a *non-homogeneous Poisson Process* where the arrival rate $\lambda(t)$, is a function of time. More specifically we focus on the *Sine Modulated Poisson Process* (SMPP). Consider a popular web service and the arrival of customers to that web service throughout an average working day. Intuitively, the number of customers will be at its peak during working hours and

then fall to a lower bound at night. The same applies to other real life situations such as hospitals (even though our model of costs and discounts may be not be fit for such a real life application, we are only interested in the arrival scheme so as to use more realistic arrival models). [17] uses empirical evidence of walk-in patients and ambulances at a London hospital between 2002-2007. Referring to Figure 3.2, the average walk-in patient arrivals are shown on the left and can be approximated to a SMPP due to similarity to the sine wave. The average ambulance arrivals shown on the right however cannot be modelled as such and so, during our simulations, high priority customers will probably not be modelled using SMPP.
If we consider a shorter period of time t, such as hour, the average rate of change of the number of customers arriving into the system n during the period t could be approximated to a constant parameter (i.e $\frac{dn}{dt} \approx 0$). Using this strategy, we will select a window of arbitrary size where we will monitor the arrival of customers and determine the average λ in order to approximate the system to a M/M/m system. Due to the short time of the sampling interval taken to make our calculations, the error involved with this technique could be significant, leading to an unacceptable suboptimal allocation of resources, therefore a revision of this theory may be required. Perhaps event-driven bookkeeping could be implemented where the program monitors arrival rates on weekdays and improves it predictive values in the long run.

3.1.2 System Characterization

Given the aim of this project is to identify the optimal allocation of resources, which in our system will vary with time, there is no fixed number of resources. We will further assume, although unrealistic, an infinite pool of resources whose elements can be switched on/off upon request, is available, although an optional maximum limit will be provided in the simulations. These servers will be parallel and identical, which in this context means that a server can serve a customer of any class. Furthermore, one server can service one customer at a time. We will use a non-blocking system with non-preemptive priority queueing.

3.1.3 Output Process

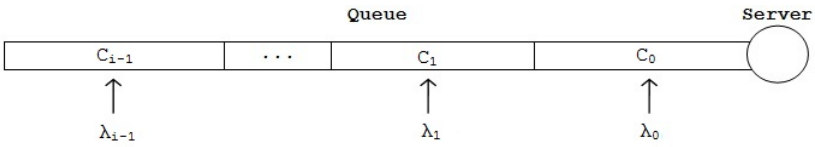


Figure 3.3: Priority queueing system schematic

We will assume customers of the same class have the same service time requirements and will follow exponentially distributed service times (M_i). We impose the head-of-the-line (HOL) queueing discipline on our model, given that it is perhaps the most common priority queueing discipline [18]. Figure (3.3) shows the structure of this queueing discipline where c_j indicates customers of class j ($j = 0..i$-1)and λ_j the arrival rate for the class. It shows that the queue is divided into groups according to customer class, with class 0 in the front of the queue since they have highest priority and class i at the end of the queue.

Any arriving customers join the end of their class so they obey a first-come-first-served discipline within their own class. The system will therefore always choose the 'head' of the queue for service. It is important to note that there are various ways of defining which customers are given higher priority, such as service-time-dependent disciplines, but for our purposes, we give higher priority to the highest paying customer class. The value of a customer's priority will remain constant in time, although it would be interesting to investigate an algorithm that changes the priority of customers.

3.2 SLA and QoS

All customers will need to agree to a SLA to use the service. Each customer class will have different values assigned to the SLA to reflect their difference in priority. For the purposes of this project we assume three different static types of SLA to reflect the three customer classes.

QoS can be measured in various different ways such as in terms response time W (the time between job arrival and job completion) or in terms of waiting time w (the time between job arrival and the beginning of service). For our model we will use waiting time as the measure of QoS.

In some cases, such as [5, 19], it is possible for users to be able to negotiate SLAs with service providers for different QoS metrics. To find the best values for SLA to maximize revenue whilst providing the service at a reasonable price is beyond the scope of this project therefore we will instead use values from Amazon's EC2 service for realism. If the reader wishes to know more on this, [19] studies some issues to be considered when designing grid applications such as QoS metrics and the relationship between QoS and SLA. We define the SLA using the following four parameters:

$$(c_{min}, c_{rate}, q, q_{max}) \tag{3.4}$$

1. **Charge (c_{min} and c_{rate})**
   This is the charge imposed on the customer for using the service. In our model we will use a linear relationship between charge and how long the customer took to be serviced (W-w), with a minimum charge c_{min} and a charge rate of c_{rate}. The gradient of the slope (c_{rate}) increases with priority, indicating customers of higher priority will be charged more as shown in Figure 3.4.

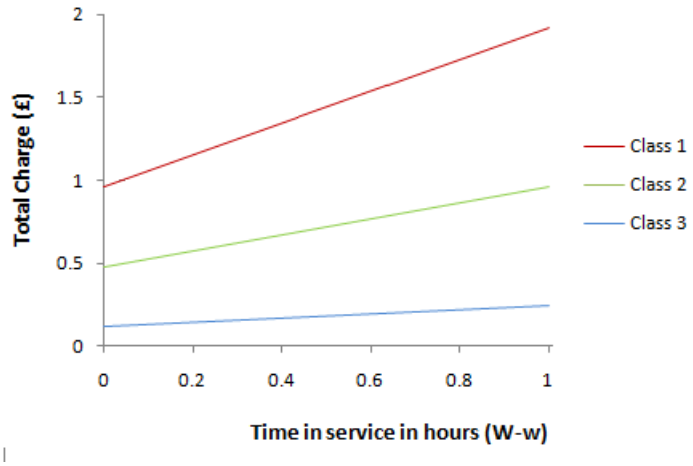


Figure 3.4: Linear relationship between total charge and time being serviced

2. **Obligation (q and q_{max}) and Penalty**
   If the provider fails to start serving customers after their waiting time in the queue has exceed a predefined time q, then a penalty must be paid. This penalty will be in the form of a percentage discount of the total charge imposed on the customer which will have a linear relationship with waiting time. If the customer is not served by q_{max} then the customer is given free service. Reasonable values for q and q_{max} would be depend on class hierarchy and so we will use an increasing waiting time before a penalty is paid as class levels decrease. This is depicted below in Figure 3.5 where the values for obligation are much stricter for higher priorities and relate to average service time.

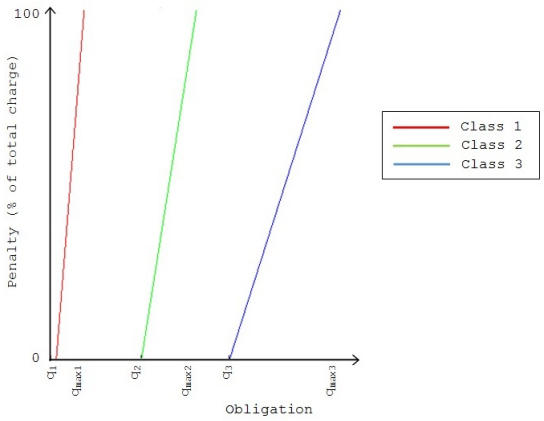


Figure 3.5: Linear relationship between penalty and obligation

3. **Class Parameters**
   The values we will use for SLA agreements comes from the values Amazon uses for Windows Standard On-Demand Instances. We will assume that higher priority customers have higher service requirements and therefore are willing to pay the extra amounts as necessary. We define our SLA parameters as follows:

$$\text{Class } 0 : (0.96,\ 0.96,\ 1/\mu_0\ ,\ 2/\mu_0)$$

Class 1 : (0.48, 0.48, $2/\mu_1$, $4/\mu_1$)
Class 2 : (0.12, 0.12, $3/\mu_2$, $6/\mu_2$)

3.3 Assumptions

We will simplify the model by making the following basic assumptions. Customers, upon arrival, cannot leave the system until they have been serviced. We further assume no failures of any kind such as network or server failures and delays due to network data transfer are negligible. These assumptions will allow us to concentrate on the factors we are interested in and easily analyze results.

The service provider has two sources of cost, namely *running costs* and *switching costs*. Each server which is active will incur a running cost of $\boldsymbol{u}$ per unit time whilst switching a server on/off will also incur a cost of $\boldsymbol{s}$. Unfortunately no realistic values are available so we use reasonable values of $u = 0.05$ and $s = 0.1$. The values were selected so that it is financial beneficial to switch on a server to serve a customer rather than pay a penalty.

3.4 Performance measures of the a M/M/1/∞/Pr

Although our model is concerned with transient arrival streams, steady-state numerical analysis will prove useful as a foundation and can be used for bookkeeping purposes. As shown in Figure 3.1, the arrival rates will not be constantly fluctuating and so can be considered being in a *state* ($T_k + t_{12}$ up to T_{k+1}) or in a *state transition* (T_k up to $T_k + t_{12}$). When the average arrival rate remains constant we can use M/M/1 and M/M/m measures to reason numerically about these systems whilst during state transitions we will use the algorithm proposed by [13] for overestimating the measures. Note that the following equations are for a M/M/1 system only. For a multi-server system the service distribution rate will be adjusted to $m\mu$. We assume that n customer priority classes exist, each with its own arrival rate λ_i ($i = 1..n$) and average service time distribution $\bar{x}_i$. Although customers join the same queue regardless of class, for easier reasoning we will imagine that the single queue is split up into n infinite capacity queues, one for each customer class. Within each of these queues, customers are served using FCFS discipline.

If we take the system as a whole, we define the total arrival rate as $\lambda = \lambda_1 + ... + \lambda_n$ and average service distribution $\bar{x} = 1/\mu$. Utilization of each class of customers is defined as $\rho_i = \lambda \bar{x_i}$ with total utilization of the system defined as $\rho = \sum_{i=1}^{n} \rho_i$. As already mentioned, the total system utilization must be less than unity in order for the system to be stable.

For a customer of class i who arrives at the system, the average queue waiting time is given by:

$$w_i = R + \bar{x}_i N_q^i + \sum_{j=1}^{i-1} \bar{x_j} N_q^j + \sum_{j=1}^{i-1} \bar{x_j} \lambda_j W_i \tag{3.5}$$

The average customer waiting time depends on the following four factors, in the order shown in Equation (3.5):

1. R
   R is known as the *mean residual service time* for all customers in the system and

is the average time required for a customer already being serviced to finish being served. R is given by

$$R = \frac{1}{2} \sum_{i=1}^{n} \lambda_i \left(\frac{1}{\mu} \right)^2 = \frac{\rho}{\mu} \tag{3.6}$$

2. The average service time of the customers of the same class (i) which are already in the system upon arrival.

3. The average service time of the customers in the system with higher priority.

4. The average service time of the customers with higher priority who join the system after the customer has joined the system.

It's obvious that customers of class 1 will not have their waiting time affected by other customers of other classes and so we can safely deduce a simplified average customer waiting time for customer of class 1:

$$w_1 = \frac{R}{1 - \rho_1} \tag{3.7}$$

Once the waiting time is known, several other important performance measures can be found:

1. The average number of customers of each class, in their own queue:

$$(N_q)_i = \lambda_i w_i \tag{3.8}$$

2. The total time a customer of class i spends in the system:

$$T_i = w_i + \bar{x_i} \tag{3.9}$$

3. The total number of customers in the system:

$$N = \sum_{k=1}^{n} (N_q)_k + \rho \tag{3.10}$$

3.5 Revenue Evaluation

For a system with n servers and k customer classes the average revenue per unit time earned from a system can be defined as shown below. Average revenue is derived using waiting time as the QoS measure. K_i is the queue limit for class i.

$$V = \sum_{i=0}^{i=k-1} V_i \tag{3.11}$$

where

$$V_i = \lambda_i \sum_{i=0}^{K_i-1} p_{i,j}[c_i - r_i P(w_{i,j} > q_i)] - su \tag{3.12}$$

$p_{i,j}$ is the stationary probability of j customers of class i in the system which can be found by iteratively solving (using $p_{i,0} = 1$) and normalising the balance equations

$$p_{i,j} = \begin{cases} \rho_i p_{i,j-1}/j & \text{if } j \leq n \\ \rho_i p_{i,j-1}/n & \text{if } j \geq n \end{cases}$$

We can derive the conditional waiting time, $w_{i,j}$

$$P(w_{i,j} > q_i) = \begin{cases} 0 & \text{if } j < n \\ e^{-n\mu q_i} \sum_{k=0}^{j-n} \frac{(n\mu q_i)^k}{k!} & \text{if } j \geq n \end{cases}$$

The above formula is derived from the Erlang distribution with parameters $(j-n+1, n\mu)$. It, in combination with Equation 3.11, can be used to quickly compute average revenue and find the local maximum of the function of K_i. This value of K_i is the optimal limit for a class, whereby accepting more customers after that limit will result in a decrease in profit. This is the basis of the optimal allocation algorithm which determines when best to switch servers on/off.

3.6 Optimal Criterion

Like most work on this subject, our ultimate goal is to find the optimal allocation policy for the maximization of profits (Equation 3.11) which is our optimization criterion. Such a policy, under our model, does not yet exist due to its complex nature and is practically impossible to calculate analytically. One approach to find optimal allocation would involve trying out all possible combinations of servers at each step of a run, whilst holding a history of all steps already taken, to find which path maximized our revenue. This exhaustive search would require huge amounts of processing and memory therefore heuristics may be used for improvement.

The main focus will be on the Markov decision process which will aim at maximizing the revenue function. Intuitively one can see that for any system state with multi-class customers, there is an optimal number of servers to service these customers so that maximum profit is achieved.

Chapter 4

Extending JINQS

Lots of packages exist for the simulation and modelling of queues, such as MATLAB [20], simul8 [21], risk, but unfortunately fall short in flexibility to meet our requirements. Main issues involve lack of adaptability to implement our modelling needs as well as issues with implementing a dynamic resource controller. JINQS [22, 23], a JAVA based extensible library for simulating multi-class queueing networks is available to be extended and adapted to our model. Although very powerful, several changes were required to fully meet our demands. These changes are explained below through the use of UML diagrams as well as a description of the changes. For the reader's ease, any item in the UML diagrams with a blue gradient existed in the original JINQS and was not changed for our purposes. Due to the complexity of some objects and their relationships, only the important and relevant factors are included in the UML diagrams, with details omitted.

To ensure progress is secure, a SVN repository will be set up on

`svn+ssh://svnuser.doc.ic.ac.uk/homes/dn106/svn.repository`

which is where the end result will be available for submission or if the reader wishes to view the source code evolution.

4.1 Introduction to JINQS

JINQS is an event-based simulation package. The classes `Event` and `Sim` can be used to construct the event-driven simulation, with network package being built on top of these. Simulations work by setting up the network and scheduling arrival instances of the `Event` class in a hidden time line and the processing these events to generate exit instances of the `Event` class. The `Sim` and `Network` classes are the 2 central classes of package and contain only static methods so that they may be invoked from anywhere without having to pass around instances of these objects explicitly.

4.2 New Features

The new features to JINQS include a user interface with graphical representation of results. Minor changes include additional packages to support economic concepts of revenue, penalties and profits for a simulation and file logging capabilities to record various events of statistical importance. The major changes to the JINQS library involve changing the whole structure to allow for the running of multiple concurrent simulations and to allow

dynamic changing of resources during a run. In combination, all the above changes allow for policies to be implemented to monitor and alter the system accordingly, such that profits are maximized.

4.3 GUI Elements

4.3.1 User Interface

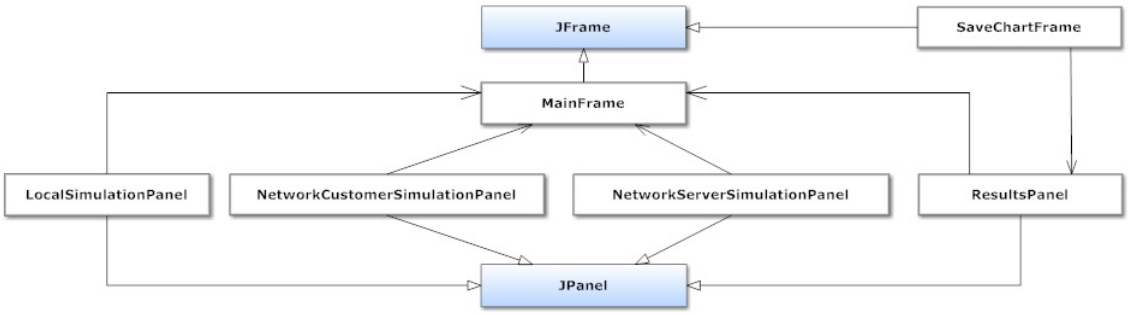


Figure 4.1: UML of user interface

The large number of simulations which would be run with a large set of variables suggested that a user interface would be beneficial. The user interface has 4 tabs: Local Simulation, Network Simulation (Server), Network Simulation (Customer) and Results. The first three are for the user to input the parameters of the current run such as class inter-arrival, batch and service settings, SLA settings and the simulation settings. The options offer as much flexibility as possible for each simulation, however, problems encountered when trying to fit so many options in a compact window (size was chosen for users with 1024x600 resolution to able to view whole form). To get around this problem, the form changes dynamically according to the user settings. For example, if the user chooses the Erlang distribution as a class service setting, he/she will be requested to input k and θ, whilst if he/she chose the exponential distribution, then the relevant part of the form would change so only λ would be needed. The Results panel displays the results of each simulation, whether it is the financial results such as running and switching costs, amount paid in penalties, revenue and profits or any other properties of interest. The user can also pick from a selection of different charts to view the relevant information. Screenshots of the user interface are included in the Appendix.

4.3.2 Charts

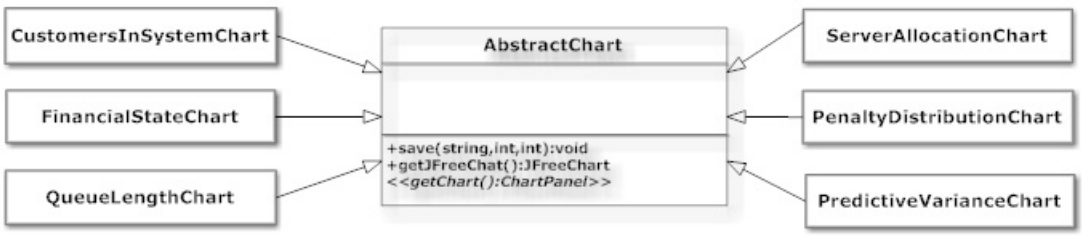


Figure 4.2: UML of charts

The charts are supported by the *JFreeChart* package [24]. Each time a simulation is run, information is logged to file which can then be interpreted for statistical purposes or plain interest. The charts take full advantage of the strengths and flexibility of the package and allow the user to view and save charts. Furthermore, the user can combine the data from different runs into a single chart. For example, the user can run a simulation under different policies and view the profit results from each policy on a single chart for easy comparison. The following charts are available:

1. Server Allocation - A line graph showing the number of active servers throughout a run. This can give insight to better strategies or even, as we will see later, defects of a given policy.

2. Customers In System - A line graph that shows the number of customers from each class in the system at any point in time. Colour coding is supported for each class so the user can read the graph easily. This graph can provide valuable information on class distinction within the system.

3. Penalty Distribution - A bar chart, divided into intervals of 10%, ranging from 0-100%, which indicates how many customers of each class received the relevant percentage penalty. This is the second most important benchmark for a policy as it directly reflects the ability of the system to deal with the customers.

4. Queue Length - A line graph the shows the queue length over time. This provides 2 important insights into the system: how well it deals with customers and, in combination with the 'Customers in System' graph, an insight to customer class distinction.

5. Financial State - The most important graph is the financial state graph. It is a line graph which displays profits over time. The user can run the same simulation with different policies and the results will be automatically added to the graph. This graph is *the* benchmark for the project, as our aim is to maximize profits.

6. Predictive Variance - Given that most of the algorithms rely on predictive techniques, it is useful to see by how much the profits from the same algorithm vary, given the same run. This line graph displays these fluctuations over time graphically to identify the volatility of each predictive policy.

4.4 Economic Factors

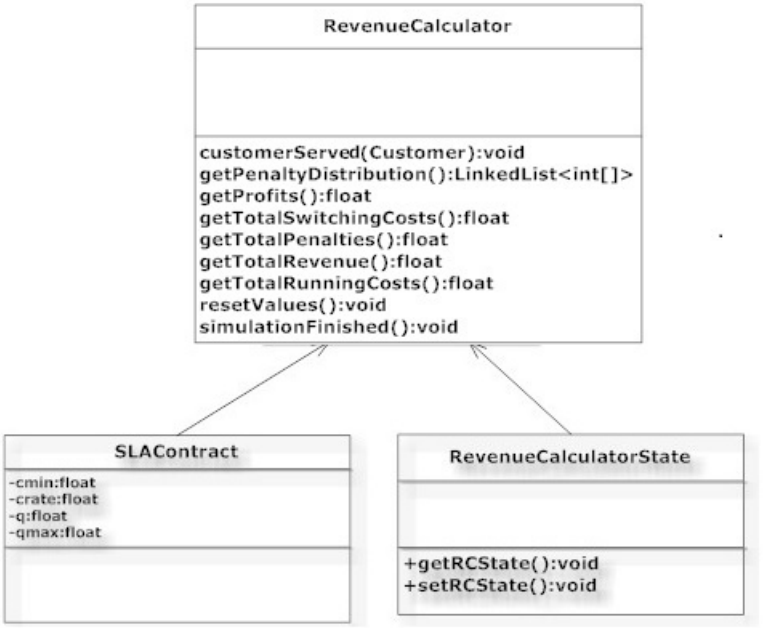


Figure 4.3: UML of classes which handle economic factors

The library needed to be changed to support economic factors such as running and switching costs, revenue and penalties. This required the addition of the SLA package which includes the 3 classes shown in the above UML diagram.

4.4.1 SLAs

The `SLAContract` class contains the charges and obligations of each class added by the user through the user interface. It holds the values discussed in the tuple 3.4.

4.4.2 Revenue Calculator

This is a central class to the whole financial system. It is responsible for handling all financial information relating to a run. It contains only static fields and methods to handle events of allocation change (to handle switching and running costs) and customers exiting the system (to calculate revenue and penalties). The `customerServiced` method is shown below in pseudo-code, with reference Figure 4.4.

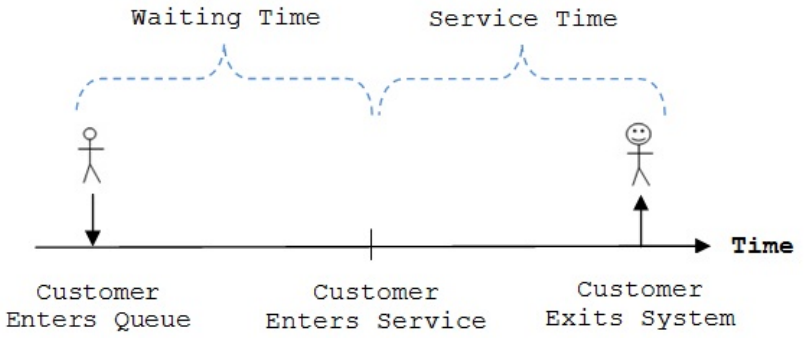


Figure 4.4: Customer traversal within the system

Pseudo-code for customerServed(Customer c) method

```
fetch customer SLA;
customerRevenue = cmin + (service time * crate);
if waitingTime > q
  if waitingTime > qMax
    discount = 1;
  else
    discount = (waitingTime - q) / (qMax - q);
  customerPenalty = customerRevenue * discount;
totalRevenue += customerRevenue;
totalPenalties += customerPenalty;
```

Furthermore, there are static methods to retrieve financial information during a run (i.e total running costs, profits etc.) so that policies can retrieve this information and make more informed decisions.

