Studies on MapReduce based Area Skyline Query and Parameter Estimation of Queueing Systems

(MapReduceによるエリアスカイライン問合せ及びキューイングシステムのパラメータ推定に関する研究)

by

Chen Li

A dissertation submitted

Graduate School of Engineering, Hiroshima University

in partial fulfillment of the requirements for the degree of

Doctor of Engineering

in

Information Engineering

under supervision of

Yasuhiko Morimoto

Department of Information Engineering

Graduate School of Engineering

Hiroshima University, Japan

September, 2019

[This page was intentionally left blank]

Dissertation Summary

In this dissertation, the author discusses two kinds of research works, MapReduce based computation of area skyline query for location recommendation, and parameter estimation of queueing systems with utilization data, respectively.

MapReduce Based Area Skyline Query

With the rapid development of computer technology and network technology, a vast amount of available information may cause information overload problem. The exponential growth of data makes customers inundate with various choices. To solve the "information overload" problem, recommender designers develop recommendation systems. Recommendation systems are widely accepted in E-commerce in recent years. The basic premise of recommenders is to reduce noise and filter out information, which is not relevant to customers' tastes. In the past decades, many researchers worked on generating excellent recommenders. Recommendation systems can help enterprises improve economies. For example, Amazon.com [1] claimed that product sales improved 35% from the recommendation system. Moreover, Netflix.com claimed that 66% movies were rented by using recommendation systems. Google News Recommendations [1] generated 38% more click-throughs.

Location recommendations are a main part of recommendation systems. For example, the selection of good locations is essential in the service filed, which may help customers find implicit preferable places in an unfamiliar environment. Moreover, it is also critical for a businessperson to find a suitable location to build a company.

Skyline query is a well-known information retrieval approach for recommendation systems. Skyline query aims to select a small number of representative objects from an extensive database, and already has been applied in the location selection problem. For

example, a traveler wants to find a hotel from several candidate hotels. In this case, we can use the skyline query to help the traveler select an excellent hotel. However, in some real-world scenarios, the candidate points are not given for the location recommendation problem. For example, the businessman looks for a good vacant area to build a hotel at any cost. In such a situation, candidate points do not exist. Besides, the two-dimensional area is much complicated than a candidate point, which makes the problem of good location selection more difficult and challenging. However, most of the skyline algorithms to solve the problem of location selection assume that the candidate points always exist. To generate the spatial relationship between objects, we consider solving such problem by using our team's previous work, called the area skyline query.

Area skyline query is a new skyline query for selecting spatial area objects on a map. Grid-based Area Skyline (GASKY) algorithm is an efficient and practical algorithm by dividing data structure into grids for retrieving interesting areas. For example, a businessman wants to find a good area to build a hotel. The good area should be close to some preferable facilities, such as bus/train stations and sightseeing spots, and is far from some unpreferable facilities, such as open landfills and noisy places. In such a situation, GASKY first divides a square region on a map into several girds. Then, we calculate the minimum and maximum distance from an area to the closest preferable facility and the farthest unpreferable facility of each facility type. After the calculation of the minimum and maximum distance of every grid, the problem of the area skyline query can be transformed into the conventional skyline query. However, the computational cost of the time complexity for GASKY algorithm is much higher than the conventional skyline query. Furthermore, the average processing time increases linearly with the growth of facilities.

To resolve the poor performance problem of GAKSY and to handle "big data" well, we consider a distributed algorithm for computing the GASKY.

MapReduce is a programming model and an associated implementation for processing

big data. Mainly, a MapReduce program is composed of a Map function and a Reduce function. The Map function takes a set of data and converts it into another collection of data, where individual elements are broken down into tuples. The Reduce function takes the output from the Map function as an input and combines those data tuples into a smaller set of tuples. In recent years, the MapReduce framework is widely used for computing skyline query. In general, there are three main MapReduce based algorithms for skyline query processing, called MR-BNL, MR-SFS, and MR-Bitmap. Unfortunately, only a few algorithms focus on spatial skyline query in such parallel way. The GASKY algorithm is a new skyline query for selecting spatial area objects. In this work, we propose a novel algorithm to improve the performance of the GASKY algorithm and handle large-scale database.

Parameter Estimation of Queueing Systems

In the second work, the author discusses the parameter estimation problem of queueing systems.

Performance evaluation plays a vital role in the computer design phase. Performance evaluation can help computer designers determine the optimal system configuration, such as the size of memory, the number of CPU, and the storage capacity. As the complexity of system increases, the new system architecture tends to integrate a collection of independent systems, called a system of systems (SoS). To configure such kind of an SoS, performance evaluation becomes more challenging and more critical.

In general, there are three main methods for performance evaluation of computer systems, measurement-based performance evaluation, simulation-based performance evaluation, and model-based performance evaluation, respectively. If the system already exists, we usually use the measurement method to evaluate performance. When the system is under designing, we select to use simulation-based method and model-based method for

performance evaluation. Simulation-based performance evaluation is more flexible, accurate, and credible, but require more time to derive models. Model-based performance evaluation constructs a systematic mathematical model and quantitatively analyzes the model. For example, Markov chains, queuing systems, and queuing networks are conventional techniques of model-based methods, which are usually used in telecommunication systems. Although to some extent, the model-based approach is an approximation, the amount of calculation is small.

Queueing systems for model-based performance evaluation are very common. Typically, a single queue has three main components: input, queue, and server. The input is a stream of customers or jobs, which wait for service in the queue. The jobs leave the system after the service. More specifically, we usually define a queueing system by the symbol $A/S/m/k$. A is the type of the arrival process, and S is the distribution of the service time of customers. m is the number of servers, and k is the queue capacity. For example, $M/M/1/K$ means that the customers arrive at the service facility via the Poisson process, and the service time follows an exponential distribution. Moreover, there is only one server in the system with a capacity of k. In other words, the input processes and service distributions are two necessary components for a queueing system. Unfortunately, in practice, it is not easy to model the input process and service distribution. In general, we must observe the arrival process and service time in a fixed time, and then we can estimate the arrival rate and service rate. In most queueing systems, the arrival process and service time is observable. In other words, we can know the exact job arrival time and service time by a period of observation. However, the parameter estimation of the arrival processes in computer systems, such as CPU utilization, are quite limited.

Moreover, we usually assume that the arrival rate is constant for a queueing system. However, in the real world, the fixed arrival rate cannot model the arrival process very well. For example, the customers' arrival rates in a convenient store are always varying

dynamically. The arrival rate of morning and evening rush hours may be higher than the rate of other periods.

Non-Homogeneous Poisson Process (NHPP) is a Poisson process over a non-linear time scale, which has already been used to solve complex independent arrival processes. In this work, the author considers an NHPP for the job arrival process of a computer system, and generate an $M_t/M/1/K$ queueing system. The author aims to estimate the arrival rate, which varies as a function of time by using utilization data.

To summarize, this dissertation introduces two kinds of research works. The first research work is tilted MapReduce based computation of area skyline query for selecting good locations in a map. The other research work is titled parameter estimation of queueing systems with dynamic arrival process from utilization data, respectively. The outline of this dissertation is organized as follows.

Chapter 1 discusses the motivations of the two research works and gives the organization of this dissertation.

In **Chapter 2**, the author introduces the first research work in detail. In this chapter, we first introduce the definitions of skyline query, spatial skyline query, and area skyline query. Secondly, we review the related works. And then, we propose our novel MapReduce based algorithm for the calculation of area skyline query. Finally, we conduct two numerical experiments on both the synthetic dataset and real dataset, The experimental results confirm that our proposed model is efficient and effective for handling "big data."

In **Chapter 3**, the author presents the detail of the other research work: parameter estimation of queueing systems with dynamic arrival process from utilization data. In this chapter, we first give some definitions of queueing system, utilization data, and Non-Homogeneous Poisson Process. Secondly, we review the related works. And then, we introduce a parameter estimation method, which is called maximum likelihood estimates. Also, we propose to use the Expectation-Maximization algorithm to overcome the

incomplete information problem. Finally, we conduct two experiments on both synthetic utilization data and real CPU utilization data to discuss the performance of our proposed model.

Finally, a concluding discussion of the contributions of the two research works and conclusions are discussed in **Chapter 4**.

Contents

List of Figures

List of Tables

The only way to do great work is to love what you do.

Steve Jobs

Acknowledgments

First of all, I would like to express my special appreciation and thanks to my supervisor Prof. Yasuhiko Morimoto, Graduate School of Engineering, Hiroshima University, Japan, for allowing me to explore new ideas in the field of knowledge discovery and data mining. He has been a tremendous mentor for me. I want to thank him for encouraging my research, for the continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. He is my inspiration to be a better scientist and researcher.

I would like to thank my committee members, Prof. HIROYUKI OKAMURA and Prof. KOJI EGUCHI, for letting my defense be an enjoyable moment, and also for their brilliant comments and suggestions that enrich my knowledge.

Also, thanks to all my co-authors and lab members, for their support and contributions during my research. Primarily I'm grateful to Dr. Annisa and Dr. A. Zaman for their generous support and cooperation since the beginning of my research work. I am also thankful for all of my countrymates. Because of their collaboration, support, and warm relationships during my stay in Japan, I can complete the doctoral dissertation.
I am deeply grateful to my family members for their comprehension, dedication, care, and support throughout the research work.

Finally, I hope this work will give significant contributions and advantages, especially for data management research in Japan and China.

[This page was intentionally left blank]

Chapter 1

Introduction

1.1 MapReduce Based Area Skyline Query

Information retrieval is a very critical part of data mining, such as the location selection problem. In general, the cases of selecting hotels, restaurant, and sightseeing spots, an excellent location may have a direct impact on time, cost, and effort. Therefore, selecting the right place can help customers and people in business save some time and cost. For example, in business filed, if a businessperson wants to a location for his company, he should consider some sites, which can attract many potential customers. The good sites should be close to the places, such as the retail centers, train/bus stations, commercial district, etc. The proximity to such locations can bring profit and benefit for the site. Besides, the businessman should also consider potential competitors and other unpreferable factors. For example, the places, such as factories, open landfills, and noisy sources, may reduce profit and bring unfavorable effect on the location.

Therefore, a good location should be close to some preferable facilities and be far from some unpreferable facilities. For example, in the business field: a businessman aims to select a location for his new company. The businessman finds some candidate locations, and he would like to choose an apartment from the candidate locations. To attract potential

customers, the site of the company should be in the region that is close to train/bus stations and commercial districts and is far from open landfills.

1.1.1 Motivation

Notice that there are some candidate points, such as the rooms of a hotel, in the above example. We aim to select the interesting points from a set of candidate points. To resolve the problem, we apply the well-known technique, which we call "skyline query." In the information retrieval field, a skyline is a simplified approach for selecting a small number of interesting data objects from a large database. In this situation, locations in a map are typical two-dimensional data. Since most of the existing skyline query approaches can only select zero-dimensional data, we consider applying the spatial skyline query to solve the problem.

Spatial skyline query can select a small number of the two-dimensional location objects from a set of geographic candidate data. However, in some real-world situation, the candidate points do not exist. For example, a businessman wants to build a supermarket in an excellent free region. For this problem, the candidate points are not given, and the businessman must find the ideal area in a map. In other words, he/she must find a two-dimensional location, which is close to the preferable facilities and is far from the unpreferable facilities on a map.

Since the two-dimensional area is much complicated than the zero-dimensional points, the selection of the two-dimensional data is challenging and critical. The previous work of our team has proposed the definition of area skyline query, which can select the two-dimensional data in a map. However, the time complexity is worse than the conventional skyline query. Furthermore, the processing time increases linearly with the growth of the number of facility types, the number of grids, and the number of objects.

In recent years, distributed skyline computations for big data has received more atten-

tion. For example, MapReduce is a popular parallel programming model for processing large amounts of data. To reduce the complexity of the algorithm, and to improve the performance of the area skyline query, the author proposes a novel parallel algorithm. The parallel algorithm can solve area skyline query computation in the MapReduce framework, which we call "MRGASKY."

1.2 Parameter Estimation of Queueing Systems

In another work, the author proposes a novel model to estimate the parameter of queueing systems by using utilization data.

Parameter estimation is always of great importance in the computer design phase. In general, parameter estimation can capture the underlying structure of the whole system. For example, the statistical inference can determine the optimal system configuration, such as the number of CPU, memory size, and hard disk capacity. In recent decades, the integration of complex systems, which we call "a system of systems (SoS)" has drawn the attention of researchers. To configure such kind of an SoS, performance evaluation becomes more challenging and more critical.

Queueing systems, a typical method for model-based performance evaluation, are widely used in various fields. The problem of parameter estimation of queueing systems consists of several components, such as input, queue, and server. The input is a stream of customers or jobs, which wait for service in the queue. The customers or jobs leave the system after the service. More specifically, we usually define a queueing system by the symbol $A/S/m/k$. A is the type of the arrival process, and S is the distribution of the service time of customers. m is the number of servers, and k is the queue capacity. For example, $M/M/1/K$ means that the customers arrive at the service facility via the Poisson process, and the service time follows an exponential distribution. Moreover, there is only one server in the system with a capacity of k. In other words, the estimation of the arrival rate and service rate are

the essential and challenging thing in the queueing systems.

1.2.1 Motivation

The Poisson distribution is a discrete distribution that observes the counts of the event in a given interval of time. Define the parameter of the Poisson distribution as the mean number of events per an interval of time. We usually use the Poisson distribution as the arrival rate of a queueing system. In other words, the arrival rate is constant in this case. However, in the real world, the fixed arrival rate cannot model the arrival process very well. For example, the arrival rate of the customers is always changing dynamically in a convenient store. Usually, the arrival rate in the morning rush hour and mealtime may higher than the rate of midnight.

In this case, we usually use Non-Homogeneous Poisson Process (NHPP) to represent the arrival process. NHPP is a Poisson process over a non-linear time scale, which has already been used to solve complex independent arrival processes.

On the other hand, when we estimate the arrival process and service distribution, we need to observe the system in a fixed time. In some observable queueing systems, we can observe the exact job arrival time and service time by a period of observation. However, in computer systems, some observation cannot get easily. For example, CPU utilization data can reflect the utilization of computing resources and are a widespread type of data in computer systems. However, we can not observe the arrival process and service process of jobs directly from the CPU utilization data. In this case, the parameter estimation is quite limited.

In this work, the author considers an NHPP for the job arrival process of a computer system, and generate an $M_t/M/1/K$ queueing system. The author aims to estimate the arrival rate, which varies as a function of time by using utilization data.

1.3 Thesis Organization

To summarize, this dissertation introduces two kinds of research works. The first work is titled "MapReduce based computation of area skyline query for selecting good locations in a map." The other research work is titiled "parameter estimation of queueing systems with dynamic arrival process from utilization data." The rest of the thesis is organized as follows.

In **Chapter 2**, the author introduces the first research work in detail. In this chapter, we first introduce the definitions of skyline query, spatial skyline query, and area skyline query. Secondly, we review the related works. And then, we propose our novel MapReduce based algorithm for the calculation of area skyline query. Finally, we conduct two numerical experiments on both the synthetic dataset and real dataset. The experimental results confirm that our proposed model is efficient and effective for handling "big data."

In **Chapter 3**, the author presents the detail of the other research work: parameter estimation of queueing systems with dynamic arrival process from utilization data. In this chapter, we first give some definitions of queueing system, utilization data, and Non-Homogeneous Poisson Process. Secondly, we review the related works. And then, we introduce a parameter estimation method, which is called maximum likelihood estimates. Also, we propose to use the Expectation-Maximization algorithm to overcome the incomplete information problem. Finally, we conduct two experiments on both synthetic utilization data and real CPU utilization data to discuss the performance of our proposed model.

Finally, a concluding discussion of the contributions of the two research works and conclusions are discussed in **Chapter 4**.

Chapter 2

MapReduce-based Area Skyline for Location Recommendations

2.1 Introduction

With the advancement of technology, more and more people are inseparable from mobile devices and GPS systems. For example, Google announced that there over 2 billion monthly active users use Google Maps on the Android system. Mobile devices usually have limitations on display size and processing power. We can generate recommendation systems to recommend the information of potential interest, such as good locations of Google Maps, to the users. However, with the ever-changing environments and locations of mobile users, it challenges the researchers to generate appropriate recommendation systems in mobile devices.

2.1.1 Skyline Query

In the information retrieval field, skyline operation [2] is a well-known approach for selecting a small number of interesting objects from a large database. Let $\mathbf{D}$ be a n-dimensional database. $D = \{d_1, d_2, ..., d_n\}$ are the n attributes of $\mathbf{D}$.

Definition 2.1.1. *(Dominance) Given a set of points* $\{p_1, p_2, ..., p_r\}$*, a point* p_i *is said to dominate another point* $p_j (i \neq j)$ *if* p_i *is not worse than* p_j *in any of the d-dimensions and* p_i *is better than* p_j *in at least one of the n-dimensions.*

The skyline query returns some objects; the objects are not dominated by each other. Each of the objects dominates at least one other objects in **D**. Since the skyline query has an excellent ability in filtering the uninteresting objects, the author utilizes the skyline query for the location recommendation problem.

In general, a good hotel should be close to excellent facilities, such as bus/train stations, sightseeing spots, and shopping malls. Also, it should be far from unpreferable facilities, such as other competitors, noise pollution areas, and open landfills. In this example, we want to choose a hotel, which has a lower cost, and a shorter distance to the bus/train station. But in general, the hotels near the bus/train stations usually have higher room prices; while the hotels with inconvenient transportation tend to have a lower price for the rooms.

Table 2.1 and Figure 2.1 show a typical example of skyline query for selecting locations. In Table 2.1, there are five candidate hotels $\{h_1, h_2, ..., h_5\}$ with two dimensions, which are the room price and the distance to bus/train stations. For instance, the room price of the hotel h_1 is 3, and the distance the bus/train station is 8. The hotel h_1 has the lowest room price but the longest distance; while the hotel h_4 has the highest room price but the shortest distance. In Figure 2.1, we map the five two-dimensional data into the coordinate. According to the definition 2.1.1, the hotel h_2 and the hotel h_5 are dominated by the hotel h_3. The hotels h_1, h_3 and h_4 are not dominated each other. Therefore, the skyline objects are $\{h_1, h_3, h_4\}$.