The `RevenueCalculatorState` class saves the state of the `RevenueCalculator` using the method `getRCState()` so a given policy could alter the system in a given way and then set the state back to its original state using `setRCState()`. For example, some of the policies implemented try out 'test runs' with a different number of servers to see which allocation yields the highest profits. To enable this, the state of the `RevenueCalculator` must be saved and restored accordingly by using the 2 methods in the `RevenueCalculatorState`.

4.5 Additional Distributions

4.5.1 Markov Modulated Poisson Process (MMPP) and Sine Modulated Poisson Process (SMPP)

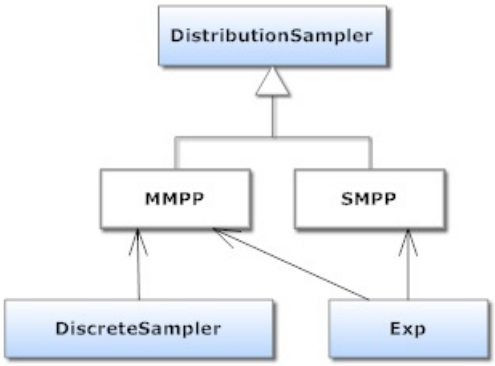


Figure 4.5: UML of additional distributions

Our simulations will mainly be concerned with transient arrival schemes. As already explained, MMPP and SMPP have qualities which make them ideal for our simulations. The user can input the parameters of the distribution through the user interface (if input is a matrix, then space separated values in the order of row by row is required) and using already existing classes from within the JINQS library, the MMPP and SMPP distributions were implemented.

4.5.2 Replay of Last Arrival Stream

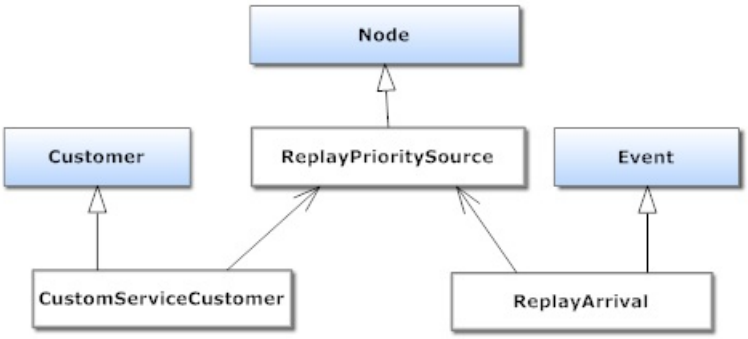


Figure 4.6: UML of classes responsible for replaying an arrival stream

A desirable feature in order to benchmark the various policies is to replay the same arrival streams and see how each policy performed. Each time a simulation is run, except if it is a replay, each customer arrival time, class and service time is recorded to a file. This enables the `ReplayPrioritySource` class to read in the file and schedule the same arrival events of customers with the same service and arrival times for the next run. Two additional classes were required to schedule the arrivals this way: `CustomServiceCustomer` and `ReplayArrival`. JINQS samples the distribution for service times when the customer

is selected to enter the system, therefore, a new type of customer was required where the service time could be set through the event. The `ReplayArrival` event is used to build customers of type `CustomServiceCustomer`, with the relevant parameters so as to replicate the previous arrival stream. Note that this is not the same as using a seed for the sampling of the distributions as probabilistic, non-deterministic paths are used for the MMPP process.

4.6 Network/Distributed Simulations

To fully examine the effectiveness of the optimal allocation algorithm,we can relax some of our assumptions and simulate the queueing system over a local area network (most probably labs) and see how this affects performance. Various different techniques were tried out to set up such a system such as remote procedure calls to indicate the arrival of customers, TCP connections to the customer sources and UDP datagrams. The RPC brought up several issues relating to security which could pose problems when running the simulations of different networks, therefore was rejected. TCP connections were also rejected due to the fact that it set a limit of the number of customer sources as well as increased implementation complexity (threads were required to monitor different ports resulting in an increase of resources too). UDP datagrams were chosen because they are easy to implement, can be used on any network without complications or overhead without sacrificing usability. However, they are not reliable for large number of hops and some packets might get lost. Each UDP datagram is used to indicate a customer arrival from a source. Thorough exception handling has been set up throughout the network simulations, with meaningful error messages, to allow the user to run the simulations as smoothly as possible.

4.6.1 Server

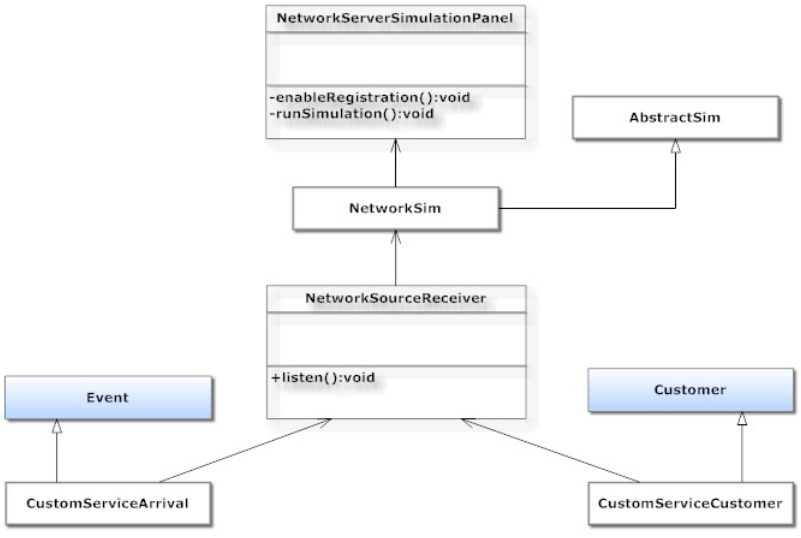


Figure 4.7: UML of network server simulations

The settings for running the server side of the network simulations can be changed from the 'Network Simulation (Server)' tab of the user interface. The user must set up the

simulation settings and then select a port to open a connection to the customers. There are 2 stages to run a remote simulation. The first is the customer registration using `enableRegistration()`, whereby a new `NetworkSim` object is created which in turn spawns a thread to monitor the port for customer arrivals (`listen()`) record their arrivals in the system. The second stage is the running of the simulation through the `runSimulation()` method and the results generated locally.

4.6.2 Customer

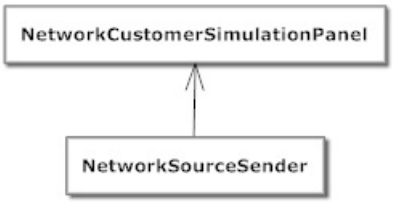


Figure 4.8: UML of network customer simulations

The network customer settings can be set through the 'Network Simulation (Customer)' tab. The options available are more flexible than the local simulation to allow for further relaxation of our assumptions. For example, the user can selected the time at which arrivals begin (rather than always begin at time 0). When the user inputs the settings and starts the sending of customers, a new thread is spawned to send the UDP packages. The user can see the progress of the sending through a logging textbox used for feedback.

4.7 Dynamic Allocation

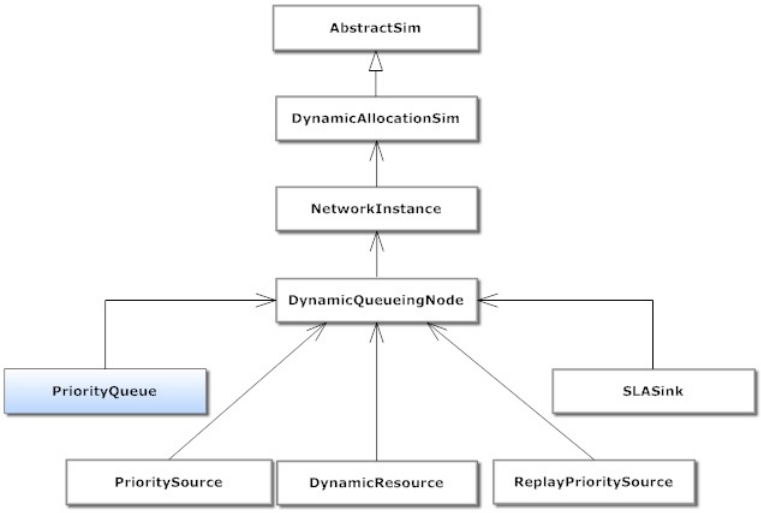


Figure 4.9: UML of classes required to support dynamic structure

To allow a variable number of servers during a run, several classes needed to be added with the corresponding functionality. The 2 main classes which support this functionality are the `DynamicQueueingNode` and the `DynamicResource`. They are the same as their

static counterparts other than a couple of methods which needed changing in order to allow resources to be added/removed dynamically. An initial approach was to model these resource changes as events in the system, however this approach lacked in flexibility as the changes in the system would be delayed with respect to customer events and was therefore rejected.

The main challenge was adding/removing resources and having the system respond to this change immediately, without waiting for the next event to occur. This was achieved by making the changes just before the `DynamicQueueingNode` checked for available resources upon customer arrival in the `accept()` method, as shown below.

Pseudo-code for accept method

```
alert allocation policy of customer arrival;
if resource available
   service customer;
else
   enqueue customer;
```

After the policy has been alerted, it will make the decision on the new allocation and change the resources accordingly through the `setResources(int)` method in the `DynamicQueueingNode`. The pseudo-code below shows the `setResources(int)` method and how the system handles changes in allocation. After the number of resources (total and free) are updated accordingly, any free resources are assigned to customers. The system then follows to record these changes and continues on. The `DynamicQueueingNode` has been implemented in such a way to allow unlimited resources to be used. The policies are responsible for setting the upper limit of resources and keep the resources within the allowed range.

Pseudo-code for setResources(int *n*) method

```
let nresources = total resources;
let resources = free resources;
if n > nresources
   resources += (n - nresources);
else if n < nresources
   resources -= (nresources - n);
nresources = n;
while (queue is not empty && resources are available)
   assign resource to head of queue;
record new allocation to file;
record switching costs;
```

The allowed range has as an upper bound the maximum number of available resources and as a lower bound 1 resource. This brings forth a limitation of the system whereby the policies cannot decrease the number of servers upon customer arrival, only upon exit. This is a reasonable assumption since an increasing number customers implies more servers are required. Furthermore, from an implementation point of view, the only server which can be switched off is the one that has just been freed by the exiting customer since the other servers will be servicing other customers.

4.8 Policy Enforcement

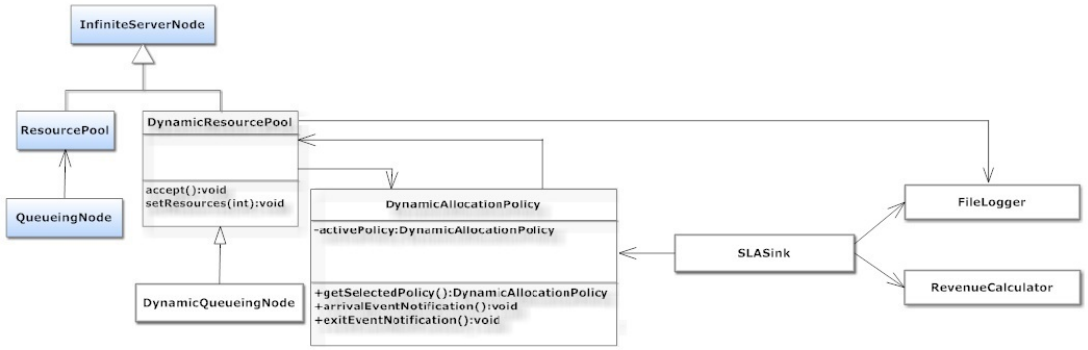
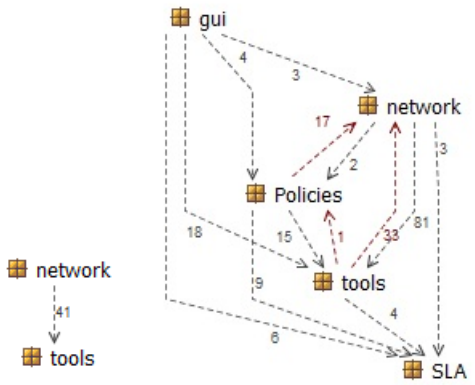


Figure 4.10: UML of classes responsible for policy enforcement

Each policy inherits from the abstract class `DynamicAllocationPolicy`. The abstract class has `activePolicy` as a private field which points to the instance of the active policy to alert it of any events of interest. The `DynamicResourcePool` alerts the policy of arrival events and the `SLASink` alerts the policy of exit events through the static methods `arrivalEventNotification()` and `exitEventNotification()` respectively. The policy can then decide of the number of servers at each event and change the allocation through the `setResources(int)` method in the `DynamicResourcePool`. This strategy is depicted in the UML diagram above. The system has been structured in such a way that adding a policy is as simple as creating a class which extends the `DynamicAllocationPolicy` abstract class and adding an instance of the policy to the static policies list in the abstract class. The user interface will be updated automatically and the user can use the new policy. It is also worth noting that the `DynamicResourcePool` records arrival events and the SLASink records exit events to file through the relevant static methods in the `FileLogger` class. In addition, the `SLASink` alerts the `RevenueCalculator` of a customer exit event using the `customerServed(Customer)` static method, so the economic factors can be calculated.

4.9 Overall Architecture



(a) Before (b) After

Figure 4.11: Dependecies between packages

The original JINQS architecture (Figure 4.11a) was designed in such a way, that only one simulation could run at a time. This was due to the 2 critical classes `Network` and `Sim` having only static methods and fields. To allow more power and flexibility for policy implementation, JINQS was changed to allow the running of multiple concurrent simulations. After discussing the changes with Dr. Field, we concluded that having instantiable classes saving the states of the 2 static classes and then restoring them when necessary, would allow for multiple concurrent simulations to be run with careful use of their states. This strategy has the advantage that minimal changes to the structure of the library were required but had as a disadvantage a performance decrease due to the time consuming deep copying of objects. This technique was successfully implemented and worked in some cases, however, as the system grew more complex, it started to fail on occasion. This implementation is left in place as a proof of concept in case the reader wishes to review the code.

This implied a new approach was necessary. The best approach was to drastically change the JINQS library in order to allow instantiable classes of the 2 static classes. In the end, 2 classes were added, namely `AbstractSim` and `NetworkInstance`, such that they are exactly the same as their static counterpart, but can be instantiated. The next step was to change *all* classes to support these changes, while still allowing the running of simulations using the static classes. These changes were necessary since each object needs to know which simulation and which network it belongs to. The changes were successfully implemented resulting in the structure seen above on the right. From the dependency graph it is obvious that the consequence of these changes was that cyclic dependencies emerged (shown in red above). Admittedly, better planning of the changes could have resulted in fewer dependencies.

Chapter 5

Implemented Policies

A total of 6 policies were implemented, each providing useful information with regard to policy strategy and optimal allocation. Below is a description of each policy along with the respective pseudo-code. It is important to note that the pseudo-code is a generalization of the actual implemented policy, in order to abstract away implementation details and focus on the functionality of the policy. The reader is welcome to review the code of the policies, which has been properly commented for the explanation of the implementation details. Furthermore, due to the massive time requirements of the 'Exhaustive Search', no simulations will use it as a policy.

To better describe the strategy undertaken by each policy we consider the timeline below. The blue squares depict 10 customer arrivals and the orange squares depict customers exiting the system, over a given time. For each policy, we assume the system has just accepted its 6^{th} customer and show what is used to make a decision on each event.

Figure 5.1: Event timeline

5.1 General Pruning Technique

Most of the policies which will be described are based on search techniques, each given different information to help make better decisions. Considering the 'branching' of each policy, the worst case scenario is checking every possible allocation on each event. The number of total simulations in this scenario are:

$$(max.\ servers)^{(number\ of\ customers)*2} \tag{5.1}$$

This therefore implies that techniques are required to improve run-time of the simulations, whether using computational techniques or heuristics.

Our decision process is based on the notion that there is an allocation which will yield more profit than any other choice and our goal is to find it, given other sources of information to make a more informed decision.

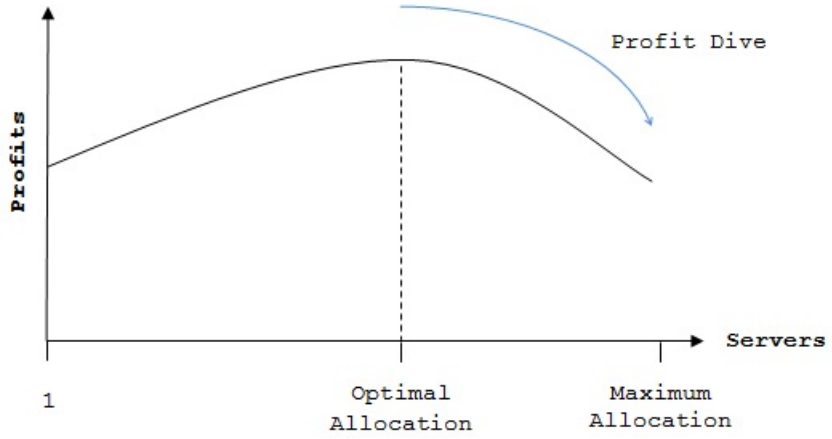


Figure 5.2: Profits Using Different Static Allocations

Experience has shown that given a system with the same customers arriving, the above diagram holds but is highly dependant on the simulation economic parameters. For our purposes, the values used for economic parameters are such that there is one local, and hence global, maximum. This means that we can use a simple pruning technique when searching for this maximum. When searching for the optimal allocation, if we see that 5 consecutive allocations are of decreasing profits (blue line in Figure 5.2), then we can assume we have found the global maximum and can stop the search. This technique will be used in almost all search based policies since experience with running simulations on our model has shown that simulations could take days if the decisions made use small number of servers and hence cause a build up of customers in the queue.

Other techniques were examined for runtime boost performance but proved unsuccessful. For example, some policies cannot be multi-threaded because of simultaneous access of a list. This required locking which eventually slowed performance down rather than improve it. An attempt was made to cache results as a (state,allocation) pair, however the problem of how to define the state accurately to make the correct decision came into play. For example, one could say that the state is defined by the number of customers of each class in the system, however this is not an accurate description due to the various different combinations of arrival and service times.

5.2 System Information Based

These are policies which use system information to make decisions on allocation. We will investigate if local information is enough to make accurate decisions to maximize profits.

5.2.1 Utilization Based Policy

Figure 5.3: Event timeline

This policy is based on the system's utilization (ρ). Amazon's EC2 Auto-Scaling system allows users to dynamically change the number of active resources by setting upper and lower bounds on their utilization. If the system utilization goes above the upper bound percentage, then another instance is switched on, whilst if it goes below the lower bound, an instance is switched off. This policy will allow us to evaluate the given strategy from Amazon's point of view.

Referring to the above timeline, let's assume the system has just accepted its 6^{th} customer. The policy will check the system utilization and increment, decrement or do nothing according to the bounds set.

Let the algorithm parameters u and l be the upper and lower bounds respectively:

Algorithm For Utilization Based Policy

On customer arrival and exit:
if $\rho > u$
  return +1;
else if $\rho < l$
  return -1;

5.2.2 State Evaluation Policy

Figure 5.4: Event timeline

A valid assumption is that inter-arrival times between customers are independent and therefore, we can only focus on the customers currently in the system. This policy takes advantage of this and uses the current system state to determine the best allocation, ignoring all past and future events. This will be useful in evaluating the benefits and drawbacks of any predictive policies.

The policy keeps track of the customers currently in the system using a linked list. On every customer entry and exit event, it will update the list accordingly and run simulations with different allocations to see which allocation yielded the maximum profits. This will be considered the optimal allocation and the policy will continue to change the system

resources according to the given value. As a reminder, switching off a server can only be done on an exit event and only the server which the leaving customer has just freed can be switched off. This is shown in figure 5.4, with events which are not of interest faded out. The events left are the 4 customers currently in the system whose arrivals will be used in the test runs.

Let the algorithm parameter *maxAlloc* be the upper bound of the number of servers:

Algorithm For State Evaluation Policy

```
LinkedList <Customers> currentState;

On customer arrival :
addCustomerToList();
setOptimalAllocation();

On customer exit:
removeCustomerFromList();
setOptimalAllocation();

setOptimalAllocation method:
disableFileLogging();
saveRevenueCalculatorState();
float maxProfits;
int optimalAllocation;
if (exitEvent)
  servers = currentAlloc - 1;
else
  servers = currentAlloc;
end if
while servers <= maxAlloc
  runSim(currentState, servers);
  if (5th consecutive decrease in profits)
    servers = maxAlloc;
  end if
  if (maxProfits < RevenueCalculator.getProfits())
    maxProfits = RevenueCalculator.getProfits();
    optimalAllocation = servers;
  end if
  servers++;
  restoreRevenueCalculatorState();
end while
setResources(optimalAllocation);
enableFileLogging();
```

5.3 Prediction Based

Predicting future arrivals and service times would allow for the system to prepare/adapt accordingly so that profits can be increased in light of this information. These policies

attempt to plan ahead using statistical measures to predict future arrival and service times.

5.3.1 Predictive Planning Policy

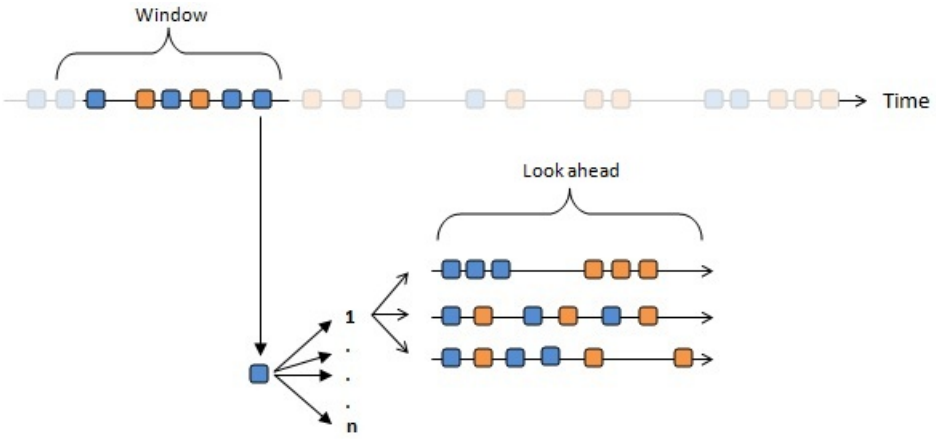


Figure 5.5: Event timeline

This policy uses a pre-defined window (of size 5 in the above depiction) to estimate statistical properties of inter-arrival and service times and attempt to predict future demand using exponential maximum likelihood estimate for the rate parameters; $\hat{\lambda} = 1/\overline{x}$ where $\overline{x}$ is the sample mean. On each arrival and exit event, the policy will take the current system state and will use the estimates to schedule further arrivals (of a pre-defined size). Then, for each allocation within the accepted range, 3 simulations will be run to take the average profit, where the allocation that yielded the maximum profit is considered optimal. The reason for taking the average and using this extra overhead is to allow for the variance in the statistical measures. A 'tweak' was included in the prediction algorithm to allow an increase/decrease of the effects of the variance between each run so that they do not different by much. In the above diagram, 3 different arrival patterns have been estimated for the 3 runs, according to the window statistics (only shown for one allocation - this is done for all possible allocations 1 to n).