Table 2.1: A Hotel Example.

ID	Price	Distance
h_1	3	8
h_2	5	4
h_3	4	3
h_4	9	2
h_5	7	3

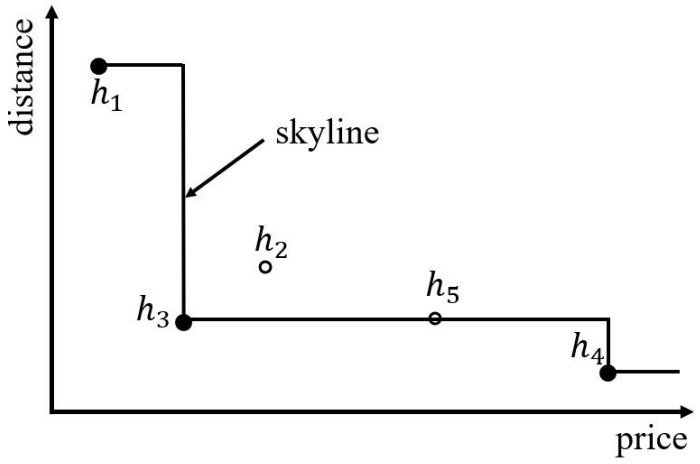

Figure 2.1: A Conventional Skyline Example.

There already existed many skyline query algorithms, which are recorded in [3, 4, 5]. However, most of the existing skyline query approaches can only retrieve zero-dimensional data, which means we cannot retrieve the two-dimensional data, such as locations. To filter the interesting objects of spatial data, we apply the spatial skyline query.

2.1.2 Spatial Skyline Query

Let P be a set of spatial points, and F be a set of facilities, which can be categorized into m types, $F = \{F_1, F_2, ..., F_m\}$. We annotate "+" mark on the facility symbol to represent preferable facilities F^+, while we annotate "−" mark on the facility symbol to represent unpreferable facilities F^-.

Figure 2.2 and Table 2.2 illustrate the skyline query for spatial points. In this case, the businessman would like to find a good hotel on a map. In Figure 2.2, we use square marks,

triangle marks, and star marks to indicate the location of facilities. The star marks, denoted as $F1^+ = \{f1_1^+, f1_2^+, f1_3^+\}$, and triangle marks, denoted as $F2^+ = \{f2_1^+, f2_2^+, f2_3^+\}$, are preferable facilities. The square marks, denoted as $F3^- = \{f3_1^-, f3_2^-, f3_3^-\}$, are unpreferable facilities.

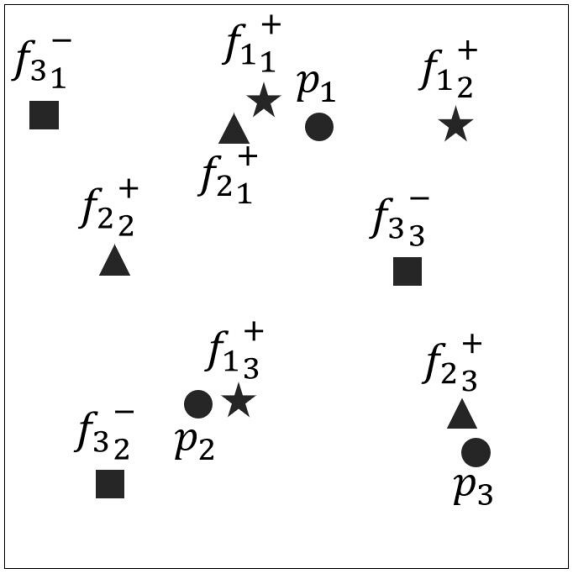

Figure 2.2: Some Facilities in a Map.

Table 2.2: Distance Table.

Point	$F1^+$	$F2^+$	$F3^-$
p_1	3	5	-10
p_2	4	9	-7
p_3	8	1	-8

Based on the map of Figure 2.2, we calculate the Table 2.2. In the table, there are three candidate points p_1, p_2, p_3. We record distance from each candidate point $p_i (i = 1, 2, 3)$ to the closest facility of $F1^+, F2^+$, and $F3^-$ types. For example, the closest $F1^+$ (star) facility from p_1 is $f1_1^+$, and the value of the distance is 3. Similarly, the closest $F2^+$ (triangle) facility from p_1 is $f2_1^+$, and the value of the distance is 5. The closest $F3^-$ (square) facility from p_1 is $f3_3^-$, and the value of the distance is 10. Notice that we assume the smaller value is better in each of the attributes. Then we multiply -1 to each distance value of

unpreferable facilities F^-. In Table 2.2, point p_1 dominates point p_2, since the values of $F1^+, F2^+$ and $F3^-$ of point p_1 are smaller than the values of point p_2. p_3 is not dominated by p_1 and p_2, since p_3 is closer to $F2^+$ than p_1, p_2 but farther to $F3^-$ than p_2. So the skyline objects are p_1 and p_3.

After comparing the distance values of each candidate points, we can calculate that the points p_1 and p_3 are the skyline objects. Therefore, we can get the spatial skyline query by calculating the two-dimensional data in Table 2.2.

In the above case, we assume that the candidate point p_i always exists. However, in some real-world scenarios, such point p_i does not exist. For example, a businessman would like to build a new supermarket in a blank area. In this case, we cannot find any candidate points on a map. We must find such an area based on more preferable facilities around it, and more unpreferable facilities away from it. The previous work of our team's, recorded in [6], Annisa have proposed the novel skyline query algorithm, called "Area Skyline Query", for selecting good areas in a map.

2.1.3 Area Skyline Query

Assume that A is a rectangular region, where a businessman wants to find an excellent location to build his/her supermarket, on a map. Let $F = \{F1, F2, ..., Fm\}$ be a set of facility types, which are categorized into m types. Notice that the "+" mark and "-" mark represent preferable facilities and unpreferable facilities, respectively. Each facility has a set of facility objects, i.e., the preferable facility $F1^+ = \{f1_1^+, f1_2^+, f1_3^+\}$. In this example, we define *area skyline* and *area dominance* as follows.

Definition 2.1.2. *(Area Skyline) For simplicity, we assume that the region A is a square region. We divide a square region A into $n \times n$ grids $G = \{g_{1,1}, ..., g_{n,n}\}$. A grid g_i is said to be in skyline if there is no other grid $g_j (i \neq j)$ in a map such that the distance of g_j to the closest preferable facilities are smaller than that of g_i and the distance to the closest*

unpreferable facilities are larger than that of g_i.

Definition 2.1.3. *(Area Dominance) Area Skyline is a set of grids, each of which is not dominated by another grid in a map. Specifically, if there exists such gird g_j, we say that g_i is dominated by g_j or g_j dominates g_i.*

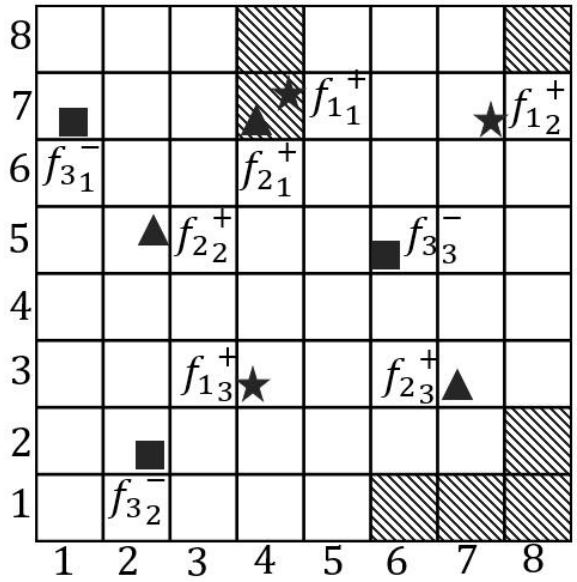

Figure 2.3: An Area Skyline Example.

Figure 2.3 is an example of the area skyline query. In this example, the square region A is divided into 8×8 grids. The facilities $F1^+, F2^+, F3^-$ are located into the grids. We calculate the area skyline objects by the algorithm proposed in [6]. We call the algorithm grid-based area skyline query (GASKY). In Figure 2.3, the shaded grids are the area skyline objects, which dominate other unshaded grids, and are not dominated each other. For example, the shaded grid at 7-th row and 4-th column is closer to the preferable facilities $f1_1^+$ and $f2_1^+$, and is farther from the unpreferable facility $f3_3^-$ than other unshaded grids.

By using the area skyline query, we can easily select good areas from a map. However, the complexity of the area skyline query is much higher than the conventional skyline query. Furthermore, the average processing time of area skyline query increases linearly with the number of facilities increasing. In this work, we propose a novel algorithm, which is based

on the MapReduce framework, to reduce the complexity and average processing time of area skyline query. We call the proposed novel algorithm "MRGASKY."

The main contributions of this work are as follows.

1. In our previous work, we have proposed the algorithm for area skyline query computation. In this work, to reduce the high cost on complexity and time processing, we propose a novel distributed algorithm for the area skyline query.