The difficulty of the implementation of this approach is the proper handling of events and the allocations linked to these events. A historical record needs to be held with the policy handling past and new decisions. In order to save decisions already made, an internal class `AllocationRecord` is associated with each customer using a hashmap which maps the unique customer id to the respective `AllocationRecord`. The class stores the allocations of his/her arrival and exit events. This way, when test runs are performed, the policy can allocate the resources according to past optimal decisions and new allocations accordingly.

Let the policy parameters *maxAlloc* , *windowSize* , *planAhead* be the maximum number of servers to consider, the window to keep for statistical measures and the number of future arrivals to consider when performing test simulations respectively:

Algorithm For Predictive Planning Policy

```
LinkedList <Customers> currentState;
LinkedList <Customers> window;
HashMap <Integer, AllocationRecord> customerAllocations;

On customer arrival:
addCustomerToCurrentState();
if (window.size() == windowSize)
  removeFirstFromWindow();    addCustomerToWindow();
end if setOptimalAllocation();
updateCustomerAllocationRecrd();

On customer exit:
removeCustomerFromCurrentState();
removeCustomerFromWindow();
setOptimalAllocation();
updateCustomerAllocationRecrd();

setOptimalAllocation method:
disableFileLogging();
saveRevenueCalculatorState();
float maxProfits;
float averageProfits;
int optimalAllocation;
if (exitEvent)
  servers = currentAlloc - 1;
else
  servers = currentAlloc;
end if
while servers <= maxAlloc
  for retry = 1 to 3
    runSim(currentState, servers, window, planAhead);
    averageProfits += RevenueCalculator.getProfits();
    retry++;
  end for
  averageProfits = averageProfits / 3;
  if (5th consecutive decrease in profits)
    servers = maxAlloc;
  end if
  if (maxProfits < averageProfits)
    maxProfits = averageProfits;
    optimalAllocation = servers;
  end if
  servers++;
  restoreRevenueCalculatorState();
end while
setResources(optimalAllocation);
enableFileLogging();
```

5.3.2 Predictive Planning Policy (with threshold)

The predictive planning policy has one major flaw; although it takes into consideration all costs, switching costs could grow rapidly in a transient system. This policy aims at reducing the switching costs by tweaking the above policy as follows: A server will not be switched off unless a certain threshold amount has been earned between the previous event and the current event being considered. A logical value would be 2 times the switching costs so that each server earns the money required to switch it on and off. During the simulations we will investigate if this threshold is beneficial and under which circumstances. The changes to the above policy are shown in pseudo-code below.

Let *threshold* be the threshold amount needed to be earned between events:

Algorithm For Predictive Planning Policy

```
LinkedList <Customers> currentState;
LinkedList <Customers> window;
HashMap <Integer, AllocationRecord> customerAllocations;

On customer arrival:
addCustomerToCurrentState();
if (window.size() == windowSize)
  removeFirstFromWindow();   addCustomerToWindow();
end if setOptimalAllocation();
updateCustomerAllocationRecrd();

On customer exit:
removeCustomerFromCurrentState();
removeCustomerFromWindow();
setOptimalAllocation();
updateCustomerAllocationRecrd();

setOptimalAllocation method:
disableFileLogging();
saveRevenueCalculatorState();
float maxProfits;
float averageProfits;
int optimalAllocation;
if (exitEvent && (RevenueCalculator.getProfits() - previousProfits) > threshold )
  servers = currentAlloc - 1;
else
  servers = currentAlloc;
end if
while servers <= maxAlloc
  for retry = 1 to 3
    runSim(currentState, servers, window, planAhead);
    averageProfits += RevenueCalculator.getProfits();
    retry++;
  end for
  averageProfits = averageProfits / 3;
  if (5th consecutive decrease in profits)
    servers = maxAlloc;
  end if
  if (maxProfits < averageProfits)
    maxProfits = averageProfits;
    optimalAllocation = servers;
  end if
  servers++;
  restoreRevenueCalculatorState();
end while
setResources(optimalAllocation);
enableFileLogging();
```

5.3.3 Predictive Planning Policy (future known)

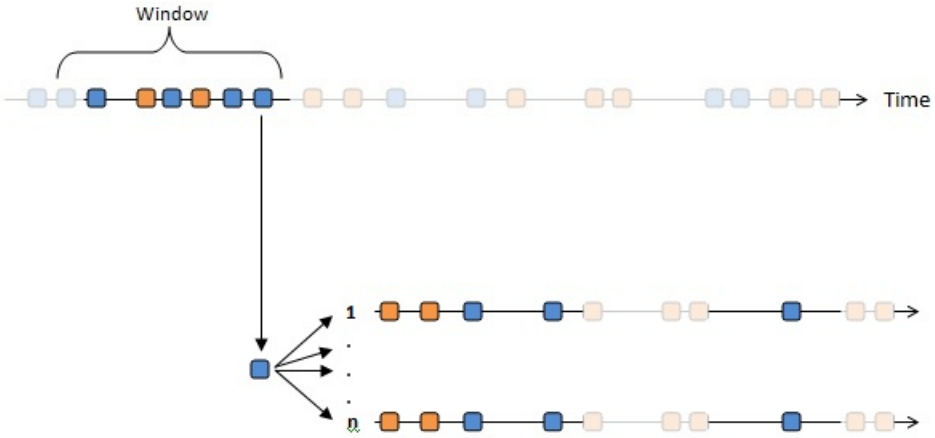


Figure 5.6: Event timeline

In order to evaluate the effectiveness and accuracy of the above prediction algorithms, a 'prediction benchmark' policy was implemented. This policy can only be used when replaying arrival streams. It is essentially the same the the predictive planning policies, except that instead of estimating future arrivals during the test runs, the actual arrivals are used. We can assume that the results produced by this policy are the best we can do in our context. Since this a replay stream it is possible to schedule a pre-defined number of arrivals in the future and have the system adapt to them. The only parameter is the number of arrivals to schedule ahead of the current system state during the test runs. Note that in Figure 5.6, the future events scheduled are identical to the actual *arrival* events (exit events will be different according to current allocation and hence faded out) and they are the same for each allocation, unlike the predictive planning policies up to now. Furthermore, the window used for future scheduling is of size 3, leaving 1 arrival event unscheduled. This is the only search policy which does not incorporate the pruning technique presented in the beginning of the chapter in order to ensure best results. Note that due to the overhead of opening and closing the file to read in the arrival stream, the policy will take a long time to finish a simulation. It is possible to read the arrival stream in memory to overcome this problem but this will increase the program's memory requirements significantly.

5.4 Other

5.4.1 Static Policy

This is the simplest policy possible and has been included for comparison reasons only. Through the simulations we will be able to identify the benefits and drawbacks of having a dynamic system.

5.4.2 Exhaustive Search

Our ultimate goal is to find the optimal resource allocation to maximize profits. In order to discover what this allocation is, an exhaustive search policy was implemented. This

policy will try out all possible combinations of allocations for each event, until the entire state space has been searched, and an optimal strategy discovered.

This policy explodes in size and this exponential growth makes it impossible to discover the global maximum within a logical time frame, even with the fastest computers. In order to improve the timing, some techniques are used to prune away some simulations as well as using a thread pool to run simulations concurrently. Despite this, the policy still takes too long and exists only as proof of concept. As an example, with a maximum of 2 servers and 10 customers, the simulation took on average 4 minutes. If we were to extend this simulation to 500 customers we would need $4x10^{295}$ minutes! The 2 main pruning techniques used are:

- If 5 consecutive decisions have been made such that profits are constantly decreasing, prune away the simulation.
- If for a given event the only cost increasing is the switching cost, the prune away the remaining branches which change the allocations for the given event.

Chapter 6

Simulations and Evaluation of Results

Having extended the JINQS library and implemented the different policies, we now follow to evaluate their effectiveness through 5 different simulations based on the model already described. Each simulation scenario will allow us to examine the policies under different situations of increasing fluctuations in inter-arrival times to see how well they perform. In addition, we will investigate how changing policy parameters affects profits, and determine any values which seem to generate better results. The arrival stream will be generated upon the first run of each simulation (static policy) and then replayed under different policies to see how each policy handles the arrival stream.

Due to the extremely long time the 'Exhaustive Search' policy would take to discover the optimal allocation for a given run, we cannot benchmark the policies against maximum profits. We instead analyse the results of each simulation and compare each policy with each other to determine their strengths and weaknesses.

In addition, for the 2 policies which involve predicting future arrivals, we show 3 runs of the same simulation to further evaluate the volatility of results. Each simulation will have a network consisting of the 3 sources, the HOL queue and a maximum of 50 servers (starting with only 1 instance switched on) and will run until 30 000 customers have been served. The common parameters for all simulations are:

- **Class 0 μ:** 2
- **Class 1 μ:** 2
- **Class 2 μ:** 2
- **SLA Class 0:** (0.96, 0.96, 0.5 , 1.0)
- **SLA Class 1:** (0.48, 0.48, 1.0 , 2.0)
- **SLA Class 2:** (0.12, 0.12, 1.5 , 3.0)

6.1 Simulation 1

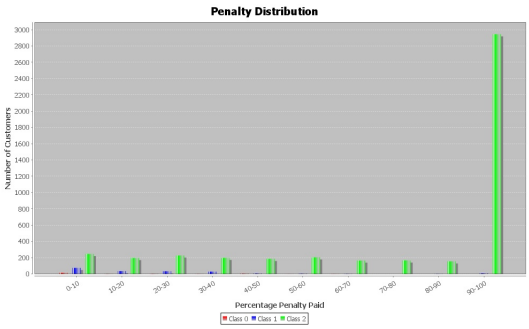
In the first simulation we will use the same exponential arrival streams and exponential service times for all 3 classes. We can use this scenario to evaluate how well policies react to relatively stable arrival streams under relaxed circumstances. For predictive policies, we will investigate the effects on profits of changing window sizes. We use the following parameters to define the simulation:

- **Class 0 λ:** 5
- **Class 1 λ:** 5
- **Class 2 λ:** 5

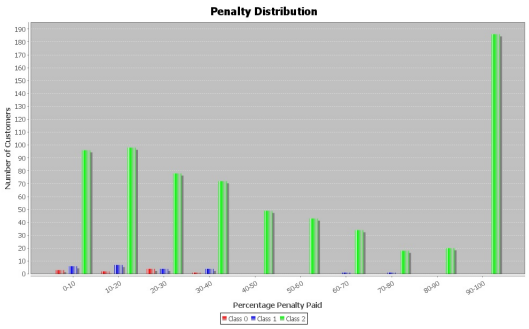
6.1.1 Static Policy

Allocation	Running Costs	Switching Costs	Penalties	Revenue	Profits
8	798.97	0.0	705.97	23269.27	21764.33
9	898.64	0.0	70.18	23269.27	22300.46
10	998.36	0.0	13.55	23269.27	22257.36

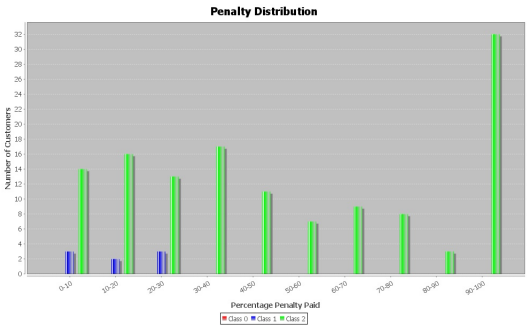
The optimal static allocation for simulation 1 would be 9 servers. The interesting point to note is that maximum profits were achieved even though a relatively small number of penalties were paid (approximately 400 customers got a discount). As expected, this was mainly at the expense of the class with the lowest priority. Figure 6.1 below shows clearly the problems raised when the wrong number of servers are being used. For example, using 8 servers will result in more than 10 times the penalties of the 9 server run, which will in turn lead to high customer dissatisfaction with the service. However, using 10 or more servers results in customers of Class 0 never getting discounts, with only a few Class 1 customers receiving minor discounts, implying high customer satisfaction since there we no or slight delays in their service. If we were to assign a price to this 'customer satisfaction' it would be the difference in profits between 9 and 10 servers (43.1). We will use this optimal static policy as a policy comparison to see if the dynamic policies are worth while.



(a) Penalty distribution (8 servers)



(b) Penalty distribution (9 servers)



(c) Penalty distribution (10 servers)

Figure 6.1: Static Policy Penalty Distributions

6.1.2 Utilization Based Policy

Running Costs	Switching Costs	Penalties	Revenue	Profits
4966.75	70.90	0.0	23269.27	18231.62

This policy is heavily dependent on the bounds set on utilization. In the following simulations we will use values of 20% and 90% for lower and upper bounds respectively. From Figure 6.2 we can see that this policy resulted in almost all instances being switched on until the end of the run. As a result, no customers received a discount, hence maximum customer satisfacation was achieved and waiting time was at a minimum.

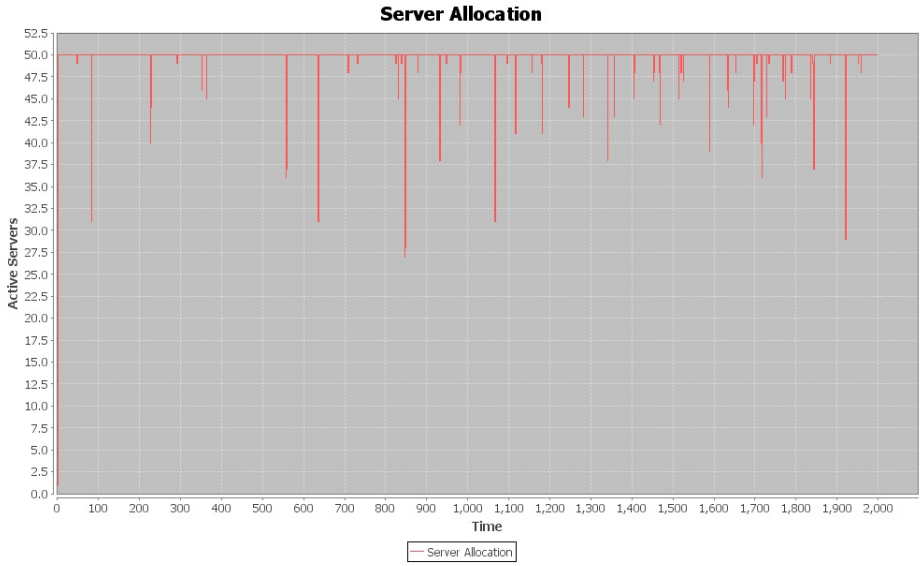


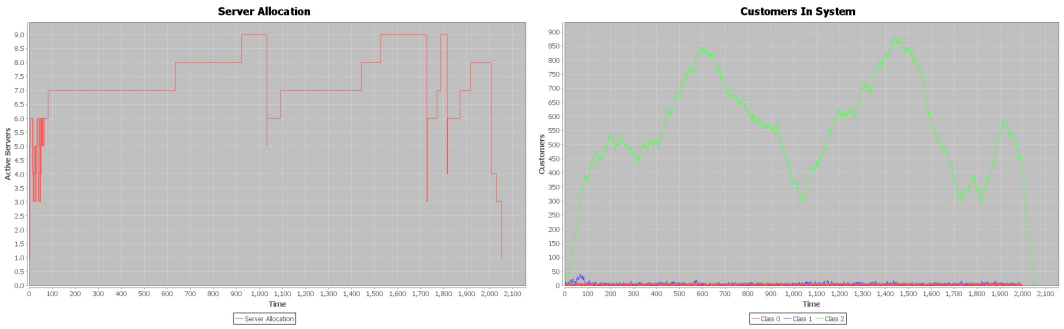
Figure 6.2: Utilization Policy Allocation

However, compared to the optimal static allocation, there is $\approx 20\%$ decrease in profits. This was caused by the increased running costs from all instances being switched on, with minor influence from switching costs. This was to be expected since this policy ignores all financial elements and aims at on-time job completion.

6.1.3 State Evaluation Policy

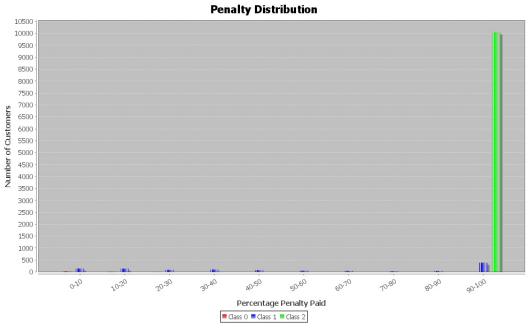
Running Costs	**Switching Costs**	**Penalties**	**Revenue**	**Profits**
750.34	12.00	2261.79	23269.27	20245.14

A significant improvement in profits in relation to the utilization policy can be seen. This is mainly attributed to the large difference of running costs. The allocation shown in Figure 6.3a shows a strong link to the optimal static policy, however, this dynamic policy is not an improvement on profits. In addition, a third of Class 2 customers received maximum discount, with a portion of the total discounts ranging over all classes and percentages, indicating high customer dissatisfaction.



(a) Allocation

(b) Customers In System



(c) Penalty Distribution

Figure 6.3: State Evaluation Policy Results

This policy was expected to perform better than the optimal static policy under a stable arrival stream but turns out to be far from optimal but cannot be dismissed as a strategy, given it has produced relatievly good results.

6.1.4 Predictive Planning Policy

Table 6.1: Results using window of size 20

Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	765.80	2361.73	277.33	23269.27	19864.41
2	765.66	2389.96	300.79	23269.27	19812.86
3	766.65	2405.57	262.47	23269.27	19834.58

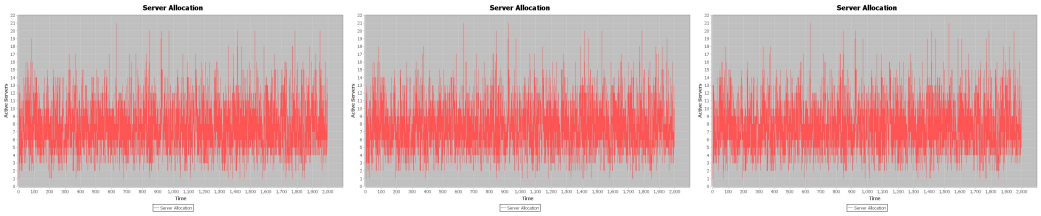
Table 6.2: Results using window of size 100

Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	767.04	2381.15	260.18	23269.27	19860.90
2	766.86	2376.55	231.32	23269.27	19894.54
3	768.14	2369.74	211.71	23269.27	19919.68

The predictive policies rely on windows for statistics to predict future arrival and service times. To investigate the impact of these windows on profits in a relatively stable system, we investigate 2 scenarios of small and large windows. Simulations whose results are not

shown in this report, have shown that a window of less than 20 will provide unreliable statistics and the policies will make wrong decisions. Therefore we continue to see if windows of greater than 20 have any effect on the decisions made.

From the tables above we see minor fluctuations in costs and profits, an indication that the predictive mechanisms are stable when using a simple arrival stream. Figures 6.4 and 6.5 show the allocations for each of the 2 windows sizes used for the simulations. We cannot see any differences between the 6 figures due to their nature, but rather notice high fluctuations of allocations between events. These fluctuations were unexpected when using a simple arrival stream and consequently result in very high switching costs.



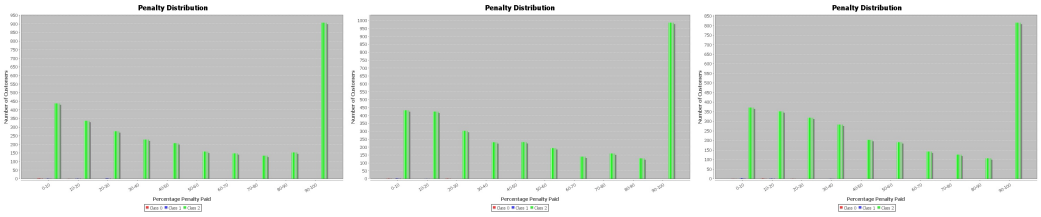
(a) Allocation For Run 1 (b) Allocation For Run 2 (c) Allocation For Run 3

Figure 6.4: Allocation For Predictive Policy (Windows Size = 20)

(a) Allocation For Run 1 (b) Allocation For Run 2 (c) Allocation For Run 3

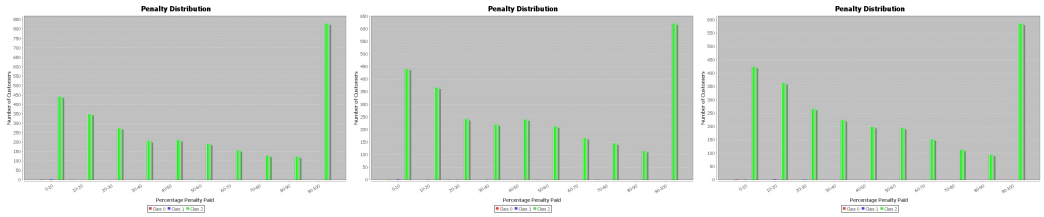
Figure 6.5: Allocation For Predictive Policy (Windows Size = 100)

As expected, the relatively high penalties were attributed to almost 2000 Class 2 customers receiving discounts, half of which received free service! The penalty distributions are shown below, and are almost identical in all 6 runs. We can safely assume that this type of service would be considered unacceptable by customers looking for a quick and reliable service.



(a) Penalty Distribution For Run 1 (b) Penalty Distribution For Run 2 (c) Penalty Distribution For Run 3

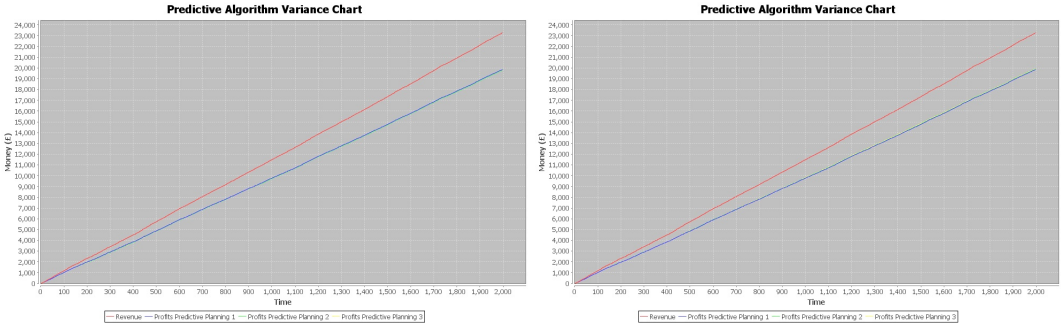
Figure 6.6: Penalty Distribution For Predictive Policy (Window Size 20)



(a) Penalty Distribution For Run 1 (b) Penalty Distribution For Run 2 (c) Penalty Distribution For Run 3

Figure 6.7: Penalty Distribution For Predictive Policy (Window Size 100)

As the financial measures implied, profits followed each other throughout the simulation. The variance is shown in Figures 6.8 below. There are minor differences in profits which cannot be seen in the graphs since they are too condensed, but we deem these differences negligible.



(a) Profit Variance For Window Size = 20 (b) Profit Variance For Window Size = 100

Figure 6.8: Profit Variance For Predictive Policies

Evidently, for stable arrival streams, window size was not an issue and resulted in approximately the same profits. Furthermore, the maximum likelihood estimator seems to generate approximately the same arrivals, since each set of simulations seem to follow each other very closely, implying that for stable systems it is quite reliable. Therefore, for the next set of simulations (the threshold policy), we will continue to only use a window size 100. The effects of the window size on a transient arrival stream will be investigated in simulations 2 and 3.

Issues which raise concern about this policy are the high switching costs (which will be taken care of by the next policy) and the perhaps unnecessarily high use of resources at points. Although this policy is a great improvement over the 2 policies which only use system information, there is room for improvement.