2. We propose a MapReduce-based area skyline query computation, which can reduce the complexity, and efficiently reduce the cost of time processing.

3. We conduct an extensive performance evaluation, which shows the high efficiency and scalability of our proposed algorithm.

2.2 Related Works

2.2.1 Skyline Query

In recent decades, there exist much literature working on skyline query computation. Brozsonyi et al. [2] first proposed the skyline operator in large database applications. In this paper, the author proposed three main algorithms, Block Nested-Loop (BNL), Divide-and-Conquer (D&C) and B-tree based schemes, to compute the skyline queries. BNL is a simple algorithm by comparing an object to other objects in a naive way. Sort-Filter-Skyline (SFS) is a novel algorithm, which improved BNL by the presorting operator, was proposed by Chomicki et al. [3]. Tan et al. [4] proposed *Bitmap* and *Index*, the two progressive algorithms, to improve the computation of skyline queries. To improve the retrieval ability of skyline query algorithm, Papadias et al. [5] proposed an effective algorithm, called Brach and Bound Skyline (BBS). BBS algorithm is a progressive algorithm, which is based on the Best First Nearest Neighbor (BFNN) algorithm. In general, all the above skyline

query algorithms can be categorized into two classes, index-based algorithms, and non-index-based algorithms, respectively. The BNL algorithm, D&C algorithm, and *Bitmap* algorithm belongs to the non-index-based skyline algorithm. The B-tree based schema, BF-NN algorithm, and BBS algorithm are examples of the index-based skyline algorithm.

With the development of technology of the Internet, the information shows exponential growth. Also, data becomes more and more complex, and the dimensions become higher and higher. Many researchers aim to address the dimensionality problem of skyline query, such as the problem of skyline frequency [7], k-dominant skyline [8] and k-representative skylines [9]. All the proposed skyline query processing algorithms on the above focused on the non-spatial database. However, spatial data are quite common in the real world and playing a vital role in real life. To improve the processing ability of spatial data, lots of papers worked on spatial skyline queries.

2.2.2 Spatial Skyline Query

Sharifzadeh et al. [10] first proposed the spatial skyline query to resolve the problem of spatial data processing. In the paper, they proposed three algorithms, B^2S^2 and VS^2 and VCS^2 of spatial skyline queries. B^2S^2 and VS^2 are algorithms for static query points, and VCS^2 algorithm is for dynamic query points, which exploited the pattern of change in query points to avoid unnecessary re-computation of the skyline.

Moreover, there are lots of other literature, which aim to address the problems of the spatial skyline queries [11, 12, 13, 14]. Kodama et al. [11] considered that the candidate objects should be close to the facilities. They calculated the distance between the closest facility and the candidates, and then transform the spatial, skyline problem to the conventional skyline query problem. Kodama et al. [12] proposed an algorithm, which considered non-spatial preferences also had an impact on the computation of spatial skyline. The facilities in [11] and [12] are preferable facilities.

In [13], You et al. first proposed a novel progressive algorithm based on the TFSS algorithm. The algorithm is called *Branch-and-Bound Farthest Spatial Skyline* (BBFS) to compute the farthest spatial skyline queries. The algorithm used the unpreferable facilities, and the experiment results outperformed the TFSS algorithm. However, the candidate points should be not only close to preferable facilities but also far away from unpreferable facilities. Based on this consideration, Lin et al. [14] proposed the EFFN algorithm by using preferable facilities and unpreferable facilities.

The above papers assumed that the candidate points always existed. However, the candidate objects may not exist in some real cases. Annisa et al. [15] first proposed the *Unfixed-Shape Areas Skyline* (UASKY) algorithm to retrieve a small number of objects without candidate points for location recommendations. Specifically, the authors divided a given region A into several disjoint subregions by using a Voronoi Diagram. Again, for every subregion, they executed the previous operation to divide the subregion by other facility types. Finally, they calculated the Euclidean distance of each sub-subregion to the closest surrounding facilities. After these operations, the spatial skyline problem without candidates can transform into common skyline query problem.

Though the past work [15] can solve the area skyline problem, the complexity is very high, since the iterative computation of Voronoi Diagrams. In [6], Annisa et al. proposed another area skyline query algorithm, named *Grid-based Area Skyline* (GASKY), to reduce the algorithm complexity and processing time. Different to the UASKY algorithm, GASKY algorithm is based on grid partition. Specifically, they divided the region A into $n \times n$ grids, and the experiment results outperformed the UASKY algorithm. However, the reduced complexity and processing time cannot fit the real world very well. The processing time increased linearly with the size of facility types, the number of objects, and the number of grids increasing.

Our work is based on Annisa's work [6]. The referenced work for the area skyline query

relies on the centralized indexing structure. However, at the age of big data, the amount of data increase much fast. The centralized indexing structures cannot work well.

2.2.3 MapReduce Based Skyline Computation

With the development of technology of the Internet, the information shows exponential growth. At the age of big data, distributed computations received more attention to companies and researchers. Information retrieval is a vital part of big data processing. In this subsection, we review the distributed skyline computations for big data.

Hose et al. [16] survived the skyline processing problem in highly distributed environments. MapReduce is a useful programming technique of the Hadoop platform and an associated implementation for a massive volume of data processing. In recent years, the MapReduce technique is widely utilized for computing skyline queries [18, 17, 19, 20, 21]. Zhang et al. [17] proposed three distributed algorithms based on MapReduce, called MR-BNL, MR-SFS, and MR-Bitmap, to calculate the skyline queries. The Evaluation results illustrated that the proposed three algorithms could process the skyline query more effective than the conventional skyline algorithms. Chen et al. [18] also worked on the computation of the skyline query by using the MapReduce framework. In the paper, they partitioned the data space by using an angular partitioning, and the proposed model significantly reduced the processing time. Papadias et al. [19] worked on k-dominant skyline query processing by developing a MapReduce-based algorithm. Wu et al. [20] discovered the skyline points progressively based on the grid structure. The proposed distributed skyline query algorithm, called DSL, can improve the performance significantly. In [21], Wang et al. adapted skyline computation to the MapReduce framework. Similarly, they splited the data dimensions into several subpartitions iteratively by a grid-based scheme.

All the proposed algorithms of the above literature worked on conventional skyline query processing. Unfortunately, few works implemented the spatial skyline queries by

using the distributed techniques. In this work, the author aims to apply the distributed platform to compute the area skyline queries. Specifically, the author proposes a novel MapReduce based algorithm computation for selecting good locations on a map.

2.3 MapReduce-based Area Skyline

In this section, the author proposes a novel distributed algorithm based on MapReduce framework, called "MRGASKY", to improve performance of the computation of area skyline query.

For simpolicity, we assume the rectangular target area A is a square region (see Figure 2.3).. Firstly, we divide the region A into $n \times n$ grids on average by using a grid structure. Then, to identify the different grids, we assign IDs to all the grids. Specifically, we assign IDs $g_{1,1}$ to $g_{n,n}$ to the grids from the bottom-left to the top-right of A. Next, for grids in the same row, we calculate the distance from each grid to each type of facility in the Map function. Finally, for the grids in each column, we calculate the Euclidean distance from each grid to the closest facility for each type in the Reduce function. The results of the MapReduce operation are saved to the table like 2.2 shown. Using such a table, we can transform the area skyline query to the conventional skyline query problem, and we can retrieve the non-dominated grids.

To simplify the distributed area skyline problem, we assume that the distance between two grids approximate to be the distance of the center points of the two grids. When the length of the grid is small enough, the approximation is approximately equal to the exact distance. Figure 2.4 shows an example of the simplification. We define the four vertexes of a grid as a, b, c and d. Then ,the distance of a vetex d to the facility f can be represented by $dist(d, f)$. And star mark f is a preferable facility. The previous model in [6] define the minmum distance and the maximum distance of a grid from to a facility point f is $dist(d, f)$ and $dist(a, f)$. In our work, we simplify the model and define the distance of the

grid to the facility f as $dis(g, h)$, where the points g and f are the centers of these two grids.

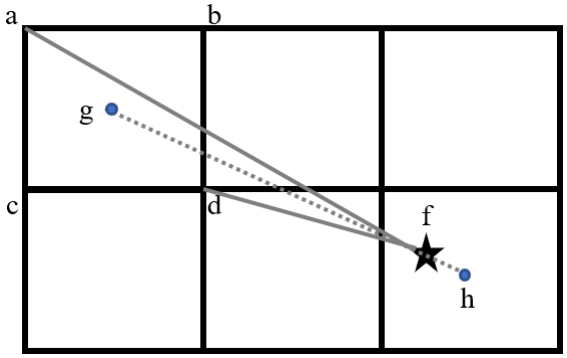

Figure 2.4: An Example of the Simplification.

2.3.1 MRGASKY Algorithm

The author explains the details of the MRGASKY algorithm as follows.

Step1 We partition the map row by row. And in each row, we calculate the distance to the closest facility of each type in the Map function.

Step1.1 Specifically, we divide a map into n rows. In the same row, the Map function reads all the grids from left to right and right to left, respectively. Then the Map function calculates the distance from every grid to the facilities. Initially, we assume the values of all the grids are infinite. When a facility is encountered, we set the value of the grid as zero. Then, the adjacent next grid is updated based on the former grid plus one, until the next facility is encountered.