6.1.5 Predictive Planning Policy (with Threshold)

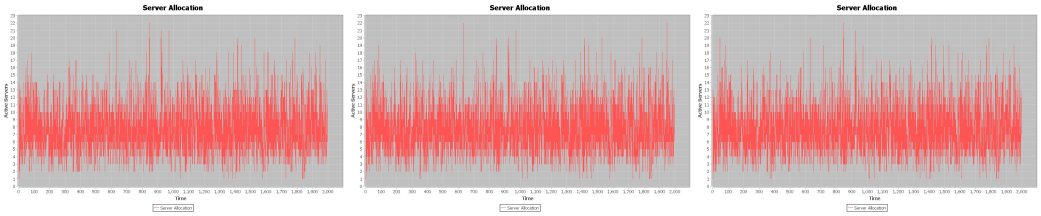
Table 6.3: Results using threshold of 0.2

Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	776.10	2140.66	146.76	23269.27	20205.75
2	776.45	2153.47	128.25	23269.27	20211.11
3	777.06	2144.36	139.63	23269.27	20208.22

Table 6.4: Results using threshold of 10

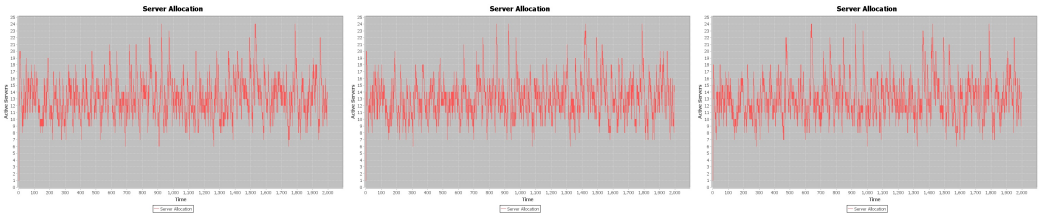
Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	1316.89	321.61	0.0	23371.05	21630.77
2	1308.17	324.51	0.0	23371.05	21636.59
3	1307.19	323.21	0.0	23371.05	21638.88

The main aim of this policy is to reduce switching costs and 'smooth out' the changes between allocations. This is very dependent on the threshold used so we continue and use 2 test cases; one is using 2 times the switching cost and the other using 100 times the switching cost. In the first case, all we are interested in is if the server being switched off has earned enough money to have a net profit of 0. The second case is to establish whether there is a need for a higher threshold to decrease switching costs even further.



(a) Allocation For Run 1 (b) Allocation For Run 2 (c) Allocation For Run 3

Figure 6.9: Allocation For Predictive Policy (Threshold 0.2)



(a) Allocation For Run 1 (b) Allocation For Run 2 (c) Allocation For Run 3

Figure 6.10: Allocation For Predictive Policy (Threshold 10)

The allocation graphs above, along with the 2 tables provide us with enough information to see that, using a threshold is indeed beneficial. In the first case, we have a

profit increase of ≈ 1.5% with switching costs remaining quite high. In the second case we have a drastic reduction in switching costs and an increase in running costs which result in a significant increase in profits by ≈ 10% compared to the policy without threshold. Another important advantage of increasing the threshold is that, due to a larger number of instances staying on for longer, no penalties are paid, guaranteeing a reliable service with minimal waiting time.

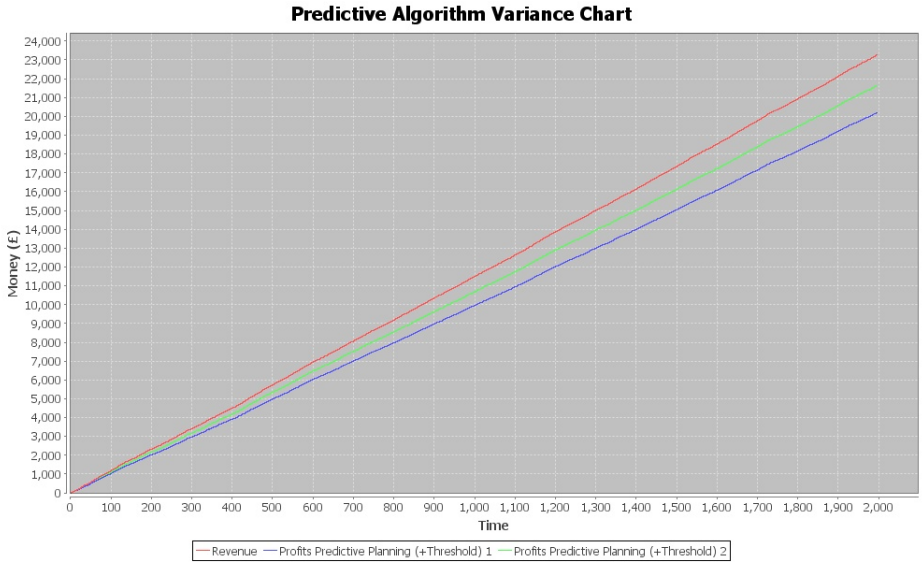


Figure 6.11: Profits Variance Using Different Thresholds

As with the policy without the threshold, the set of simulations with the same thresholds follow each other very closely, indicating that the policy will produce reliable results when dealing with stable streams. Figure 6.11 shows profits over time when using 2 times the switching cost (blue line) and 10 times the switching cost (green line). It is obvious that using 10 times the switching cost always yields better results. It would be worth further investigating what threshold would be considered optimal, however due to time constraints, we will use 10 times for our comparisons from this point on. Thresholds for transient arrivals will be investigated in simulations 2 and 3.

6.1.6 Predictive Planning Policy (Future Known)

Running Costs	Switching Costs	Penalties	Revenue	Profits
1225.90	1.30	0.18	23269.27	22041.89

Knowing the future arrivals of customers and their needs allows us to plan ahead. This policy has yielded maximum profits out of all the other policies due to its clear advantage of knowing the future. In comparison with the best policy so far, namely the predictive policy with 10 times the switching cost as a threshold, a ≈ 2% increase in profits can be seen. If we assume that this policy is the best we can do, a sub-optimal policy which yields 2% less profits can be deemed acceptable.

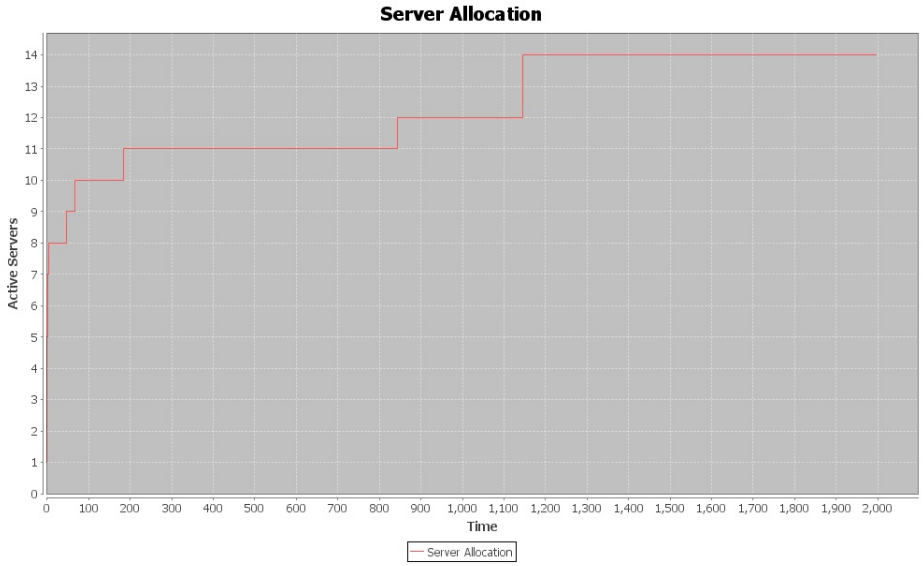


Figure 6.12: Allocation of Predictive Planning Policy (Future Known)

With the future known, costs are kept to a minimum, with a negligible number of penalties being paid. Our predictive policies can close the gap by improving the predictive mechanisms so that they make better decisions, although for stable arrival streams, they have proved to be quite good and reliable. An interest point to notice is the simplicity of the allocation graph, something we do not expect to see in the transient arrival streams to come.

6.1.7 Evaluation of Simulation 1

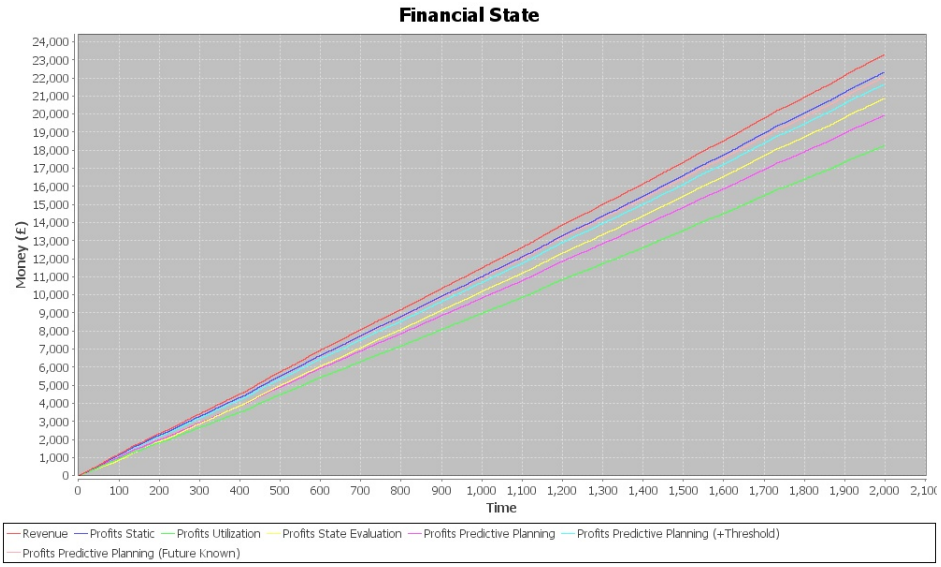


Figure 6.13: Simulation 1: Profits for all policies

Although a stable arrival stream is not the main testing ground for the policies, it is interesting to see their performance under this model. The best policy was the predictive

policy which uses a threshold. In fact, profits remain relatively close to the predictive policy which knows about future arrivals, throughout the run, indicating a reliable policy which yields excellent results under this model. A surprise was the performance of the policies which used on local system information, which were expected to perform better, but stayed within a logical range none-the-less.

An interesting point to notice is that none of the policies earned more profits than the optimal static allocation, which could be attributed to the chosen values for running and switching costs, however, a closer look can show that in a realistic setting, one would prefer the dynamic policies. The observation that the penalties paid are significantly higher in the optimal static allocation than in the predictive policies is very important. This suggests that using the predictive policy using a threshold would keep customer satisfaction high, only at a small cost, since profits did not have a big difference between them. This trade-off between profits and customer satisfaction, is expected to be seen in the next few simulations as well and must be taken into account. Furthermore, getting the right static allocation depends on the arrival stream which, as seen through the static policy simulations, if chosen wrongly, could result in huge differences in profits and customer satisfaction.

6.2 Simulation 2

The second simulation will focus on transient arrival streams of relatively low arrival rates and minor fluctuations in rate parameters. As discussed in 'The Model' section of the report, we will use MMPP to model arrivals of Class 0 customers and SMPP to model Class 1 and Class 2 customer arrivals. We further investigate the effects on profits of different window sizes and threshold levels when we have transient arrival stream. The parameters which will be used are:

- **Class 0**

$$\bar{\Lambda} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 10 \end{pmatrix} \quad \bar{Q} = \begin{pmatrix} 0.998 & 0.001 & 0.001 \\ 0.001 & 0.998 & 0.001 \\ 0.001 & 0.001 & 0.998 \end{pmatrix}$$

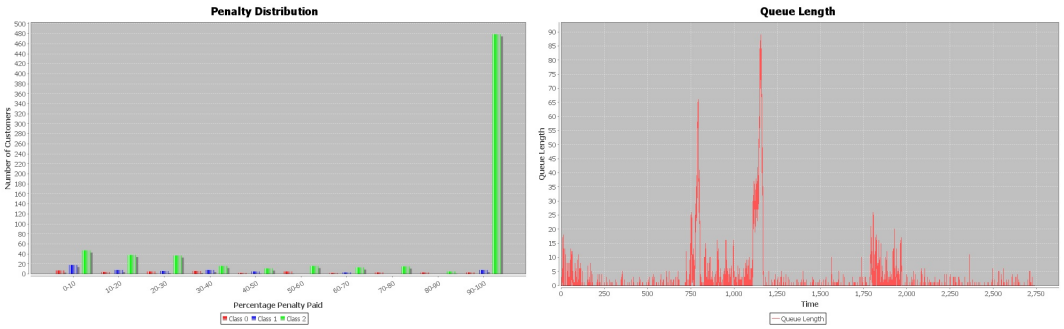
- **Class 1** $\lambda_{min} = 1$ $\lambda_{max} = 5$

- **Class 2** $\lambda_{min} = 1$ $\lambda_{max} = 10$

6.2.1 Static Policy

Allocation	Running Costs	Switching Costs	Penalties	Revenue	Profits
8	1096.63	0.0	412.54	18301.08	16791.91
9	1233.70	0.0	134.99	18301.08	16932.38
10	1370.78	0.0	30.99	18301.08	16899.31

In this simulation, the optimal static policy would be to have 9 instances switched on. This however comes at a cost of approximately 500 customers receiving full discounts, an indication that a significant percentage of customers will not be satisfied with the service. As shown in Figure 6.14a, all classes were affected by this poor service, with the lowest priority class taking the most of the hit. Furthermore, the queue length fluctuates in size significantly, probably due to the transient arrivals of the simulation. This is a clear indication a dynamic system is favourable.



(a) Penalty Distribution (b) Queue Length

Figure 6.14: Static Policy Results

6.2.2 Utilization Based Policy

Running Costs	Switching Costs	Penalties	Revenue	Profits
6668.14	42.10	0.04	18301.08	11590.80

The utilization policy profits are almost 5000 units less than the optimal static policy. This is attributed to the massive running costs of switching on almost all instances for the whole run (shown in Figure 6.15 below). The advantage of this is that only 1 customer was offered a small discount and waiting time was kept to a minimum. However, the high customer satisfaction comes at an unnecessarily high cost and there is room for improvement. For a transient system, this policy has proved to be far from optimal, even if the policy parameters are changed.

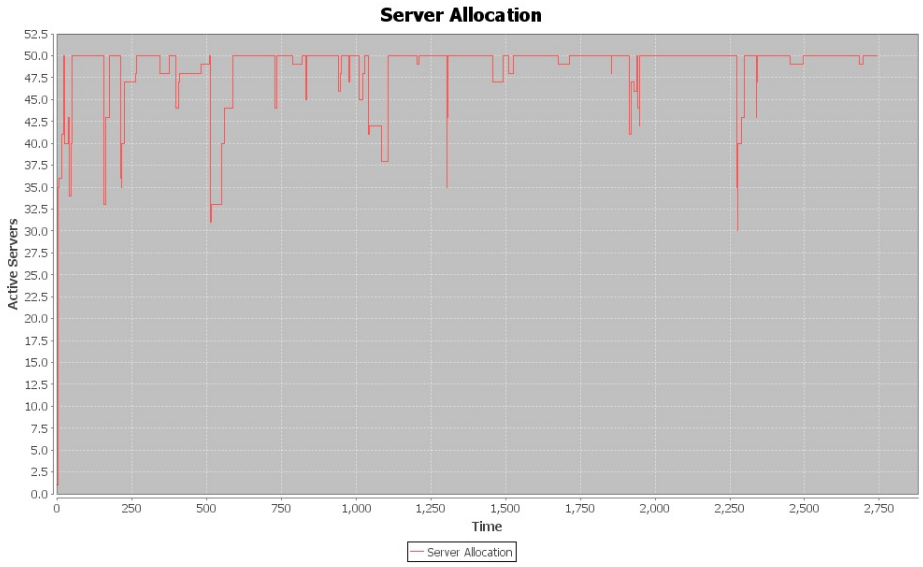
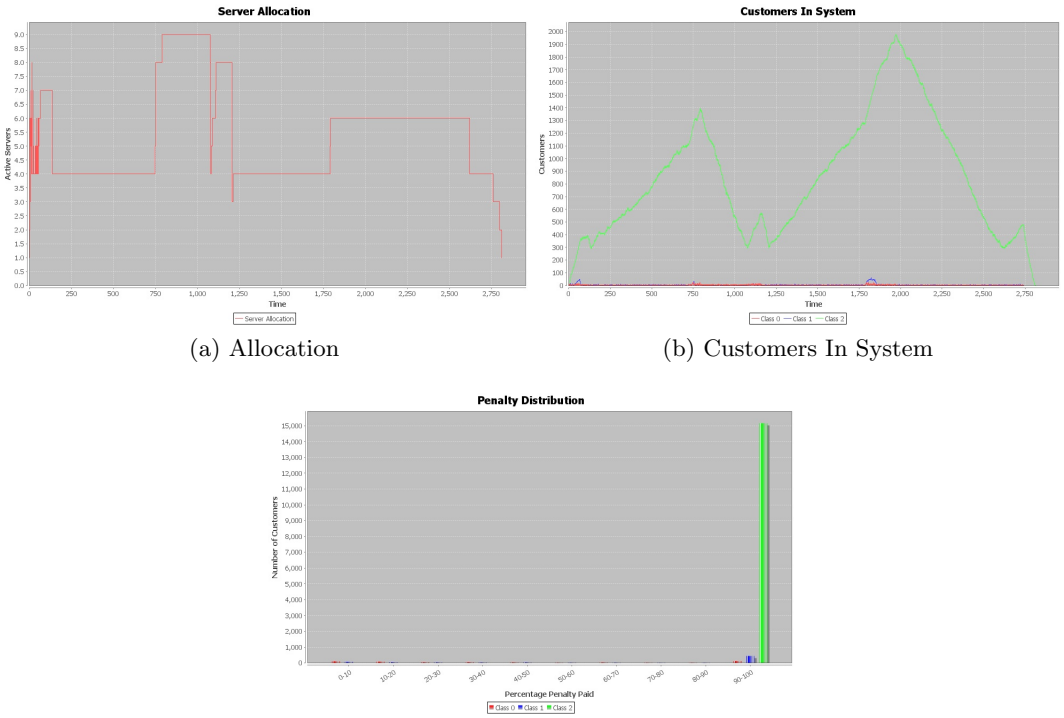


Figure 6.15: Allocation of Utilization Policy

6.2.3 State Evaluation Policy

Running Costs	Switching Costs	Penalties	Revenue	Profits
756.02	10.60	3488.66	18301.08	14045.79

As with the stable arrival stream, a strong link to the optimal static policy can be seen, however, under a transient system, this policy deteriorates. Almost half the customers of the simulation got off with an almost free service and profits are considerably low.



(a) Allocation

(b) Customers In System

(c) Penalty Distribution

Figure 6.16: State Evaluation Policy Results

As expected, this policy did not perform well under a transient arrival stream, resulting in huge penalties being paid and customer dissatisfaction being high. We expect that in the next simulations, results from this policy will deteriorate.

6.2.4 Predictive Planning Policy

Table 6.5: Results using window of size 20

Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	778.33	2153.37	26.70	18301.08	15342.67
2	778.75	2148.57	26.92	18301.08	15346.83
3	779.05	2146.36	24.90	18301.08	15350.76

Table 6.6: Results using window of size 100

Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	777.98	2162.98	29.33	18301.08	15330.79
2	778.14	2153.17	30.90	18301.08	15338.86
3	777.71	2140.96	27.24	18301.08	15355.17

From the above tables above we can see that, as with stable arrival rates, the set of runs have approximately the same results. In other words, the policy reliability is kept even

when using transient arrival rates. Approximately 300 customers in total received discounts, with the significant majority receiving less than 50%. As expected, this affected mainly Class 2 customers. As indicated by the allocation graphs below, rapid switching is still a problem, resulting in unacceptably high switching costs. This is further proof that the threshold is a necessary addition.

(a) Allocation For Run 1 (b) Allocation For Run 2 (c) Allocation For Run 3

Figure 6.17: Allocation For Predictive Policy (Windows Size = 20)

(a) Allocation For Run 1 (b) Allocation For Run 2 (c) Allocation For Run 3

Figure 6.18: Allocation For Predictive Policy (Windows Size = 100)

Another important observation comes from Figure 6.19 below. When looking closely at the Class 0 customers in the system, we can distinguish 3 peaks. These peaks can also be seen in all 6 allocation figures above, indicating that the allocation decision is highly dependent on the Class 0 customers. This could be the grounds for future improvement of these policies.

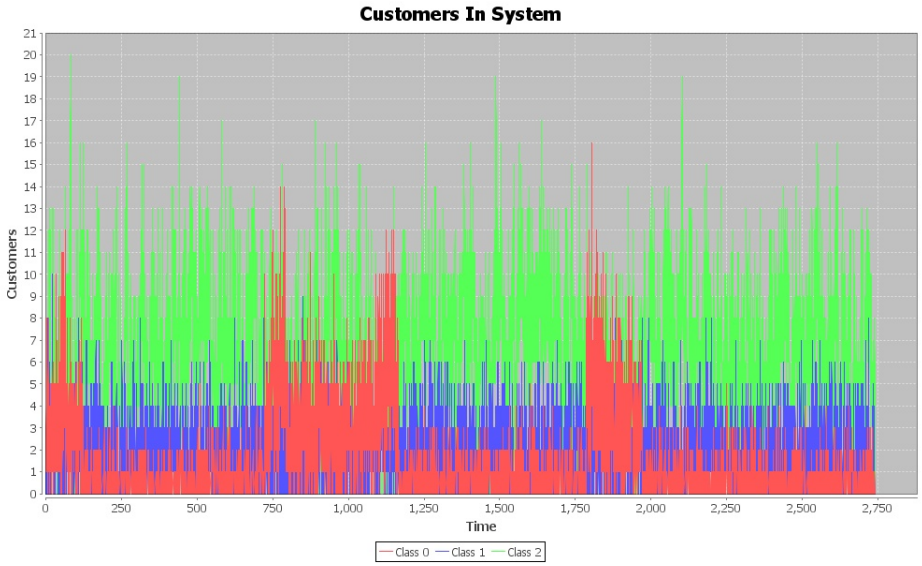
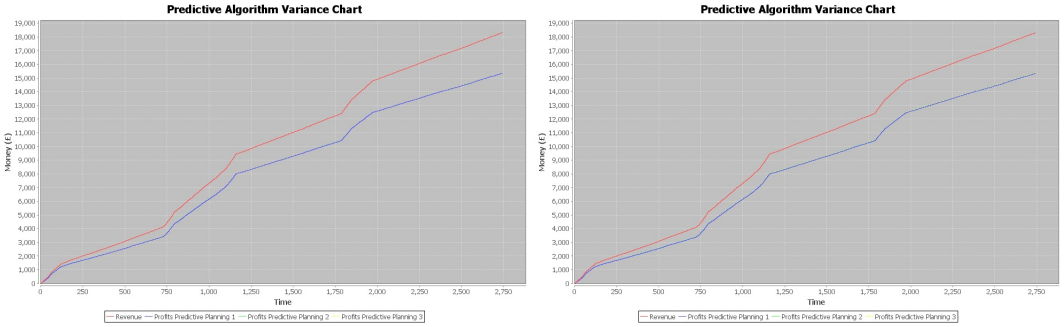


Figure 6.19: Customers in the System

As a conclusion to the issue of window sizes, the above figures and tables, in combination with the variance of profits shown in Figure 6.20, show that a window size greater than 20 has negligible effects on profits, even for transient arrival schemes. We conclude that a window of size 100 provides sufficiently good results, regardless of the nature of the arrival rates and will therefore be used from here onwards.



(a) Profits of 3 Runs (Window Size 20) (b) Profits of 3 Runs (Window Size 100)

Figure 6.20: Profits Using Different Sized Windows

6.2.5 Predictive Planning Policy (with Threshold)

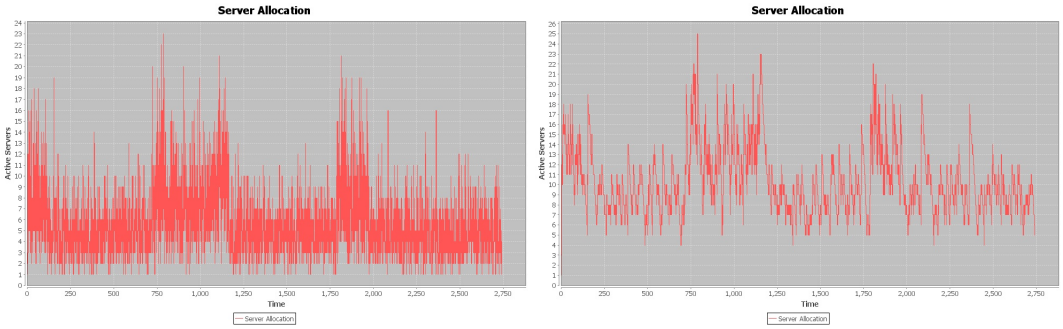
Table 6.7: Results using threshold of 0.2

Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	792.40	1870.12	13.37	18301.08	15625.19
2	792.46	1882.12	13.17	18301.08	15613.32

Table 6.8: Results using threshold of 10

Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	1427.67	251.81	0.0	18301.08	16621.60
2	1427.33	253.53	0.0	18301.08	16620.22

As expected, using thresholds provides better results with respect to our goal. It has decreased the massive switching costs and eliminated penalties, and even though running costs were doubled, an almost 8% increase in profits can be seen. The effect of decreasing switching costs can be seen in the figure below, where the lines are not so close any more due to the reduction of switching. In addition, profits are very close to the static optimal policy with much better customer satisfaction, making this the preferable policy.



(a) Allocation using threshold of 0.2 (b) Allocation using threshold of 10

Figure 6.21: Allocation Using Different Thresholds

We can conclude that using thresholds is in fact to our advantage, regardless of the type of arrival stream. Figure 6.22 confirms that using 10 times the switching cost for a threshold (blue line), yields more profits throughout the run compared to 2 times switching cost (green line). In fact, as time goes by, the difference between them is increases, indicating that the threshold must be chosen carefully. We will confirm our theory in simulation 3 and then follow on to see the effects of using larger thresholds with transient arrival streams.

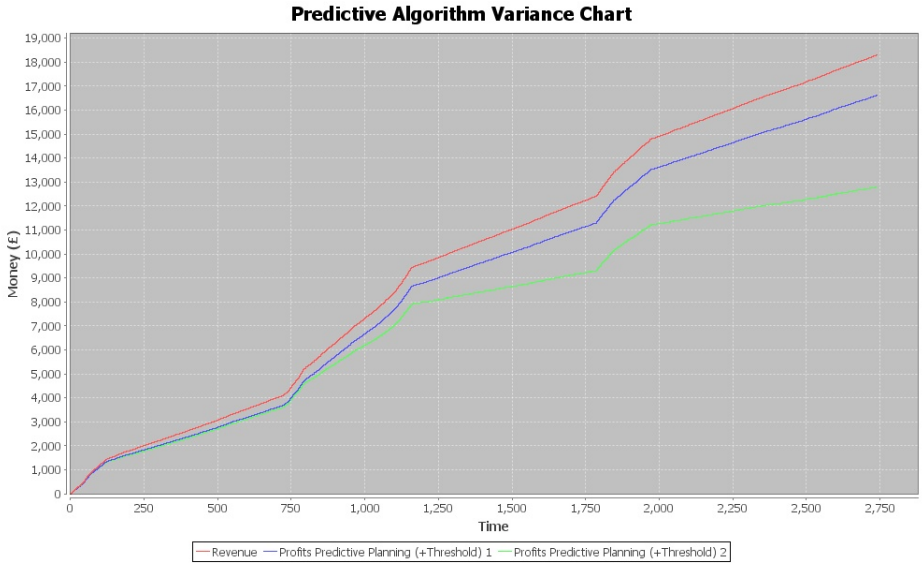
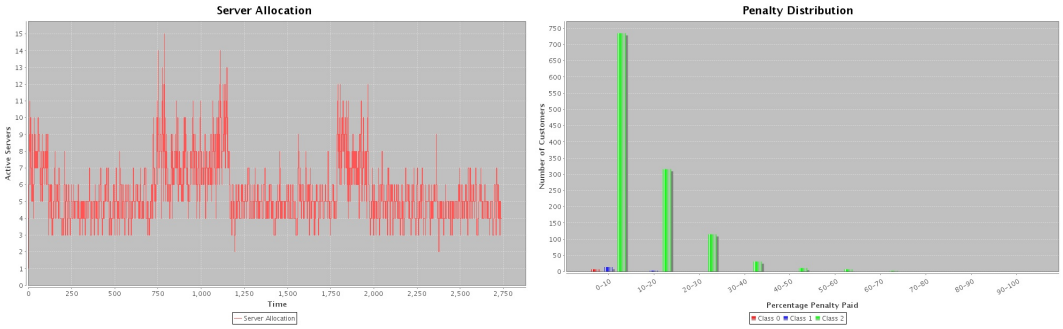


Figure 6.22: Profits Using Different Thresholds

6.2.6 Predictive Planning Policy (Future Known)

Running Costs	Switching Costs	Penalties	Revenue	Profits
756.18	168.10	22.06	18301.08	17348.73

Even when a using transient arrival stream we can see the potential of the predictive policies. This policy surpasses the profits of all policies by at least 2%, a clear indication that prediction is key for good results. These results indicate that, if the prediction mechanism of the predictive policies is improved, results could improve drastically.



(a) Allocation (b) Penalty Distribution

Figure 6.23: Predictive Planning Policy (Future Known) Results

An interesting observation is that these results are achieved with a significant portion of customers receiving discounts. Since the discounts are in the 0-10% range, we could assume that in a realistic setting this might be acceptable, however, the policy could be tweaked to reduce this cost and increase customer satisfaction.

6.2.7 Evaluation of Simulation 2

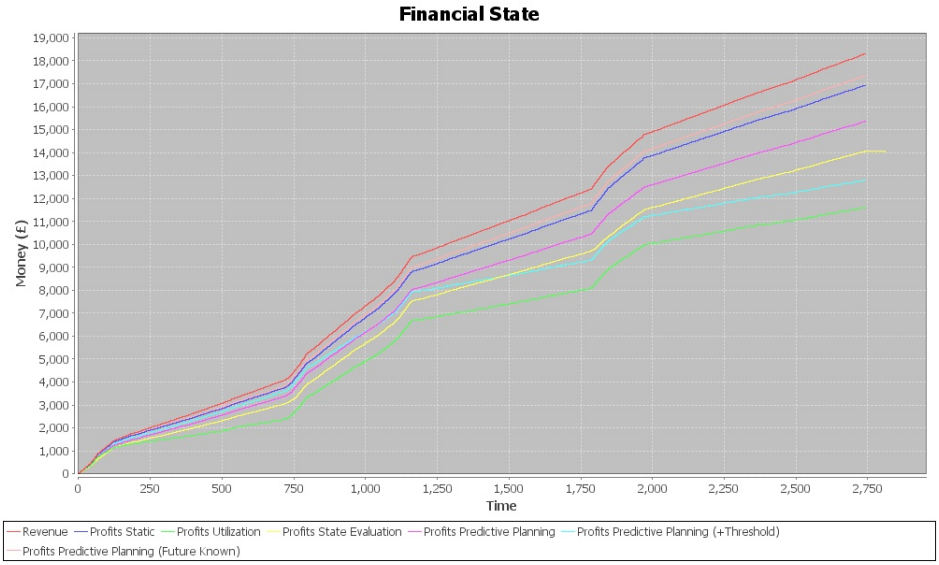


Figure 6.24: Simulation 2: Profits for all policies

Surprisingly, the optimal static policy yielded the most profits, with the predictive policy which uses a threshold closely behind. One observation from Figure 6.24 that stands out is that all policies take approximately the same amount of time to service the 30 000 customers except the state evaluation policy which takes slightly longer. This could be attributed to its use of a small number of servers throughout the run, resulting in much less throughput.

The fact that the predictive policy (future known) performed so well under this arrival stream is an indication that the other predictive policies could perform better if the predictive mechanism was improved. None the less, profits stayed within a reasonable limit and customer satisfaction was high. The policies which use only local information did not perform as well, with the utilization policy over-using resources and the state evaluation policy under-using resources (and having a massive penalty payout). Given was the simplest transient arrival stream that will used, we expect that the above effects will multiply in the next few simulations.

6.3 Simulation 3

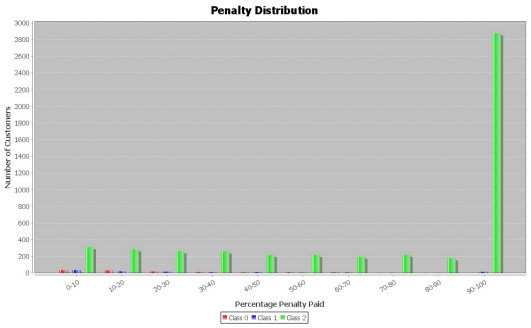
The third simulation will use the same parameters as the second simulation, except from the infinitesimal generator which we will change to allow for more rapid changes between the states by assigning equal probabilities between all states:

- **Class 0**
$$\bar{\Lambda} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 10 \end{pmatrix} \quad \bar{Q} = \begin{pmatrix} 0.34 & 0.33 & 0.33 \\ 0.33 & 0.34 & 0.33 \\ 0.33 & 0.33 & 0.34 \end{pmatrix}$$
- **Class 1** $\lambda_{min} = 1$ $\lambda_{max} = 5$
- **Class 2** $\lambda_{min} = 1$ $\lambda_{max} = 10$

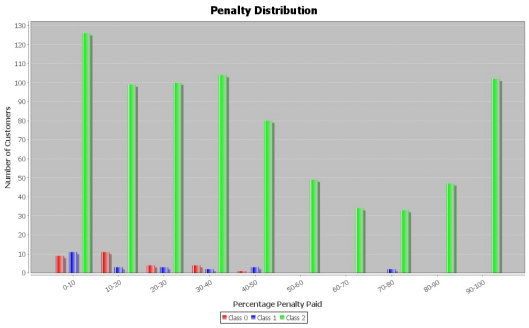
6.3.1 Static Policy

Allocation	Running Costs	Switching Costs	Penalties	Revenue	Profits
6	834.13	0.0	753.17	17990.24	16402.94
7	972.84	0.0	73.00	17990.24	16944.40
8	1111.82	0.0	8.21	17990.24	16870.21

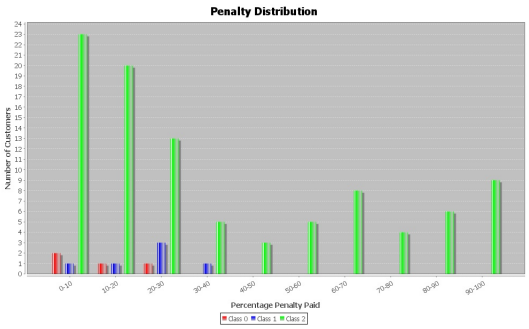
The optimal static policy for the 3^{rd} simulation is 7 servers. As the penalty distribution, depicted by Figure 6.25b shows, a large number of customers received discounts, indicating an unreliable service.



(a) Penalty Distribution (6 servers)



(b) Penalty Distribution (7 servers)



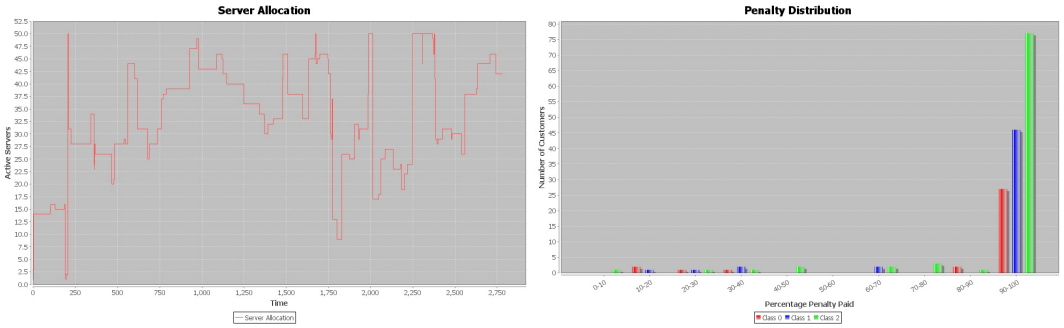
(c) Penalty Distribution (8 servers)

Figure 6.25: Penalty Distributions For Different Static Allocations

6.3.2 Utilization Based Policy

Running Costs	Switching Costs	Penalties	Revenue	Profits
4630.12	54.50	94.28	17990.24	13211.35

The utilization policy profits are again much lower than the optimal static policy. The allocation for this run is erradic and contrary to the 2^{nd} simulation, we incurred large penalties of 90%-100% from all classes. This proves that this policy deteriorates by increasing the state fluctuations of the arrival streams.



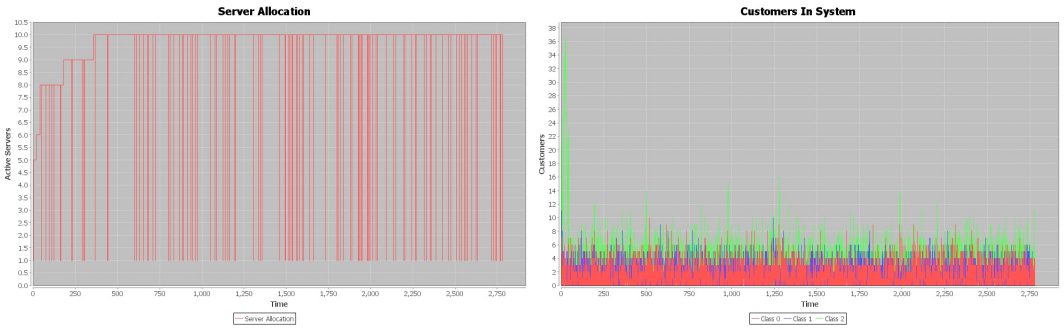
(a) Allocation Distribution (b) Penalty Distribution

Figure 6.26: Utilization Based Policy Results

6.3.3 State Evaluation Policy

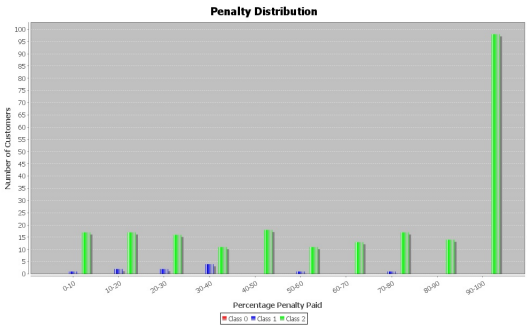
Running Costs	Switching Costs	Penalties	Revenue	Profits
1070.28	239.00	30.93	17990.24	16650.07

To our surprise, results did not deteriorate as expected. In fact, the results are very close to the optimal static policy and are a significant improvement over the utilization policy results. The main issue in this case are the switching costs which, as can be seen from Figure 6.16a, are caused by rapid fluctuations in allocation, most probably attributed to the rapid changes in arrival rates of Class 0 customers.



(a) Allocation

(b) Customers In System



(c) Penalty Distribution

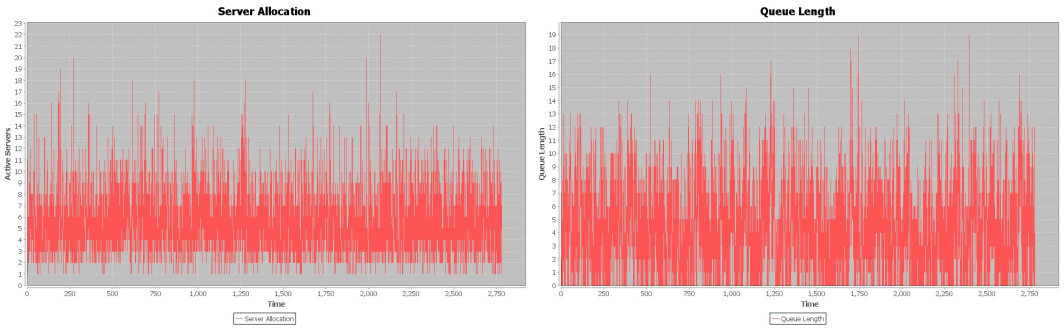
Figure 6.27: State Evaluation Policy Results

The penalty distribution above does show quite a few customers getting a discount, but is an improvement over the second simulation. Results show that the rapid changes in the arrival rates ease out the penalty distribution and decrease the penalties paid. It turns out this policy performs better for a system with faster changing states. This theory will be examined further in simulations 4 and 5.

6.3.4 Predictive Planning Policy

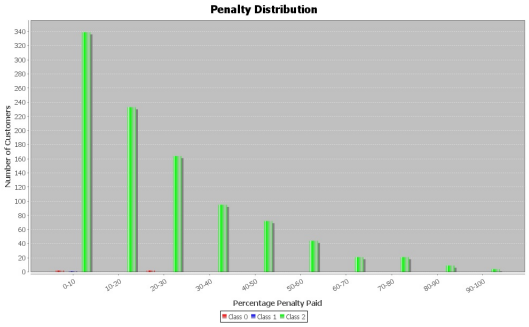
Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	776.33	2197.00	35.00	17990.24	14981.90
2	776.50	2200.41	33.87	17990.24	14979.48
3	776.89	2196.40	34.91	17990.24	14982.03

Increasing the rate of transition of the MMPP seems to have no effect on how the predictive policy works. The decisions fluctuate just as much as when the rates were low (Figure 6.28a) which is, as before, the main source of cost. As a result, the queue length changes rapidly as well (Figure 6.28b) resulting to fluctuating waiting times for the customers which, in turn, affect penalties. The penalty distribution depicted in Figure 6.28c shows the unacceptably high number of customers who received a discount. It is worth noting that the distribution is thankfully scewed towards the low percentages.



(a) Allocation

(b) Queue Length



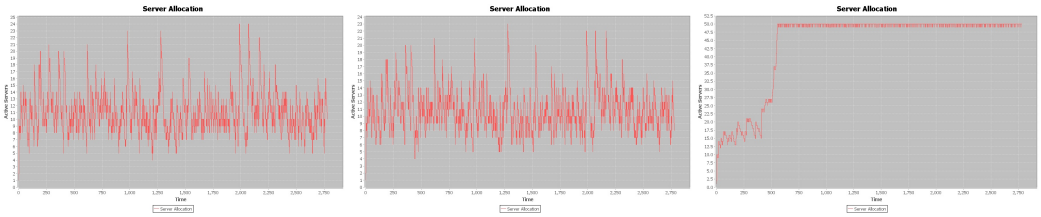
(c) Penalty Distribution

Figure 6.28: Predictive Planning Results

6.3.5 Predictive Planning Policy (with Threshold)

Run	Running Costs	Switching Costs	Penalties	Revenue	Profits
1	1540.44	247.00	0.0	17990.24	16202.80
2	1510.88	245.90	0.0	17990.24	16233.46
3	6094.82	53.10	0.0	17990.24	11842.33

When looking at the table above, the 3^{rd} run stands out due to its different values compared to the runs due to a flaw. The rapidly changing states of the MMPP processes rarely results in the prediction mechanisms thinking an increasing amount of customers are arriving and so switching on more than necessary instances. These instances cannot be switched off as easily and so remain high. This phenomenon can be seen in Figure 6.29c. The wrong decisions consequently increase running costs 6 fold and decrease profits by $\approx 25\%$! As already mentioned, the prediction mechanism has be fine tuned to prevent large variations between runs, however when using rapidly changing transient arrival streams there is an off chance this might occurr. This is considered only as a flaw in the prediction mechanism, not the policy strategy.



(a) Allocation for Run 1 (b) Allocation for Run 2 (c) Allocation for Run 3

Figure 6.29: Allocation of the Runs

This flaw affects profits throughout the run, always resulting in less profits being made (Figure 6.30). As time progresses, the effects of the wrong decision seem to increase, indicating the perhaps some mechanism should be put in place to detect this flaw and attempt to fix it accordingly.

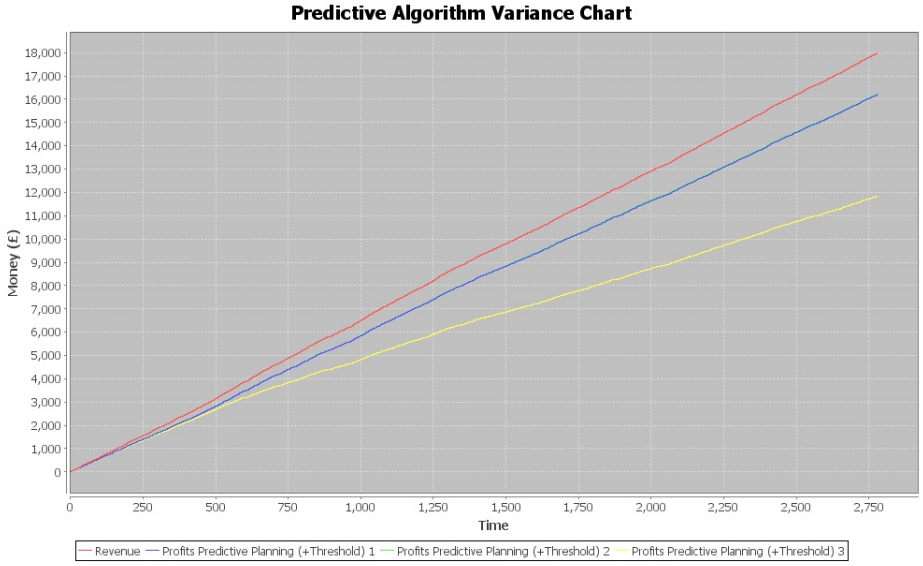
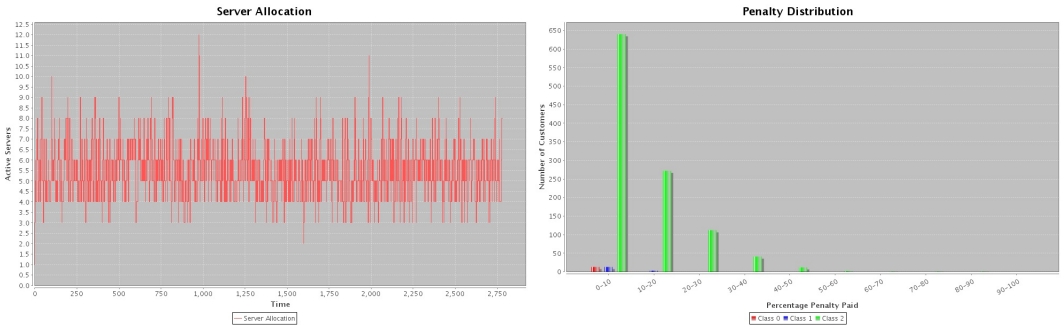


Figure 6.30: Profits of the Different Runs

6.3.6 Predictive Planning Policy (Future Known)

Running Costs	Switching Costs	Penalties	Revenue	Profits
762.14	175.10	20.00	17990.24	17033.00

As with the previous simulation, this policy produced the highest profits from all other policies indicating the advantages of a predictive policy. The change in probabilities has had as a consequence a minor increase in switching costs and penalties. As shown in Figure 6.31b, the penalty distribution is very similar to the 2^{nd} simulation, but has affected more Class 0 and Class 1 customers. Again, in a realistic setting, the penalties paid could be deemed acceptable or the policy could be tweaked to improve customer satisfaction at a cost.