Figure 2.5 demonstrates the calculation process of the 7-th row of the map shwon in Figure 2.3. The shaded grids are the preferable facilities of type $F1$. According to the above algorithm, we set the values of the shaded grids as zero. By reading the grids from the left to right side, we update the value of 5-th, 6-th, and 8-th grids to 1, 2, and 1. Similarly, we update the values of the 6-th, 5-th, 3rd, 2nd, and the first grids to 1, 2, 1, 2,

and 3.

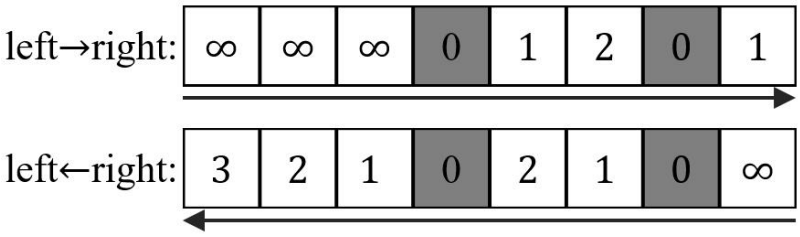

Figure 2.5: An Example of Step 1.1.

Step1.2 After the calculation from both sides, we choose the minimum value of every grid as a result. Then we return the results from the Map function.

Figure 2.6 illustrate the process of step 1.2. For example, after calculating, the values of the first grid are ∞ and 3. The minimum of this grid is 3. Similarly, the values of all the grids are updated to 3, 2, 1, 0, 1, 1, 0, and 1. Notice that the Map function calculates all the rows of a map.

Figure 2.6: An Example of Step 1.2.

The whole algorithm of the Map function is illustrated in Algorithm 1. The information of the map is saved into the Hadoop Distributed File System (HDFS). The map is formed as a binary image. That means, for the binary image of size $n \times n$, $m_{i,j}(1 \leq i, j \leq n)$ is an array, which includes every type of facilities in the grid of ith row and jth column. $m_{i,j}[k](k \leq m)$ is the kth element of array $m_{i,j}$, where m is the total number of facility types. $m_{i,j}[k] = 1$ represents the facility of kth type is inside the grid $g_{i,j}$ and $m_{i,j}[k] = 0$ represents that there are no facilities of kth facility type inside grid $g_{i,j}$.

Usually, the output format of the Map function is the $key - value$ pairs. The key is the location of the grid $g_{i,j}$. In other words, the key represents the column ID and row ID as j and i respectively for the grid $g_{i,j}$. The $value$ represents the distance, which is calculated

in step 1.

Algorithm 1 Map Function for Step 1 Process

Require: A binary image

Ensure: (key, value)=(type x-coordinate of grids. y-coordinate of grids, distance)

1: **for** each line in HDFS **do**
2: calculate the Euclidean Distance from left to right: $dist_{left \rightarrow right}$
3: calculate the Euclidean Distance from right to left: $dist_{left \leftarrow right}$
4: **end for**
5: **for** each grid in the same row **do**
6: $dist_{min} = min(dist_{left \rightarrow right}, dist_{left \leftarrow right})$
7: **end for**

∞	∞	∞	∞	∞	∞	∞	∞
3	2	1	0★	1	1	0★	1
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
3	2	1	0★	1	2	3	4
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞

Figure 2.7: An Example of Step 1.

Figure 2.7 demonstrates the results after step 1 process of facility type $F1^+$. Since there are only three facilities of $F1^+$ on the map, we only update the values of the 3rd row and 7-th row. Other values in other rows are not updated.

Step2 In the Reduce function, we calculate the distance of each grid to the closest facility by column-wise based on the results of Map operation.

Step2.1 In step 1, the $key-value$ pairs are the row numbers and column numbers. In this

step, we sort and shuffle the $key-value$ pairs by columns. First, the Reduce function reads the $key-value$ pairs in a parallel way. For example, for the i-th column, we save the values into a stack from the bottom side to the upside. Notice that there exists a corresponding stack for each column. The sorted and shuffled values are saved to the corresponding stack. Then, we map the values in the stack of the same column into a two-dimensional coordinate. The x-axis represents the column IDs of the grids, while the y-axis represents the $values$ of the stack. We name all these n points as $p_1, p_2, ..., p_n$, and bisect the two adjacent points $p_i = (x_i, y_i)$, and $p_j = (x_j, y_j)$, where $(1 \leq i < j \leq n)$. We represent the intersection of the vertical bisector $\overline{p_i p_j}$ and x-axis as x_{ij}. Then the function of x_{ij} can be calculated as follows.

$$x_{ij} = \frac{(y_j^2 - y_i^2) + (x_j^2 - x_i^2)}{2(x_j - x_i)} \tag{2.1}$$

For example, there are three adjacent points p_i, p_j, p_k and $i < j < k$. If and only if $x_{ij} > x_{jk}$, we delete the point p_j from the stack. Otherwise, we retain the point p_j in the stack. To better understand the algorithm in step 2, we illustrate the two cases in Figure 2.8 and Figure 2.9, whether deleting the point p_j or not.

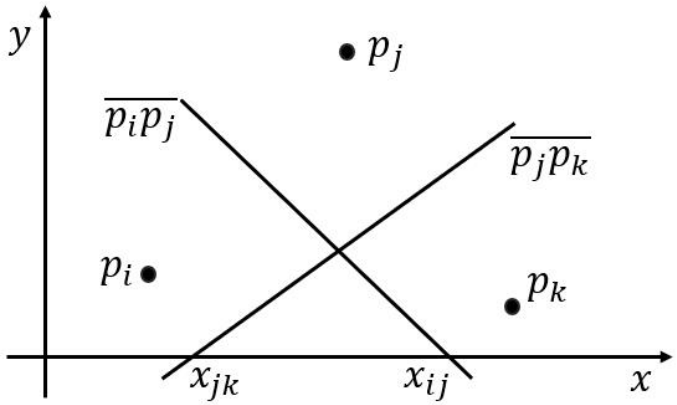

Figure 2.8: An Example of Deleting Point p_j.

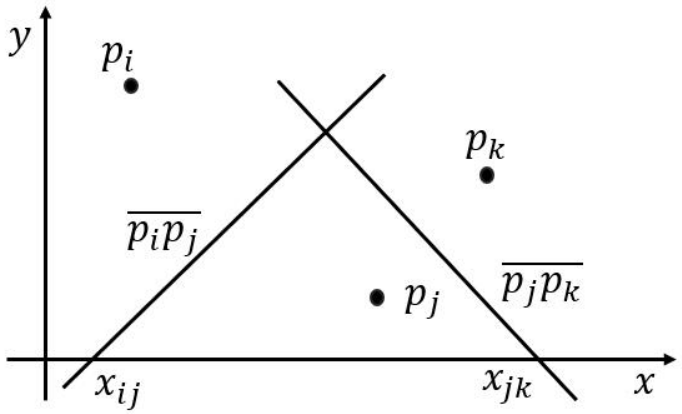

Figure 2.9: An Example of Maintaining Point p_j.

In Figure 2.8, since $x_{ij} > x_{jk}$, we delete the point p_j from the stack. Similarly, in Figure 2.9, since $x_{ij} < x_{jk}$, we save the point p_j into the stack.

Figure 2.10 shows the process of step 2.1 of the 5-th column of Figure 2.7. Firstly, we compare x_{12} and x_{23}. Since $x_{12} > x_{23}$, we delete the point p_2 from the stack. Then, the stack is updated according to the previous operation. Again, we compare the values of x_{13} and x_{34}. Since $x_{13} < x_{34}$, we retain the point p_3 into the stack. Update the stack until all the points are retrieved, we return the results of the stack. In this example, the points p_1, p_3 and p_7 are retained into stack; while the points p_2, p_4, p_5, p_6 and p_8 are removed from the stack.

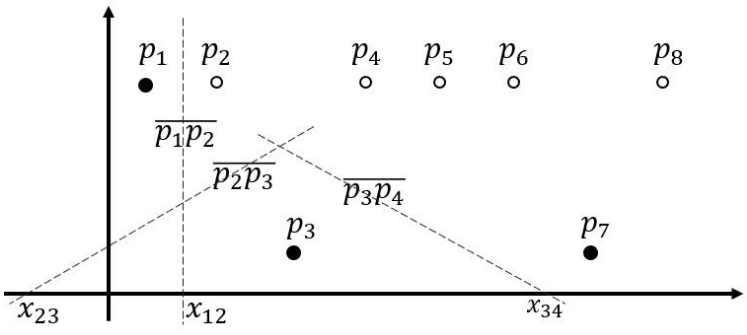

Figure 2.10: An Example of Step 2.1.

Step2.2 For each column on the map, we determine the proximate intervals [22] based on

the left points. First, we bisect the adjacent left points on x-axis. According to x_{ij} and x_{jk} $(i < j < k)$, we can determine the dominated area of all the left points, i.e., the dominated area of the point p_j is $[x_{ij}, x_{jk}]$. Then, according to the dominated areas, we determine the dominated points for the left points from the deleting points of step 2.1. Specifically, if the column IDs of the deleting points in the interval $[x_{ij}, x_{jk}]$, we say that the deleting point is dominated by the point p_j. Finally, we compute the Euclidean distance of this deleting point by using the *value* of the point p_j.