(a) Allocation (b) Penalty Distribution

Figure 6.31: Predictive Planning Policy (Future Known) Results

6.3.7 Evaluation of Simulation 3

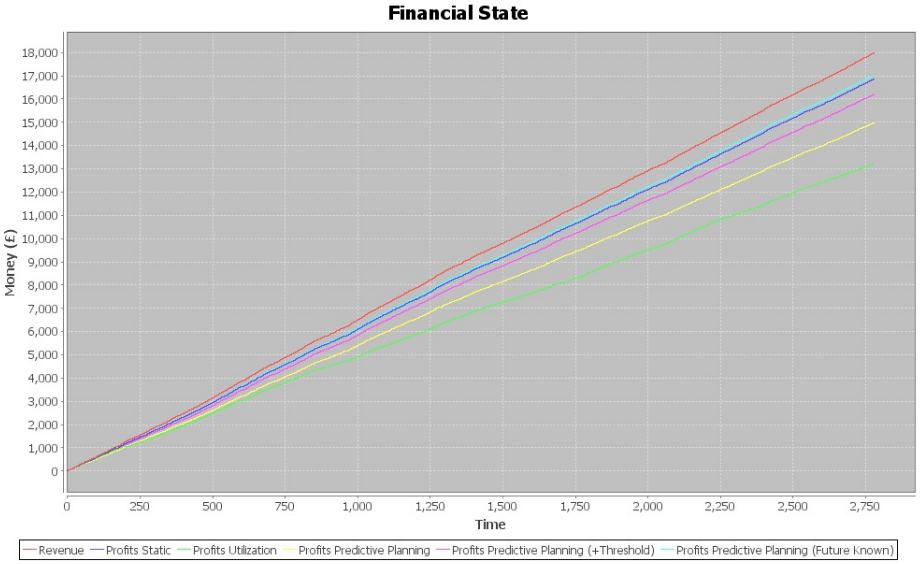


Figure 6.32: Simulation 3: Profits for all policies

From the results of this simulation, it seems that rapid changes in arrival rates reduce the negative effects we expected to see from the transient nature of the arrival stream. In fact, the state evaluation policy performed better than the predictive planning policy, however, the utilization based policy's performance did deteriorate as expected.

The predictive planning policy which uses a threshold does not seem to have been affected as much by increasing the probability of transitions and performed very well, yielding high profits and no penalty payout. Unfortunately, one out of the three runs revealed a flaw or bug with the predictive mechanism where the policy over-estimates future arrivals and increases the allocation in light of this information, only to get stuck in a high allocation and increase running costs significantly.

Given the predictive policy which uses future arrivals outperformed all other policies, it is clear that the predictive mechanism of the predictive policies falls short of expectations and can be improved to yield better results.

6.4 Simulation 4

The next step is to increase the arrival rates to strain the system even more. The maximum combined arrival rates will exceed the maximum number of servers to see how the policies cope with increased traffic. To start off we will use the same infinitesimal generator as simulation 2 to keep fluctuations between the states to a minimum. We move away from changing window size, as previous simulations have shown that using a window size of 100 provides sufficiently good results, and instead focus more on the effects of increasing threshold levels passed 10 times switching costs. We define the simulation parameters as follows:

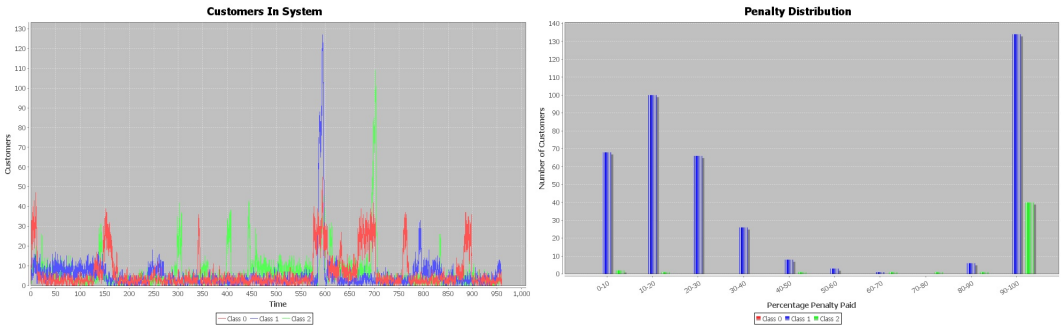
- **Class 0, 1, 2**

$$\bar{\Lambda} = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 50 \end{pmatrix} \quad \bar{Q} = \begin{pmatrix} 0.998 & 0.001 & 0.001 \\ 0.001 & 0.998 & 0.001 \\ 0.001 & 0.001 & 0.998 \end{pmatrix}$$

6.4.1 Static Policy

Allocation	Running Costs	Switching Costs	Penalties	Revenue	Profits
45	2160.18	0.0	194.84	24553.61	22198.59
46	2208.17	0.0	142.34	24553.61	22203.10
47	2256.18	0.0	98.93	24553.61	22198.49
50	2400.19	0.0	32.21	24553.61	22121.22

The optimal static allocation is 46 servers. As with all previous simulations, this comes at a high penalty rate, but as shown by Figure 6.33b, affects more Class 1 than any other class of customers. This is an indication that static policies could result in 'worst case scenarios' and end up paying penalties to higher paying customers and so lose more money. In this scenario, the penalties could be attributed to the spike in Class 1 customers at time 600, probably due to the arrival rate changing to 50. At the same same, there is a spike in arrivals of Class 0 customers which pre-empted Class 1 customers and so had to wait longer and so received a large number of discounts. This is an indication that the system is very dependent on higher priority customers and there is a clear need for dynamic systems to be able to react to these changes. Something the reader should keep in mind when reading on, is that even with full resource use, we end up paying penalties, as indicated by the above table.



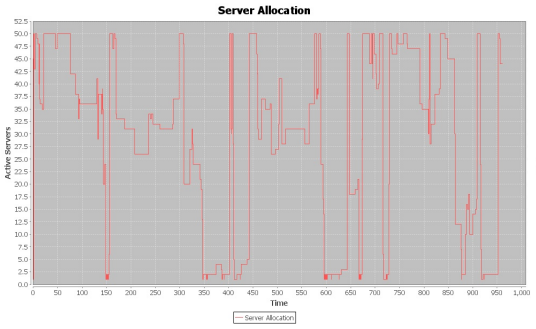
(a) Customers In System (46 Servers Allocated) (b) Penalty Distribution (46 Servers Allocated)

Figure 6.33: Static Allocation Results

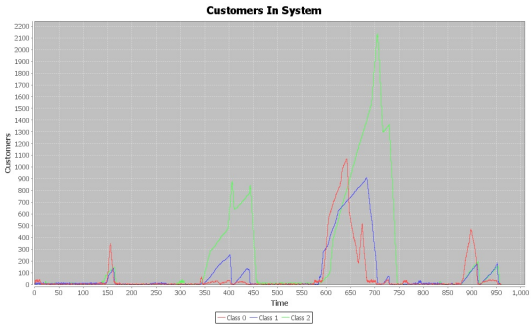
6.4.2 Utilization Based Policy

Running Costs	Switching Costs	Penalties	Revenue	Profits
1406.23	154.70	8796.46	24553.61	14196.18

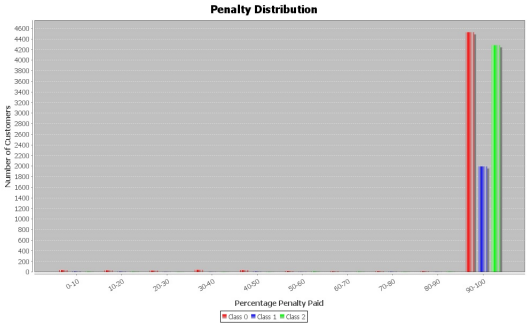
The large changes between the arrival rates make utilization fluctuate uncontrollably. Consequently, allocation fluctuates accordingly as shown in Figure 6.34a and incurs large switching costs. Furthermore, penalties paid are extremely high, ranging from all 3 classes, mainly in the region of 90%-100%. Figure 6.34b shows several large spikes of customers in the system which probably were the reason for all the delays. This policy seems to deteriorate with increasing fluctuations in arrival rates and cannot be used in transient systems.



(a) Allocation



(b) Customers In System



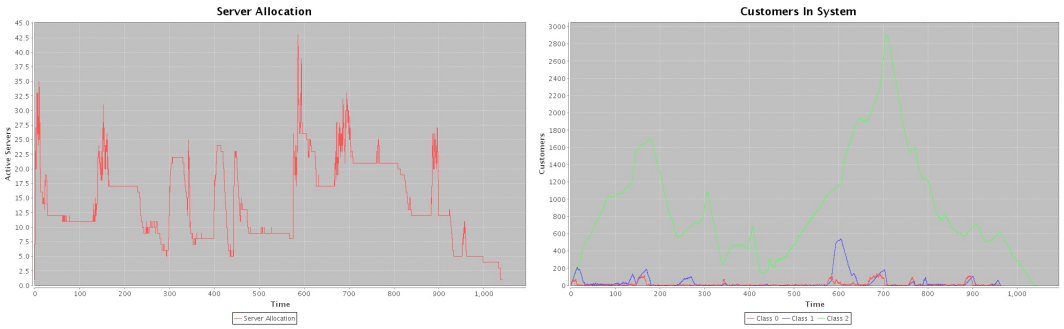
(c) Penalty Distribution

Figure 6.34: Utilization Policy Results

6.4.3 State Evaluation Policy

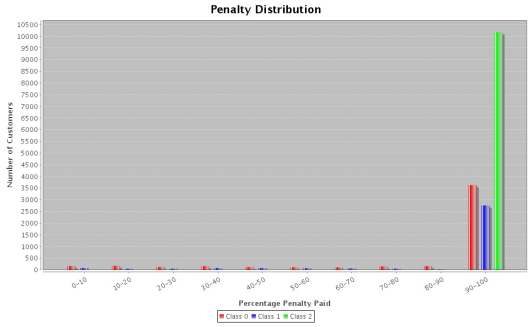
Running Costs	Switching Costs	Penalties	Revenue	Profits
751.23	230.20	9985.76	24553.61	13586.45

As already seen from simulation 3, the state evaluation policy cannot deal with transient arrival streams with low probability state transitions. This simulation confirms this theory and establishes that the larger the difference between arrival rates, the worse this policy will perform. From the allocation graph below, we can see the rapid changes in allocations, contributing to switching costs, with the main source of cost being the penalties.



(a) Allocation

(b) Customers In System



(c) Penalty Distribution

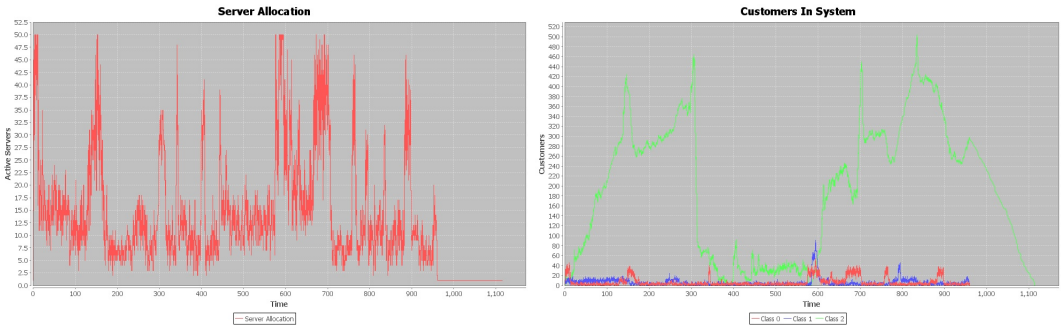
Figure 6.35: State Evaluation Policy Results

Figure 6.35c clearly depicts customer dissatisfaction for all classes and proves that this policy can be dismissed for transient arrival schemes due to the low profit return. The final simulation will be able to confirm if this policy can perform better under rapid state transitions.

6.4.4 Predictive Planning Policy

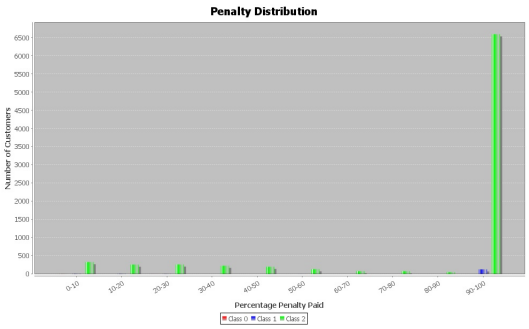
Running Costs	Switching Costs	Penalties	Revenue	Profits
752.64	2229.42	1363.59	24553.61	20207.96

The usual flaws of this policy are at the same level regardless of arrival rates, indicating a flaw with the strategy of the policy rather its inner workings. Regions of high and low allocations can be seen in Figure 6.36a, probably relating to the customer arrival rates fluctuating between high and low states. This policy cannot handle the changes between high and low rates very well, given that almost 7000 Class 2 customers received a discount of 90%-100%. The discrimination of Class 2 customers can be seen from the Figures 6.36b and 6.36c below which, in turn, resulting in a high penalty payout.



(a) Allocation

(b) Customers In System



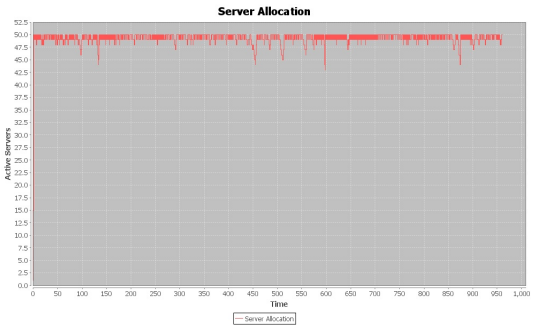
(c) Penalty Distribution

Figure 6.36: Predictive Planning Policy Results

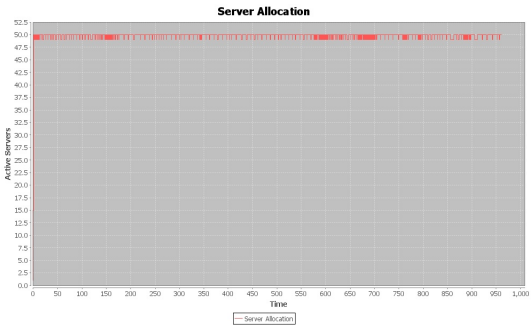
6.4.5 Predictive Planning Policy (with Threshold)

Threshold	Running Costs	Switching Costs	Penalties	Revenue	Profits
20	2375.15	166.60	36.70	24553.61	21975.17
80	2393.56	53.50	34.16	24553.61	22072.39
180	2396.82	27.90	31.50	24553.61	22097.39

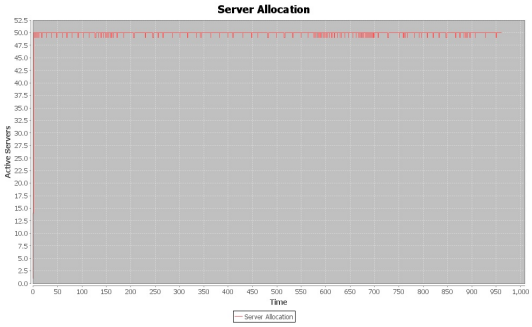
Interestingly, increasing the threshold increases profits. This can be attributed to a reduction in switching costs and penalties. This effect however is dependent on the values of the simulation parameters such as running and switching costs, which in our case result in profits. From the figures below, we can see a full use of resources and by increasing the threshold we get a thinning of the lines between events due to less switching.



(a) Allocation (Threshold of 20)



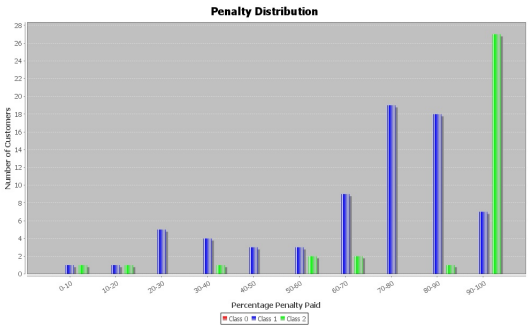
(b) Allocation (Threshold of 80)



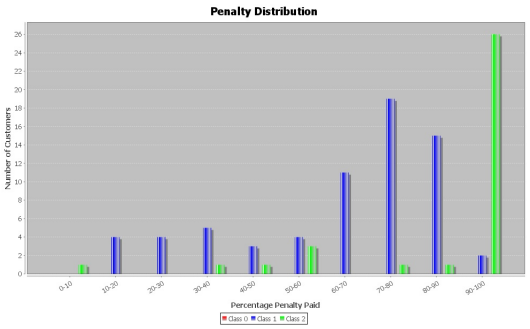
(c) Allocation (Threshold of 180)

Figure 6.37: Allocation using Different Thresholds

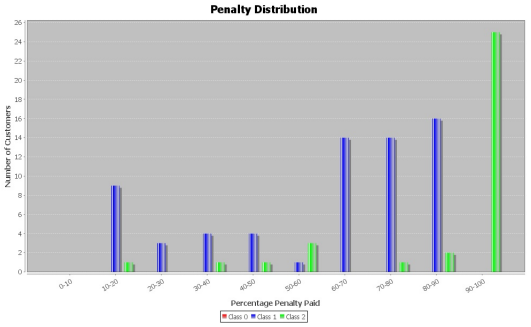
Furthermore, we have an acceptable range of penalties paid in all cases. Compared to the optimal static policy, this policy is only 100 units short in profits but with approximately 300 customers less in penalties, indicating more customer satisfaction under this policy.



(a) Penalty Distribution (Threshold of 20)



(b) Penalty Distribution (Threshold of 80)



(c) Penalty Distribution (Threshold of 180)

Figure 6.38: Penalty Distribution using Different Thresholds

Figure 6.39 below shows the variation in profits throughout the run using different thresholds (starting with 20 and going up to 180). The profits follow each other very closely, with the 180 threshold line being slightly above the rest. It is obvious that increasing the threshold is beneficial, even though only by a small margin, but more investigation is necessary to determine which levels are necessary to guarantee maximum profits.

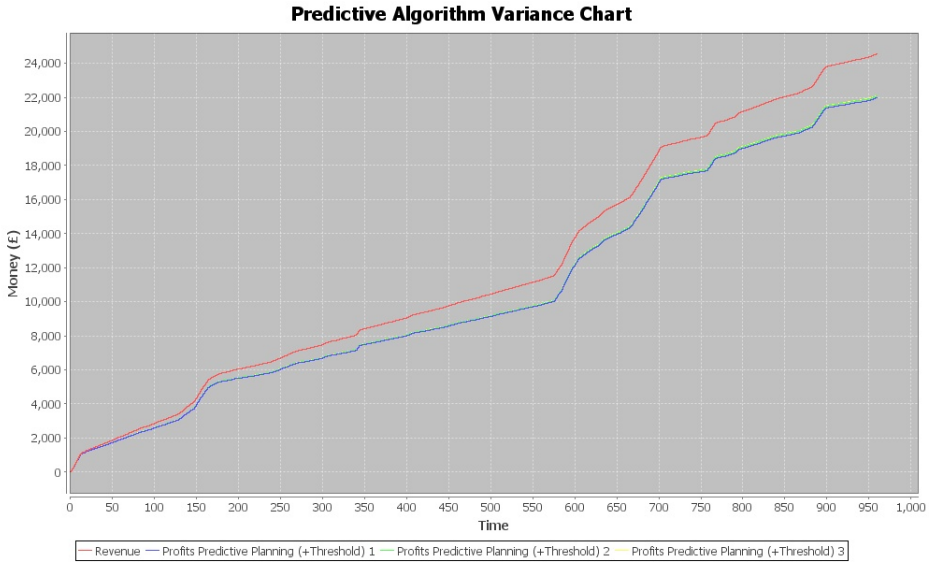
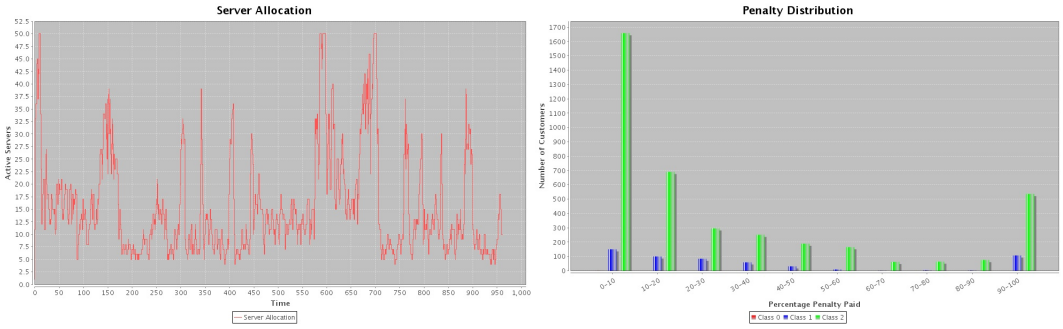


Figure 6.39: Profits Using Different Thresholds

6.4.6 Predictive Planning Policy (Future Known)

Running Costs	Switching Costs	Penalties	Revenue	Profits
755.30	230.90	353.21	24553.61	23214.20

In this simulation we begin to see the effects of transient arrival streams with large differences in rates. Although the policy achieved maximum profits yet again, this was at the expense of around 2000 customers of Class 2 and 300 customers of Class 1. We begin to see the effects of the trade off between customer satisfaction and maximization of profits for the first time. The threshold policy had significantly higher customer satisfaction levels at a sacrifice of 5% of profits, a factor which could be considered in a realistic setting.



(a) Allocation (b) Penalty Distribution

Figure 6.40: Utilization Based Policy Results

This high penalty rate could imply a new strategy might be required. None the less, the advantages of a dynamic policy with a predictive mechanism are realised through the maximization of profits, leaving the next simulation to be the judge of the extent of this advantage.

6.4.7 Evaluation of Simulation 4

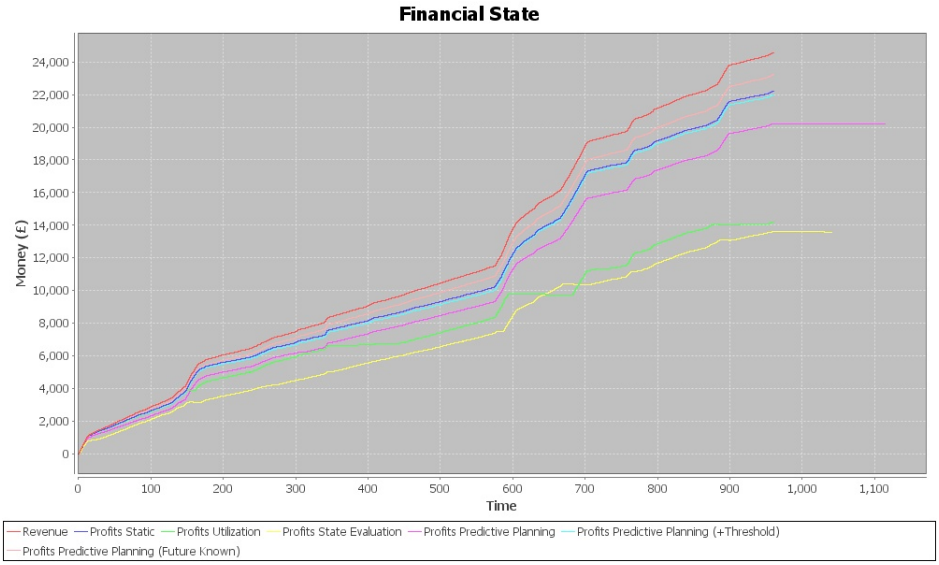


Figure 6.41: Simulation 4: Profits for all policies

The large difference between arrival rates in the MMPP seems to affect the system dramatically. We can see these effects clearly from the above figure. First thing we notice is that the time to service 30 000 customers differences significantly due to the different allocations used. Furthermore, profits have huge differences between them, with the predictive policy (future known) taking a significant lead, followed by the optimal static allocation. The threshold policy does follow the optimal static policy profits closely, but has $\frac{1}{3}$ of the penalties, indicating higher customer satisfaction. Even in the face of large differences in arrival rates, the predictive policies do not have large variance between runs, indicating they are generally stable algorithms.

The policies which use only local information have been deemed useless under transient arrival streams due to low profits and high penalties, attributed to the information they have at their disposal to make their decisions.

6.5 Simulation 5

The final simulation will take simulation 4 to the next level and increase the probabilities between states to allow for rapid fluctuations. This is the ultimate simulation to see how the policies deal with unpredictable arrival streams of a relatively large customer base.

- **Class 0, 1, 2**

$$\bar{\Lambda} = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 50 \end{pmatrix} \quad \bar{Q} = \begin{pmatrix} 0.34 & 0.33 & 0.33 \\ 0.33 & 0.34 & 0.33 \\ 0.33 & 0.33 & 0.34 \end{pmatrix}$$

6.5.1 Static Policy

Allocation	Running Costs	Switching Costs	Penalties	Revenue	Profits
17	815.59	0.0	271.03	23337.18	22250.56
18	863.55	0.0	51.49	23337.18	22422.14
19	911.53	0.0	12.61	23337.18	22413.05

The optimal allocation in the final simulation is 18 servers. The usual observation that the penalties incurred by the optimal static policy are unacceptably high still apply and are depicted below. Another important observation is that the increasing of the rate of transitions from state to state seems to have decreased the need for higher allocations.

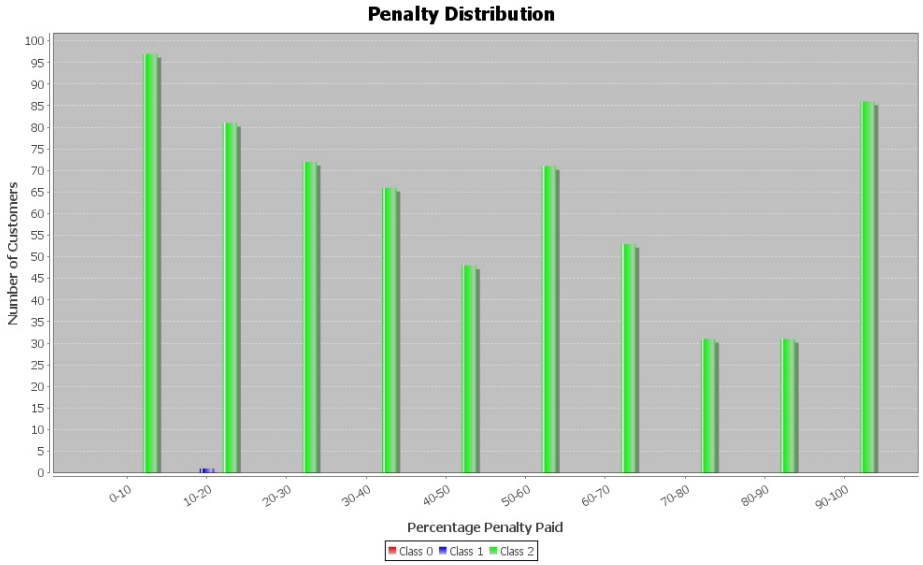


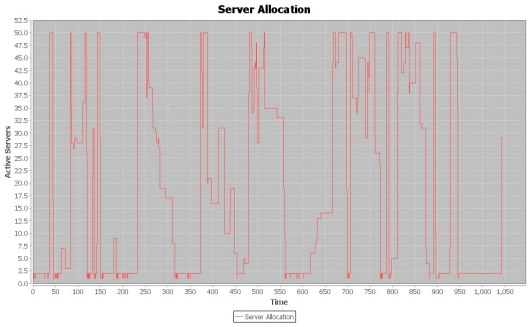
Figure 6.42: Penalty Distribution

6.5.2 Utilization Based Policy

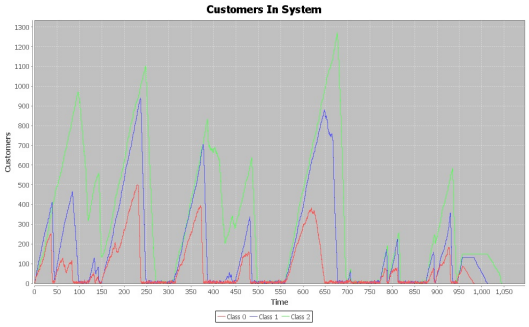
Running Costs	Switching Costs	Penalties	Revenue	Profits
965.46	183.00	12364.22	23337.18	9824.527

The effect of increasing the rate of transitions on this policy is clearly seen in Figure 6.43a where we have not only rapid switching, but large fluctuations in the number of servers from one time period to the next. This, along with the high penalties depicted below, are the cause of such low profits. These fluctuations can be linked to the peaks

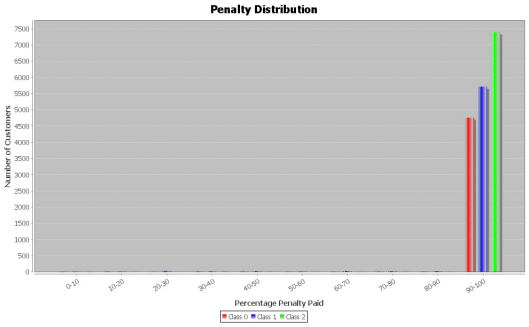
of customers shown in Figure 6.43b. This policy continues to deteriorate as we increase the volatility of the system parameters, indicating it is not a good policy for transient systems.



(a) Allocation



(b) Customers In System



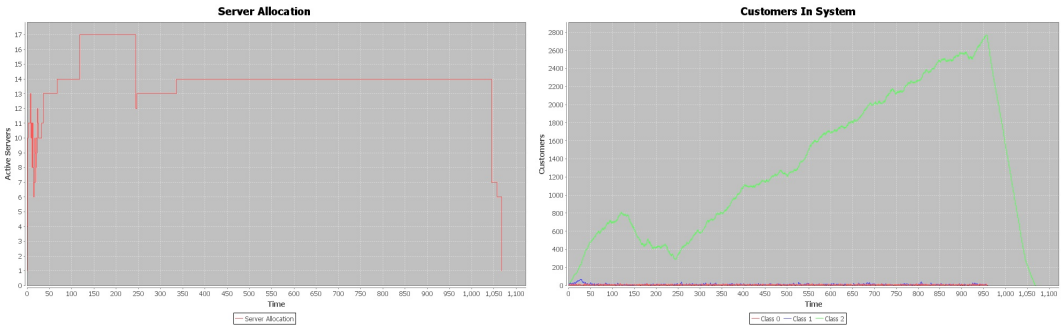
(c) Penalty Distribution

Figure 6.43: Utilization Policy Results

6.5.3 State Evaluation Policy

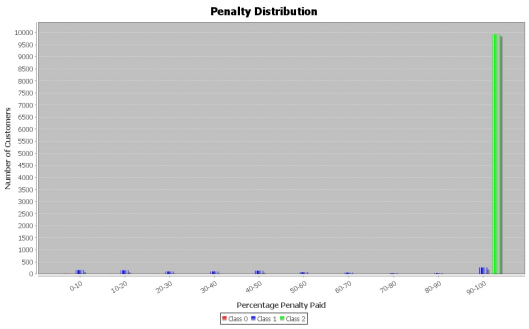
Running Costs	Switching Costs	Penalties	Revenue	Profits
751.23	230.20	9985.76	23337.18	13586.45

This simulation is the final step to confirming our observation that that this policy performs better on transient arrival streams with rapid state transitions. From this simulation we can establish that this theory is dependant on the arrival rates. In simulation 2, when rates were low, the theory was correct, however, in this simulation which uses high arrival rates, we can see that the theory cannot be applied. An observation that we have noticed in simulations 1, 2 and 3 which still holds is that the allocation selected is closely linked to the optimal static policy, as shown in Figure 6.44a.



(a) Allocation

(b) Customers In System



(c) Penalty Distribution

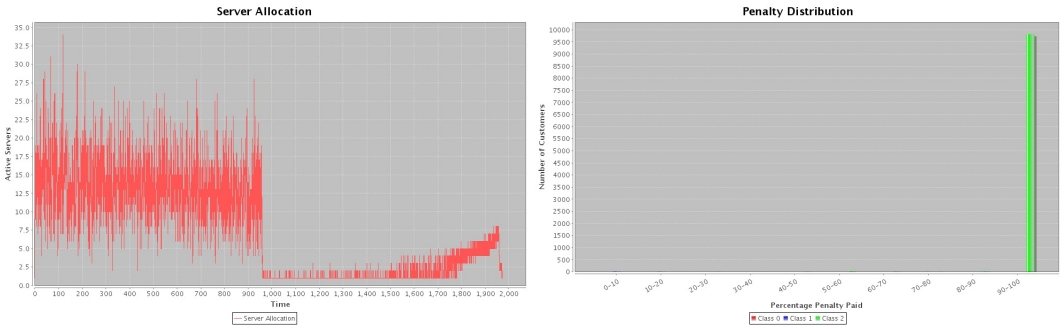
Figure 6.44: State Evaluation Policy Results

None the less, this policy yields high customer dissatisfaction and can be dismissed as a liable policy for transient arrival streams. With a few tweaks this policy could have the potential to achieve better results and reduce the penalty payout.

6.5.4 Predictive Planning Policy

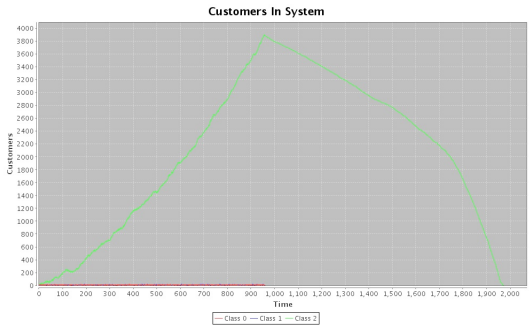
Running Costs	Switching Costs	Penalties	Revenue	Profits
744.19	2251.63	1778.71	23337.18	18562.67

As expected, this policy's results deteriorated in the ultimate test of transient arrival stream. The huge switching costs and large penalty payout make it an undesirable policy which needs improving. Figure 6.45c shows a large build up of customers within the system, making it hard to cope with the rapid arrival of customers. This is probably due to the inexplicable allocation drop after time 950. This could be attributed to the same flaw found in the threshold policy in simulation 3, but the system under-estimates arrivals rather than over-estimates. This however should not have affected the system this dramatically so as to leave only 1 or 2 servers on! This could indicate a flaw or bug with the predictive mechanism used.



(a) Allocation

(b) Penalty



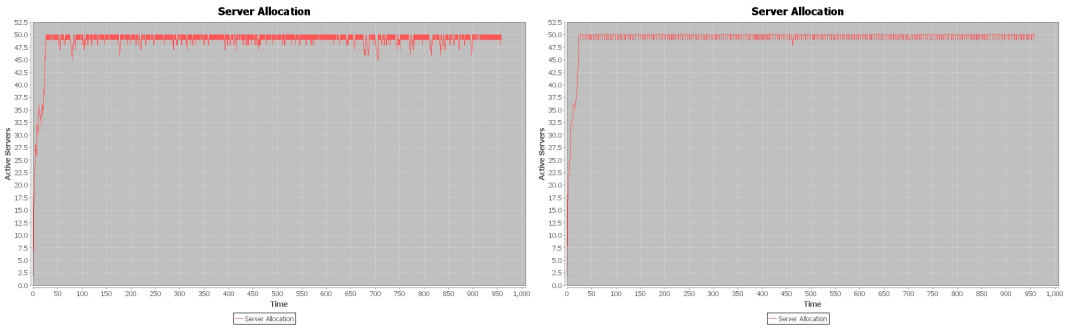
(c) Customers In System

Figure 6.45: Predictive Planning Policy Results

6.5.5 Predictive Planning Policy (with Threshold)

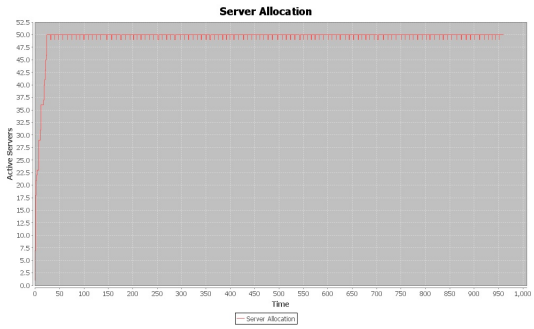
Threshold	Running Costs	Switching Costs	Penalties	Revenue	Profits
20	2352.14	161.30	0.0	23337.18	20823.74
80	2371.92	51.40	0.0	23337.18	20913.87
180	2374.54	26.10	0.0	23337.18	20936.55

Increasing the threshold on this arrival stream seems to have the same effects as the 4^{th} simulation with respect to allocation. Referring to the figures below, we can see the same thinning of lines indicating more wide-spread changes in allocation, as well as full customer satisfaction. However, given the relatively low optimal static allocation, perhaps using full use of resources is unnecessary and could be reduced. This could be due to the phenomenon we saw in the previous simulation, whereby a series of wrong decisions pushes allocation up and then is hard to bring back down.



(a) Allocation (Threshold of 20)

(b) Allocation (Threshold of 80)



(c) Allocation (Threshold of 180)

Figure 6.46: Allocation using Different Thresholds

By increasing the threshold we can see minor increments in profits, but relative improvement on switching costs. These of course are reflected as increased running costs however, they are not enough to decrease profits. As we can see below, using larger thresholds (yellow line is 180) can result in better profits throughout the run. It is possible however that the higher thresholds are the reason for increasing the instances so quickly and then finding it difficult to reduce them to 'normal' levels. Further investigation would be required into the matter to determine if this is the case but unfortunately, due to time constraints, is omitted in this report.

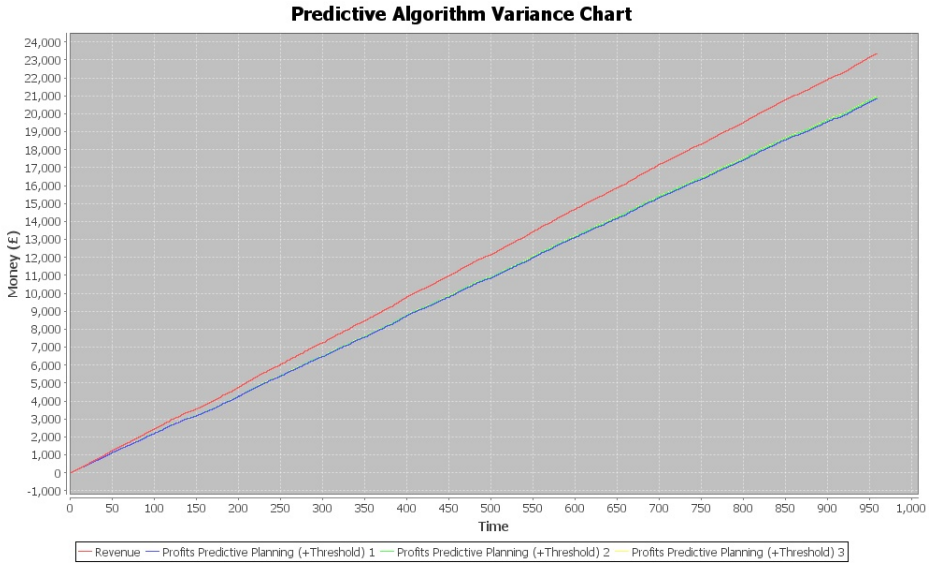
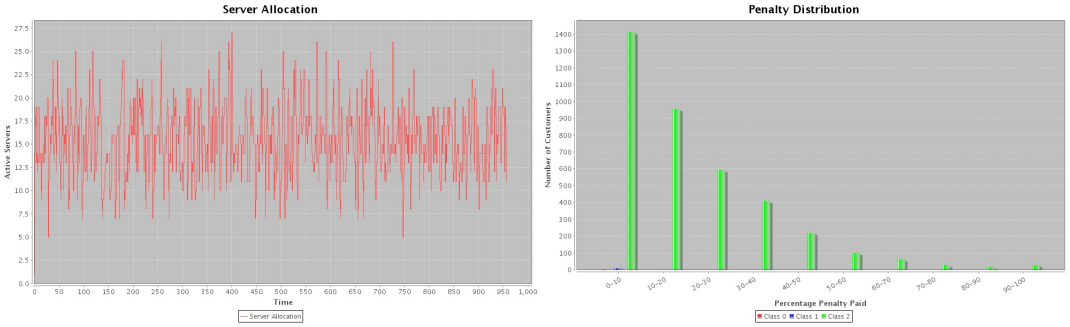


Figure 6.47: Profits using Different Thresholds

6.5.6 Predictive Planning Policy (Future Known)

Running Costs	Switching Costs	Penalties	Revenue	Profits
749.19	262.91	131.12	23337.19	22193.57

As expected, the rapid fluctuations in arrival rates have indeed deteriorated results to a point where the optimal static allocation yielded higher profits with much higher customer satisfaction. As shown by the allocation distribution below, the rapid switching resulted in high switching costs which were probably the main cause of the high penalty payout. We begin to see characteristics of the predictive planning policy and the advantages of having the threshold in place. Even without the threshold, this policy performed better than both predictive policy, yielding a 10-15% increase in profits. There is a good chance that if the predictive planning policy which uses the threshold used a better predictive mechanism, it would perform much better than this policy which knows about future arrivals!



(a) Allocation (b) Penalty Distribution

Figure 6.48: Predictive Planning Policy (Future Known) Results

6.5.7 Evaluation of Simulation 5

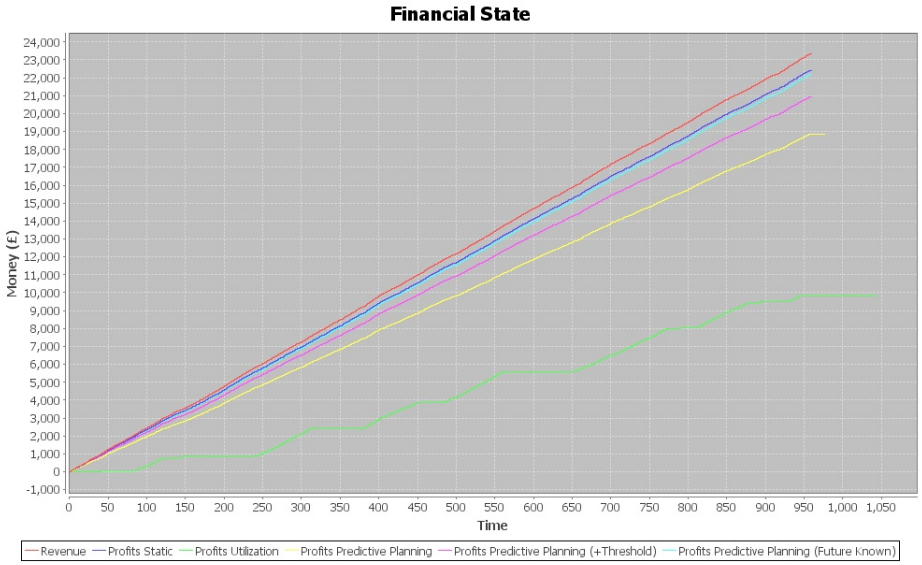


Figure 6.49: Simulation 5: Profits for all policies

This is perhaps the most interesting simulation out of the 5. Again we see the effects of having large differences between arrival rates, where each policy finishes serving the 30 000 customers at different times and large differences in profits between the policies. To our surprise, this is the first simulation which used transient arrival streams, where the optimal static policy performed better than the predictive policy (future known), in both profits and customer satisfaction. This is an indication that a good predictive mechanism is not all that is necessary to yield maximum results and that other factors should be used (such as thresholds).

This simulation is the final test to prove that policies which use only local information are inadequate under transient arrival streams, with their performance deteriorating with more rapid transitions and large differences in arrival rates.

The predictive policies on the other hand, have both performed relatively well in terms of profits, but the predictive planning policy has fallen short with respect to customer satisfaction. This simulation has proved that using the appropriate threshold is key to higher profits and higher customer satisfaction. In combination with an improved predictive mechanism, there is a good chance it could out-perform all other policies.

Chapter 7

Policy Evaluation

7.1 Static Policy

The *optimal* static policy has, surprisingly, yielded the most profits in all 5 simulations (excluding the predictive policy which knows future arrival streams). However, for transient arrival streams, this comes at the cost of high customer dissatisfaction. If switching costs were lower, it is most likely this policy would not yield the highest profits. Furthermore, as shown from 3 different static allocations in each simulation, getting the optimal static allocation is very difficult and unpredictable, with significant differences between each allocation. For example, assigning 1 server less than the optimal static allocation, usually leads to 10 times the penalty costs! These negative effects increase as the arrival stream fluctuates more in magnitude and rate.

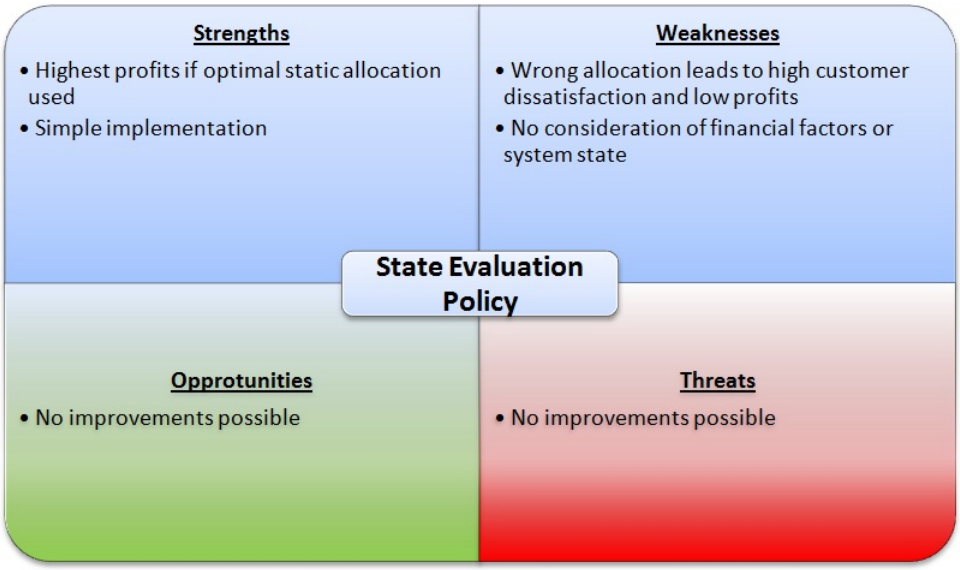


Figure 7.1: SWOT Analysis of Static Policy

7.2 Utilization Based Policy

From a service provider's point of view, this dynamic policy has proved to be the worst in all 5 simulations, especially in the simulations which used transient arrival streams. The fluctuating utilization led to rapid switching costs, with unnecessarily high allocations at times. This was to be expected from this policy since utilization ignores important factors such as customer class and financial costs. On a positive note, this high allocation led to full customer satisfaction. Note that Amazon's Auto-Scale feature which uses this technique depends on the customer deciding thresholds and uses a different financial model we do.