For example, Figure 2.11 demonstrates the process of step 2.2. In the figure, p_1, p_3 and p_7 are the results of step 2.1. In other words, these three points are the retained points. First, we bisect the adjacent two points of p_1, p_3 and p_3, p_7, to calculate x_{ij}. Since $x_{13} < 0$ and $x_{37} = 5$, we set the dominated intervals of the points p_3, p_7 are $[x_{13}, x_{37}]$ and $[x_{37}, 8]$ respectively. In other words, the point p_3 dominates four deleting points p_1, p_2, p_4 and p_5. The point p_7 dominates two deleting points p_6 and p_8. Then the Euclidean distance of all the grids in the same column can be calculated in an obvious way.

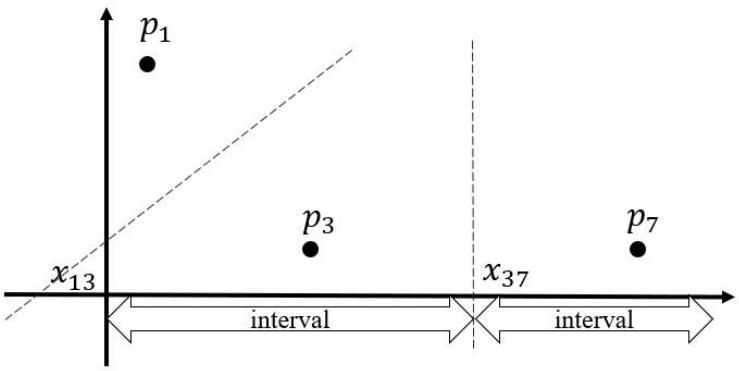

Figure 2.11: An Example of Step 2.2.

Algorithm 2 demonstrates the whole process of the Reduce function. The input of this phase is $key - value$ pairs. The *keys* and *values* are the column IDs and the output distance from the Map function. The outputs of the Reduce function are the Euclidean

distances of all the grids.

Algorithm 2 Reduce Function for Step 2 Process

Require: The results of the sorted and shuffle phase

Ensure: Area skyline objects

1: **for** each point sorted by x-coordinate in the same column **do**
2: **if** $x_{ij} > x_{jk}$ **then**
3: pop the point p_j from the stack
4: **end if**
5: **end for**
6: **for** all un-popped points which are sorted by x-coordinate **do**
7: calculate the proximate interval of each point on x-axis
8: calculate the Euclidean Distance of the points in the interval
9: **end for**
10: //remove the dominated grids
11: **for** the record r_i of the Euclidean Distance for all types of each grid **do**
12: **if** $r_i < r_j$ **then**
13: remove the dominated grid from the record
14: **end if**
15: **end for**
16: return skyline objects

After this operation, we can transform the problem of area skyline processing to a distance table of all the grids. It is a common skyline processing problem, and we can calculate the area skyline objects easily. It means the area skyline objects are the shaded grids such as Figure 2.3 shown.

Figure 2.12 shows an example results of step 2 of the type $F1^+$. Notice that other facility types, such as $F2^+$ and $F3^-$ are also calculated in the MapReduce framework.

Intuitively, the proposed MRGASKY algorithm is the same process to the Voronoi Diagram. However, the proposed MRGASKY algorithm can significantly reduce the complexity of the Voronoi Diagram in the MapReduce framework. Moreover, the average

processing time can also be reduced to a constant.

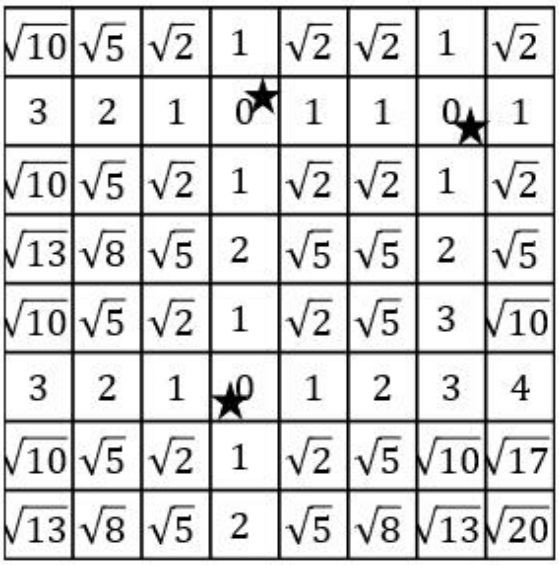

Figure 2.12: Example of the Step 2 Process.

To better understand the proposed algorithm, the author demonstrates the whole data flow of the MapReduce framework of facility type $F1^+$ and $F2^+$ in Figure 2.13.

The input of the MapReduce framework is the binary image, which saves all information on a map. The information is stored in HDFS. Specifically, for each row of the input is formed as the facility type F, row IDs n, and the values of the binary image. Then, we partition the HDFS row by row and send the separated row data into Map function. In the Map function, data are processed by the format of $key - value$ pairs. After the Map operation, the $key - value$ pairs are regenerated. According to the $intermediatekeys$, the new intermediate $key - value$ pairs are grouped and sorted. After the sorted and shuffle phase, the grids are grouped by column-wise. The grouped column-wise data are regarded as the input of the Reduce function. Then the Reduce function calculates the Euclidean distance of each grid by column-wise, and return the final results to the HDFS again.

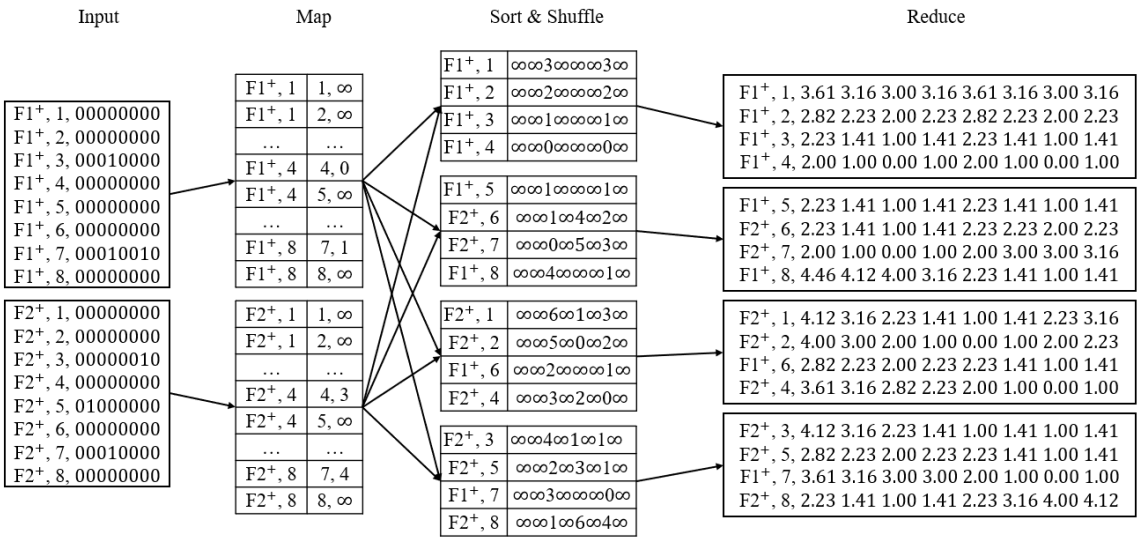

Figure 2.13: MapReduce Data Flow of MRGASKY Algorithm.

2.4 Experimental Evaluation

In this section, we conduct two experiments to evaluate the average processing time of the proposed MRGASKY algorithm. We implement all these models by using Python 3.5.2. We compare the performance of the proposed MRGASKY algorithm with the GASKY algorithm. GASKY algorithm is conducted on the Linux operating system, and the configuration of the CPU is Intel Core i7 3.40GHz processor. The size of the main memory is 4GB.

For the experiments of the proposed MRGASKY algorithm, we implement it on Hadoop 2.5.2 version. We set four computing nodes for the MapReduce framework. All the four computing nodes are conducted on the Linux operating systems, with 4GB main memories. The CPU of one of the four computing nodes are Intel Core i7 3.40GHz processor. The retaining three computing nodes are Intel Core 2 3.16GHz, 3.16GHz, and 2.13GHz processors.

For the two experiments, we use two kinds of data, synthetic dataset and real dataset respectively. The main description of the datasets is explained as follows.

Synthetic dataset: We synthesize a dataset to learn the scalability of the proposed algorithm. We denote the synthetic dataset as SYN. Specifically, we randomly generate the two-dimensional data to represent the location of the facilities. We represent the number of grids by n^2, the number of facility types by m, and the number of objects by obj, to evaluate the performance. We name these four experiments as SYN_A1, SYN_A2, SYN_B, and SYN_C, to represent the varying of the number of grids, the number of facility types and the number of objects, respectively.

Real dataset: We conduct the second experiment by using the real dataset, the U.S. Geological Survey (USGS), which we call it "US". In the US dataset, there records 2,033,545 objects at total 406,709 locations. There is a total of 40 facility types for all the objects. We represent the experiments by US_A, US_B, and US_C, respectively, where the experiment US_A is varying the number of grids n^2, the experiment US_B is varying the number of facility types m, and US_C is varying the number of objects obj, respectively.

Since GASKY is the first algorithm for area skyline query, we can only compare the proposed distributed algorithm MRGASKY with GASKY. Furthermore, in these two kinds of experiments, we select the average processing time as the indicator to evaluate the performance. All the two algorithms aim to generate a distance table to transform the area skyline problem into the skyline query problem. Since the skyline query process is the same process to remove the dominated locations from the dataset for the two algorithms, and no difference in performance evaluation, we remove the computation of this part.

2.4.1 Efficiency of Synthetic Dataset

We first investigate the efficiency of the synthetic dataset in this subsection.

Effect on grid number: In these experiments, we conduct two sets of experiment

SYN_A1 and SYN_A2 to investigate the effect of grids. The number of objects of the SYN_A1 dataset is $obj = 32$. We fix the facility type as $m = 4$. It means we select four types of facilities on the virtual map. Both the number of preferable facility types and the unpreferable facility types are two. We vary the number of grids n^2 with 32×32, 64×64, 128×128, 512×512 and 1024×1024 respectively.

For the dataset SYN_A2, the number of objects is larger than the dataset SYN_A1. We set the number of objects to $obj = 2000$. We fix the facility types as two,, which a kind for the preferable facility, and a kind for the unpreferable facility. We vary the number of grids n^2 with 100×100, 500×500 and 1000×1000.

Figures 2.14 and 2.15 demonstrate the results of effect on the number of grids. In Figure 2.14, the average processing time of both the GASKY algorithm and the MRGASKY increase with the number of grids increasing. When $n^2 < 256 \times 256$, the GASKY algorithm seems to work better than the MRGASKY algorithm. Because the number of grids n^2, facility types $m = 4$, and the number of objects $obj = 32$ are small enough, the GASKY algorithm on the centralized indexing structure can work well. However, the average processing time of the GASKY algorithm increases much faster than the MRGASKY algorithm when $n^2 > 256 \times 256$. It means that with the data are big enough, the distributed algorithm on the MapReduce can handle the big data better than the centralized indexing structure.

To better analyze the effect on the number of grids, we experiment with the dataset SYN_A2. For the dataset SYN_A2, we enlarge the amount of data. The results show in Figure 2.15. The average processing time of the GASKY algorithm increases faster than the MRGASKY algorithm at the beginning. The increase seems to be linear of the GASKY algorithm. Relatively, the MRGASKY algorithm has a steady increase with the changing of the grids. The previous paper [6] has discussed that the GASKY algorithm takes more time cost in the $min - max$ table computation and the Voronoi Diagram building. Therefore, the proposed MRGASKY algorithm has better scalability of the effect on the number of

grids.

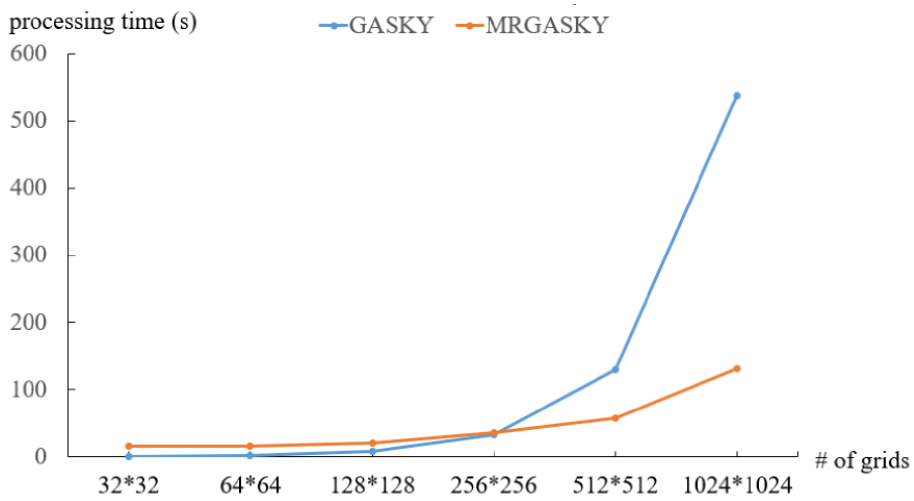

Figure 2.14: Processing Time of SYN_A1.

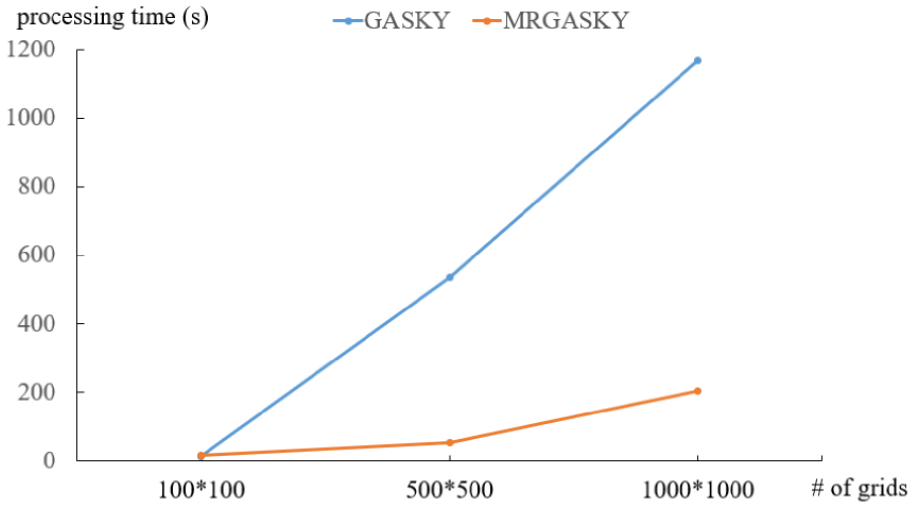

Figure 2.15: Processing Time of SYN_A2.

Effect on facility types: For the dataset SYN_B, we explore the changing of the average processing time of the effect on facility types. We fix the number of objects as $obj = 10,000$. Also, the number of grids are set to $n^2 = 128 \times 128$. We vary the number of facility types with 2, 4, 6, and 8, respectively. Notice that the number of facility types consists the preferable facilities and the unpreferable facilities with the same size. For example, when the facility type $m = 4$, both of the number of preferable facility types and

the number of unpreferable facility types are two.

We show the results of the effect on the number of facility types in Figure 2.16. It is easily observed that the average processing time of the GASKY algorithm is larger than the MRGASKY algorithm at the beginning in the figure. Furthermore, the average processing time of the GASKY algorithm increases linearly with the changing of m, while the MRGASKY algorithm is much smaller and tends to be stable at 40[*sec.*].

From the experiment SYN_B, we can say that our proposed MRGASKY algorithm outperforms the GASKY algorithm on the effect of facility types. Also, the MRGAKSY algorithm has good scalability and can handle big data better than the baseline.

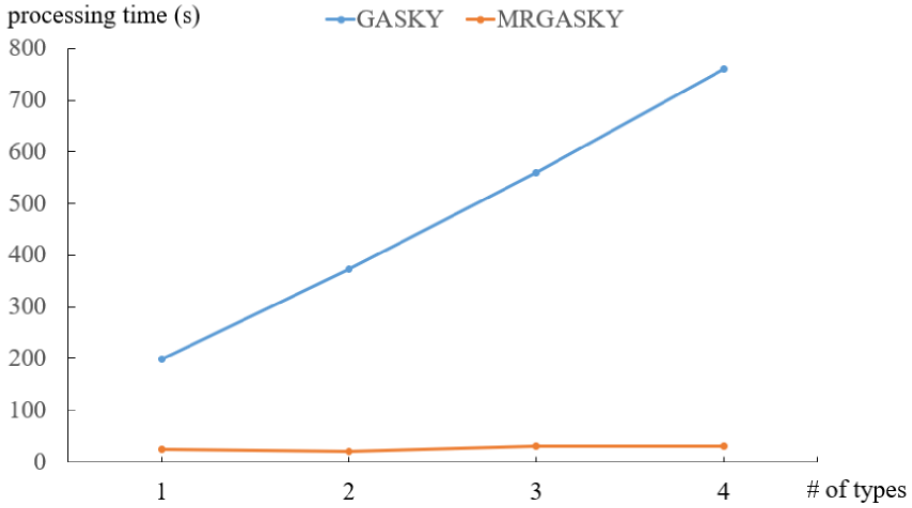

Figure 2.16: Processing Time of SYN_B.

Effect on object number: For the dataset SYN_C, we explore the changing of the average processing time of the effect on the number of objects. In this experiment, we fix the number of facility types as $m = 2$. Also, we set the number of grids to $n^2 = 128 \times 128$. We vary the number of objects with the size 4000, 8000, 12,000 and 16,000, respectively.

Figure 2.17 demonstrates the results of dataset SYN_C. We can observe that the average processing time of the GASKY algorithm is larger than the MRGASKY algorithm at the beginning in the figure with the changing of the number of objects. Furthermore, the

average processing time of the GASKY algorithm increases linearly with the changing of m, while the MRGASKY algorithm is much smaller and tends to be stable at 20[$sec.$].