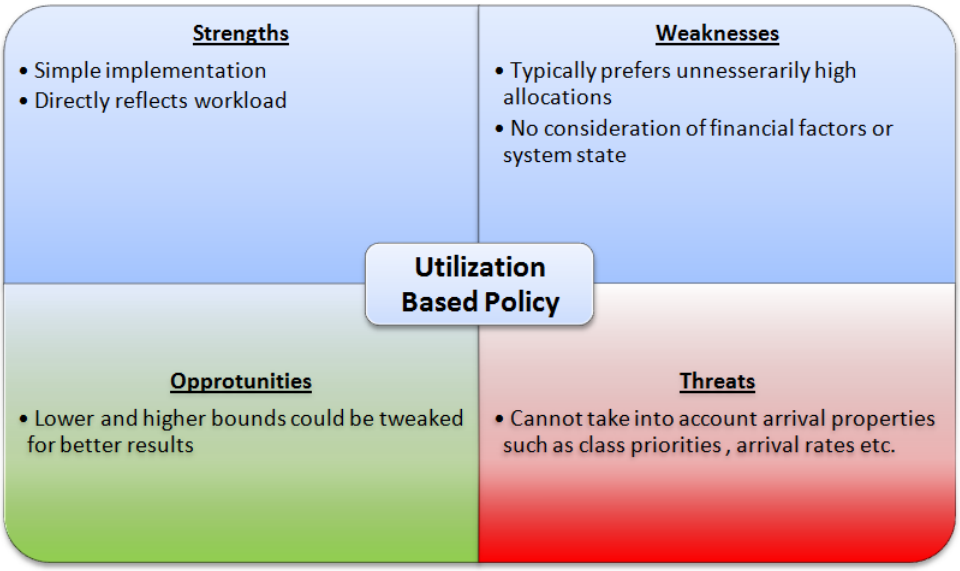


Figure 7.2: SWOT Analysis of Utilization Based Policy

7.3 State Evaluation Policy

This policy performed the best out of the 2 policies which used only local information. This was to be expected since the information the state evaluation policy uses to make decisions is directly affected by customer classes and financial parameters. Although having performed relatively well and surpassing the predictive planning policy profits under a stable arrival stream, under a transient arrival stream it did not do as well. Profits decrease significantly under the transient arrival stream due to the lack of considering the future arrivals and adapting the system accordingly. The decisions made are therefore inadequate to cope with a system with transient arrivals resulting in low profits and customer satisfaction.

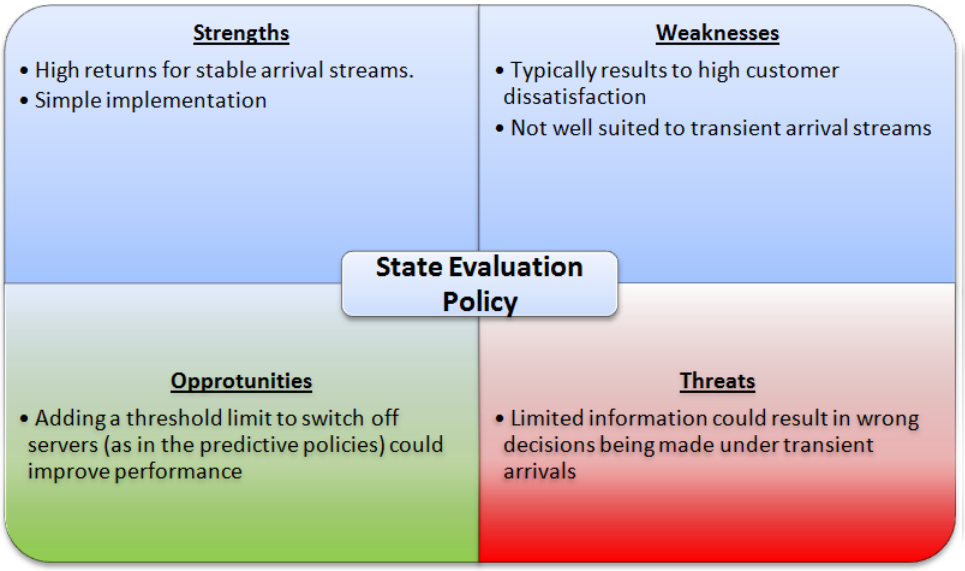


Figure 7.3: SWOT Analysis of State Evaluation Policy

7.4 Predictive Planning Policy

Regardless of stable or transient arrival stream, this policy has proved that, given a large enough window, it will provide consistent results. In simulations 2 and 4 which change arrival rates approximately every 1000 customer arrivals, the policy provided high customer satisfaction, with acceptable profits, with the obvious disadvantage of rapid switching costs. However, simulations 3 and 5 have indicated that this policy's results will deteriorate significantly if state changes are made more often. These rapid changes result in huge penalties being paid which, in combination with the high switching costs, yield low profits. In addition, simulation 5 showed that it is possible for the predictive mechanism to make wrong decisions and lead to wrong allocations. In comparison to its counterpart which knows about future arrivals, it provides significantly lower profits. These properties dismiss this policy as a reliable policy which would be used in a realistic setting.

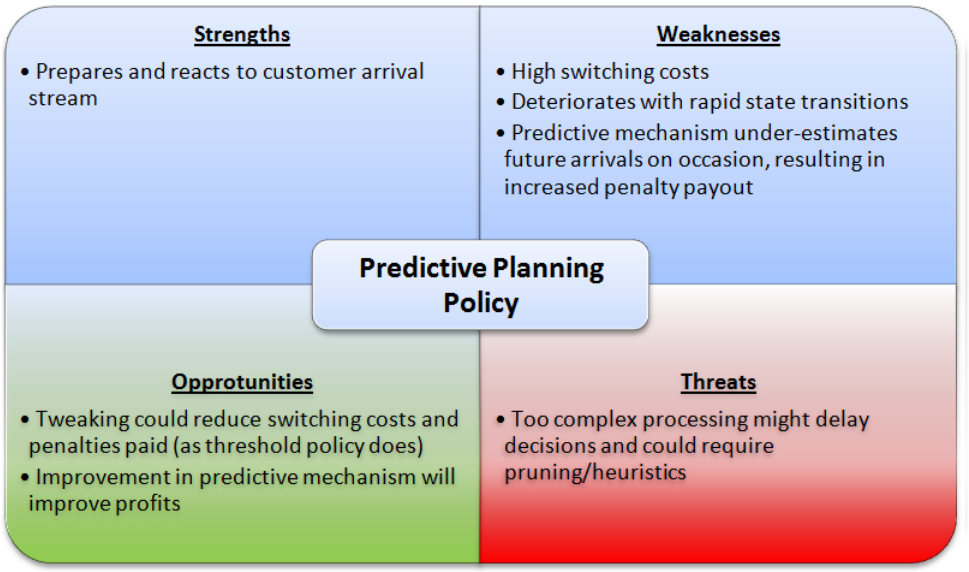


Figure 7.4: SWOT Analysis of Predictive Planning Policy

7.5 Predictive Planning Policy (with Threshold)

This policy essentially improves on the previous policy by eliminating all its flaws. Out of the dynamic allocation policies, it has yielded the most profits and the highest customer satisfaction, in all five simulations. Results are comparable to its counterpart which knows future arrivals, with only a 2% difference in profits on average. Furthermore, the profits earned are quite close to the optimal static allocation's profits (typically in the range of $3\% - 5\%$), but with much higher customer satisfaction. These are indications that this policy, if improved upon or if switching costs were less, could surpass the profits and customer satisfaction of all the other policies, including its counterpart which knows the future arrivals. In fact, this potential was clearly shown in simulation 5 which used rapidly fluctuating transient arrivals with big differences in rate parameters which closed the gap between these 2 policies.

Similarly to the policy without the threshold, given a window size greater than 20, the policy yields stable results, however, further investigation is required to determine an optimal level to set the threshold. Simulations 4 and 5, which investigate the effects of different thresholds more deeply, indicate that the higher the threshold, the higher the profits and the customer satisfaction (due to higher allocations).

Figure 6.29c indicates a flaw with the predictive mechanism. As already explained in the evaluation of simulation 3, this is most probably due to the predictive mechanism 'thinking' that a large batch of customers are going to arrive, and the system reacting to this prediction by increasing the number of active servers.

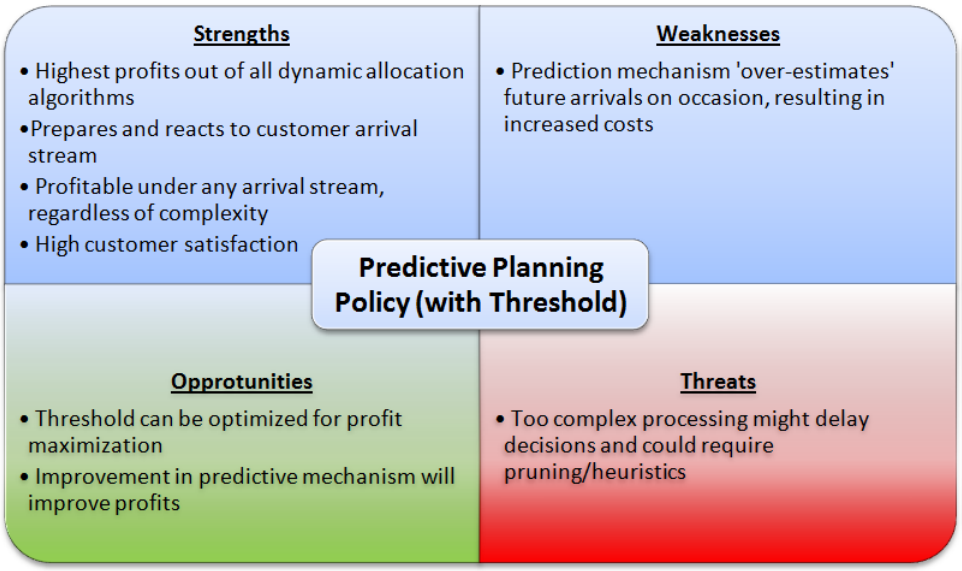


Figure 7.5: SWOT Analysis of Predictive Planning Policy (with Threshold)

Chapter 8

Conclusion

The project (a) extends the JINQS library to support our needs for a dynamic queueing system and (b) examines, through the use of simulations, different dynamic allocation policies under transient arrival streams. Through a series of different simulations we have demonstrated that the decisions made by policies are highly dependent on the information available to the policy as well as the financial parameters of the contractual obligations between clients and the provider.

The JINQS framework has been extended in many different aspects to meet our modelling needs and allows for maximum flexibility and usability without sacrificing extensibility. The user interface and graphical representation of results allow for the user run simulations easily and to observe results in a reliable and efficient manner. The 'Network Simulation' feature allows simulations to run in a distributed setting and relaxes a few assumptions. This feature could be extended for more realistic testing. Error messaging and validation of user input has also been taken into consideration to help the user when running simulations.

The policies proposed and examined have provided evidence that future planning, although difficult under transient arrivals due to its unpredictable nature, is key to ensuring maximization of profits and customer satisfaction. Policies involving only local system information have been deemed unreliable and far from optimal under our model, but have room for improvement. The two prediction based policies performed well under both stable and transient arrival streams, however, the threshold policy proved its superiority throughout our tests. The policy was compared against its counterpart which has knowledge of the future arrivals to confirm it makes decisions sufficiently well, such that profits are within an acceptable sub-optimal range. In addition, simulations have shown that an improvement in the predictive mechanism behind the policy could improve results even more and (according to simulation 5) could out-perform it counterpart which know the future arrival stream!

Future Work

The project can be changed in many aspects to be improved for realism and sets the ground for future improvement of the various different policies. It would be interesting to investigate the following:

- Relax the model assumptions to allow for more realism. For example, the 'Network

Simulations' feature could be implemented in such a way that network delays are taken into consideration. However, such an extension would require an extensive restructuring of the JINQS foundations.

- Examine optimal levels of the financial variables of the system such as SLAs for each class and system costs. [19] is a good reference on what to consider when attempting such a task.
- Use contractual obligations as the basis of the agreement but further allow users to 'buy' their way into the class queue by paying more money. [2] describes the optimal payment strategy for such a system.
- Policies can be improved in various aspects. For example, one can determine optimal threshold and window size values for the threshold policy. In addition, one could also improve the prediction mechanisms to make better decisions. Furthermore, optimizing the policies to run faster through use of better admissible heuristics or by caching results could also be beneficial.
- It would be interesting to see the performance of the dynamic allocation algorithms under a series of different inter-connected queues and servers.

All the above are worth-while tasks to take on. However, it may be in the best interest of the person who will take on any task to investigate realistic financial values of such systems (our SLA values were taken from Amazon's EC2 but switching and running costs were estimated) as they directly affect decisions, profits and hence optimality. Furthermore, it would be interesting to use a simulated system which uses real jobs such as image processing or intense calculations such as distributed large matrix multiplication.

Bibliography

[1] J. Slegers, I. Mitrani, and N. Thomas, "Evaluating the optimal server allocation policy for clusters with on/off sources," *Perform. Eval.*, vol. 66, no. 8, pp. 453–467, 2009.

[2] N. Chee-Hock and S. Boon-Hee, *Queueing Modelling Fundamentals With Applications in Communication Networks.* John Wiley & Sons, Ltd, 2008.

[3] W. L. Winston, *Operations Research: Applications and Algorithms.* Curt Hinrichs, 2004.

[4] "Amazon EC2." `http://aws.amazon.com/ec2/`.

[5] P. N. Daniel A. Menasce, "Understanding Cloud Computing: Experimentation and Capacity Planning," in *Proc. 2009 Computer Measurement Group Conf.*, (FairFax VA, USA), 2009.

[6] J. Palmer, I. Mitrani, M. Mazzucco, P. McKee, and M. Fisher, "Optimizing Revenue: Service Provisioning Systems with QoS Contracts," in *ICE-B*, pp. 187–191, 2007.

[7] M. Mazzucco, I. Mitrani, J. Palmer, M. Fisher, and P. McKee, "Web Service Hosting and Revenue Maximization," in *ECOWS '07: Proceedings of the Fifth European Conference on Web Services*, 2007.

[8] A. Kiren, "Dynamic Resource Allocation in Queueing Networks Simulation and Experimentation Framework," Master's thesis, Imperial College London, 2009.

[9] L. I. Sennott, *Stochastic Dynamic Programming and the Control of Queueing Systems.* John Wiley & Sons, 1999.

[10] M. N. Bennani and D. A. Menasce, "Resource Allocation for Autonomic Data Centers using Analytic Performance Models," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, (Washington, DC, USA), pp. 229–240, IEEE Computer Society, 2005.

[11] D. A. Menasce and E. Casalicchio, "A Framework for Resource Allocation in Grid Computing," in *MASCOTS '04: Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, (Washington, DC, USA), pp. 259–267, IEEE Computer Society, 2004.

[12] M. A. Kaboudan, "A dynamic-server queuing simulation," *Comput. Oper. Res.*, vol. 25, no. 6, pp. 431–439, 1998.

[13] B. Ciciani, A. Santoro, and P. Romano, "Approximate Analytical Models for Networked Servers Subject to MMPP Arrival Processes," *Network Computing and Applications, IEEE International Symposium on Network Computing and Applications*, vol. 0, pp. 25–32, 2007.

[14] D. Gross and C. M. Harris, *Fundamentals of Queueing Theory.* John Wiley & Sons, 1998.

[15] P. S. Steven L. Scott, "The Markov Modulated Poisson Process and Markov Poisson Cascade with Applications to Web Traffic Modeling," 2003.

[16] W. Fischer and K. Meier-Hellstern, "The Markov-modulated Poisson process (MMPP) cookbook," *Perform. Eval.*, vol. 18, no. 2, pp. 149–171, 1993.

[17] S. W. M. Au-Yeung, *Response Times in Healthcare Systems.* PhD thesis, Imperial College London, January 2008.

[18] L. Kleinrock, *Queueing Systems Volume II: Computer Applications.* John Wiley & Sons, 1976.

[19] D. A. Menasce and E. Casalicchio, "Quality of Service Aspects and Metrics in Grid Computing," 2004.

[20] "MATLAB." `http://www.mathworks.com/`.

[21] "simul8." `http://www.simul8.com/`.

[22] T. Field, "JINQS: An extensible library for simulating multiclass queueing networks," 2006.

[23] T. Field and J. Bradley, "Simulation and modelling (lecture notes),"

[24] "JFreeChart." `http://www.jfree.org/jfreechart/`.

Appendix

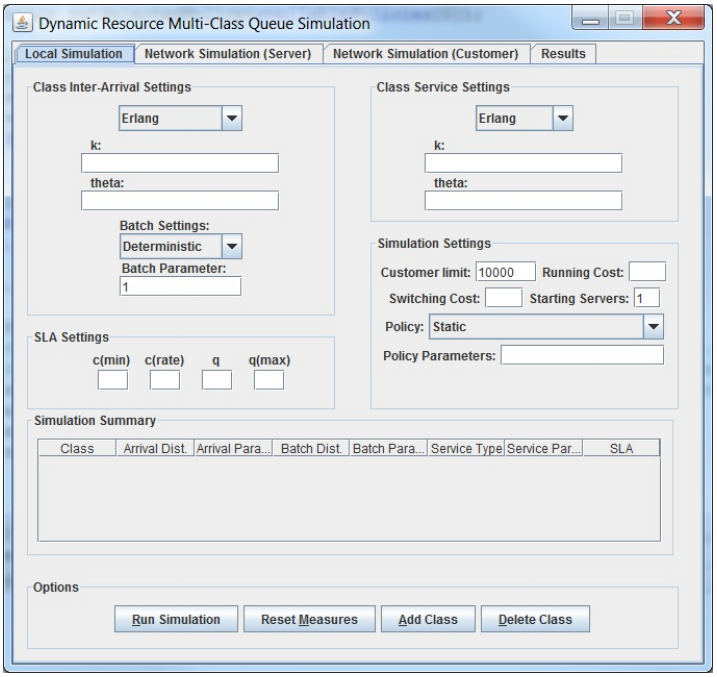


Figure 8.1: Local Simulation Options

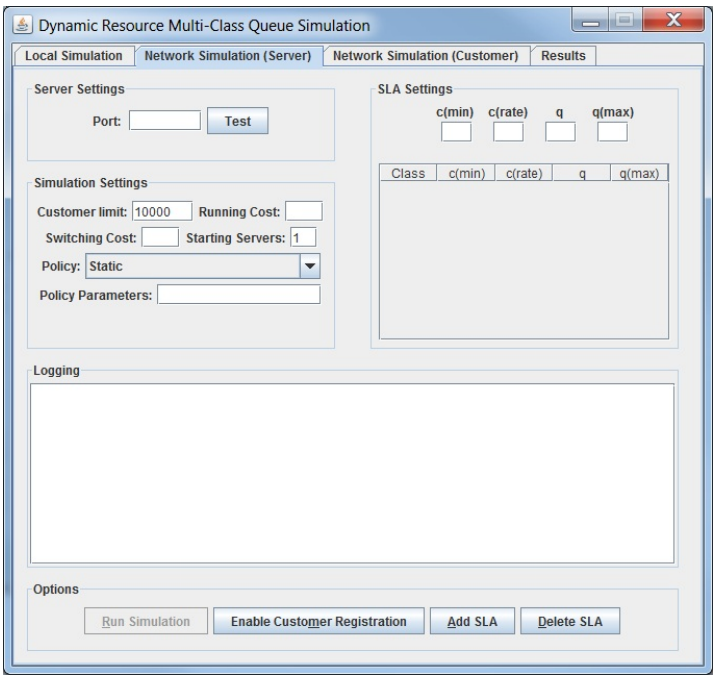


Figure 8.2: Network Simulation Server Options

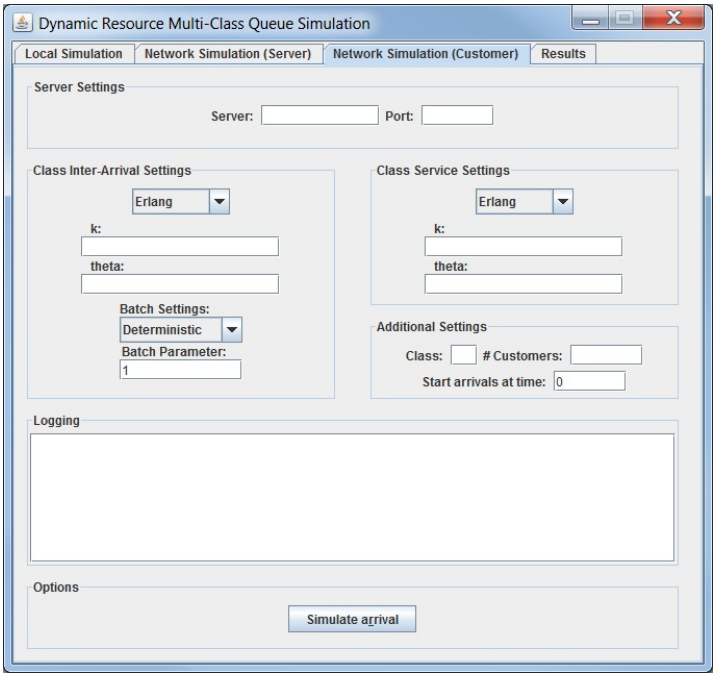


Figure 8.3: Network Simulation Customer Options

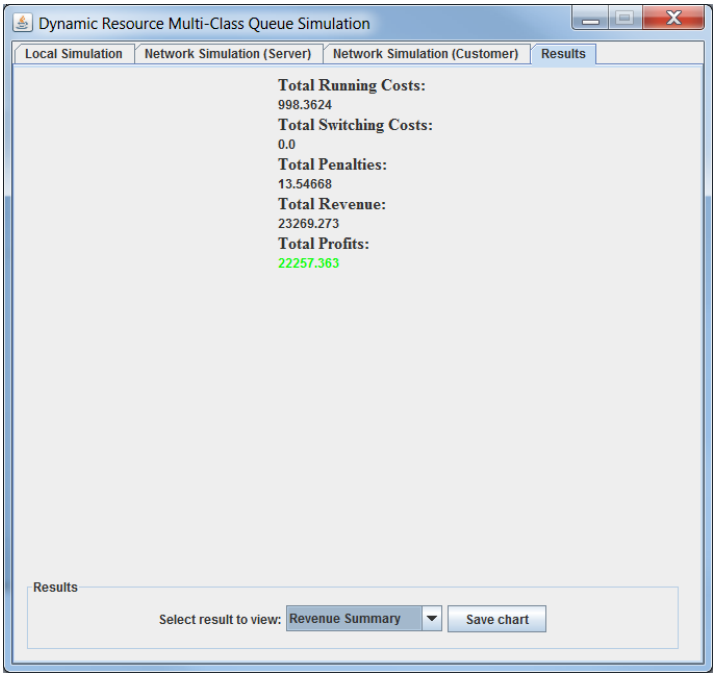


Figure 8.4: Results Tab