From the experiment SYN_C, we can say that our proposed MRGASKY algorithm outperforms the GASKY algorithm on the effect of objects. Overall, the MRGASKY algorithm maintains sufficient stability to handle "big data."

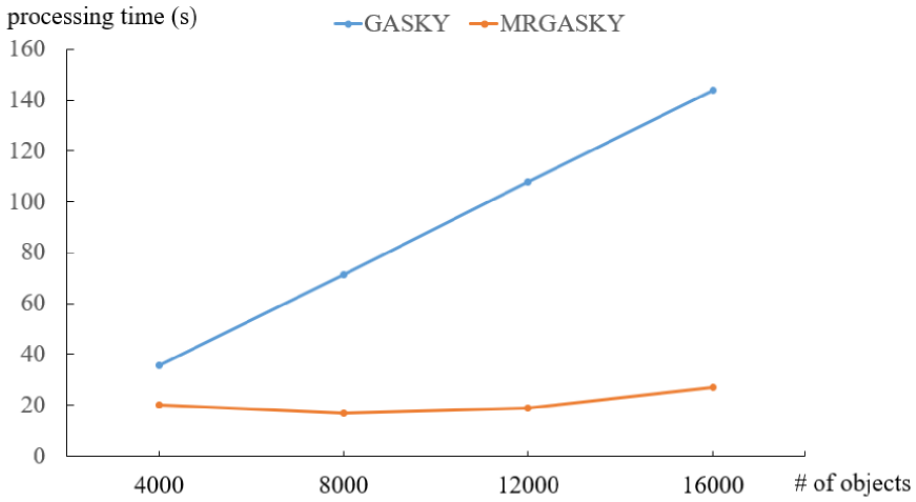

Figure 2.17: Processing Time of SYN_C.

2.4.2 Efficiency of Real Dataset

In this subsection, we conduct several experiments, which explore the effect of the average processing time. We vary the number of grids, the number of facility types, and the number of objects, in the real dataset, respectively.

Effect of grids: In the dataset US_A, we mainly analyze the effect on the number of grids. We fix the number of facility types as $m = 2$. Notice that both of the excellent and unpreferable facility types are the same, and the values are 1. We set the number of objects to $obj = 1000$. Then, we change the number of grids with the values are $n^2 = 200 \times 200, 300 \times 300, 400 \times 400, 500 \times 500$, and 600×600, respectively.

Figure 2.18 demonstrates the effect on the number of grids of the real dataset US_A. We can easily observe that the average processing time of the GASKY algorithm is larger

than the MRGASKY algorithm at the beginning in the figure. Furthermore, the average processing time of the GASKY algorithm increases linearly with the changing of n^2, while the MRGASKY algorithm is much smaller and tends to be stable at 40[*sec.*].

From the real dataset of the US_A, we can say that our proposed MRGASKY algorithm outperforms the GASKY algorithm on the effect of grids.

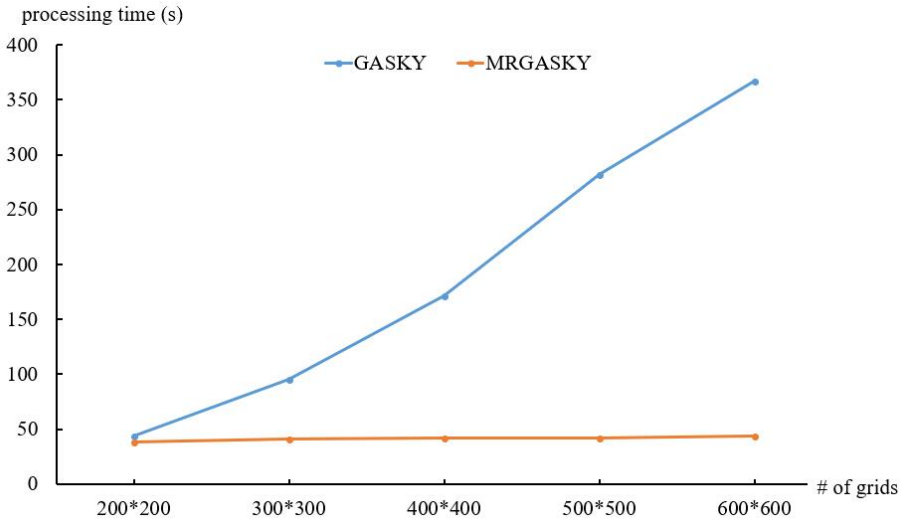

Figure 2.18: Processing Time of US_A.

Effect of facility types: In the dataset US_B, we mainly analyze the effect on the number of facility types. We fix the number of grids as $n^2 = 200 \times 200$. Also, we set the number of objects to $obj = 1000$. Then, we change the number of facility types with the values is $m = 2, 4, 6$, and 8, respectively.

Figure 2.19 demonstrates the effect on the number of grids of the real dataset US_B. We can easily observe that the average processing time of the GASKY algorithm is larger than the MRGASKY algorithm at the beginning in the figure. Furthermore, the average processing time of the GASKY algorithm increases linearly with the changing of m, while the MRGASKY algorithm is much smaller and tends to be stable under 50[*sec.*].

From the real dataset of the US_B, we can say that our proposed MRGASKY algorithm outperforms the GASKY algorithm on the effect of facility types.

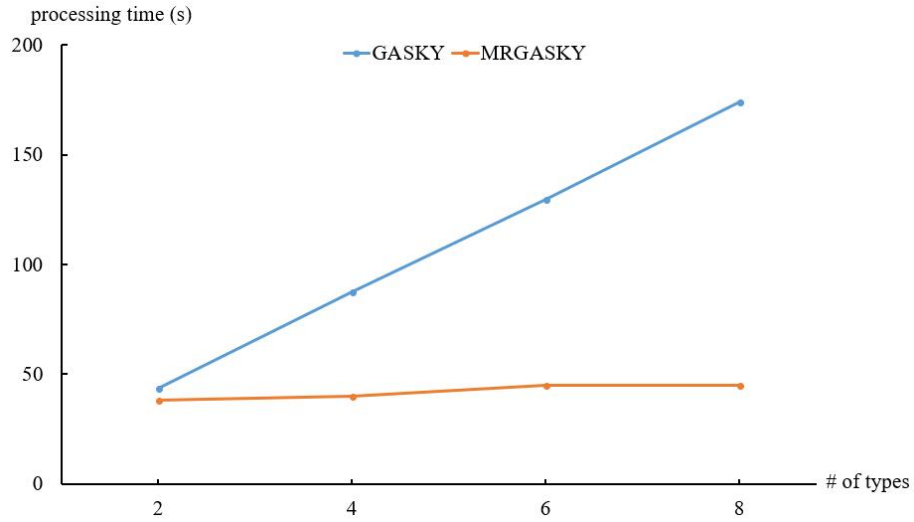

Figure 2.19: Processing Time of US_B.

Effect of objects: In the dataset US_C, we discuss the effect on the number of objects. We fix the number of grids as $n^2 = 200 \times 200$. Also, we set the number of facility types to $obj = 2$, and both of the preferable facility type and the unpreferable facility type are 1. Then, we change the number of objects with the values is $m = 2000, 4000, 6000, 8000$ and $10,000$, respectively.

Figure 2.20 demonstrates the effect on the number of objects of the real dataset US_C. We can observe that the average processing time of the GASKY algorithm is larger than the MRGASKY algorithm. Furthermore, the average processing time of the GASKY algorithm increases linearly with the changing of obj, while the MRGASKY algorithm is much smaller and tends to be stable under $50[sec.]$.

From the real dataset of the US_C, we can say that our proposed MRGASKY algorithm outperforms the GASKY algorithm on the effect of objects.

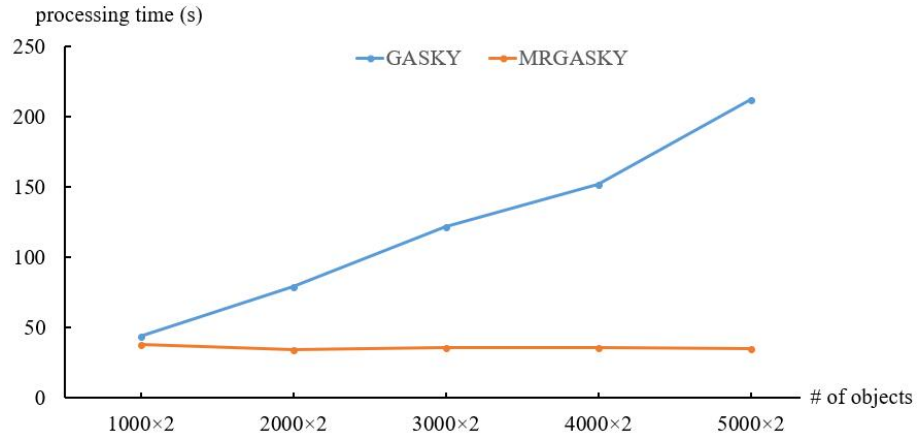

Figure 2.20: Processing Time of US_C.

Overall, our proposed MRGASKY algorithm has a significant improvement on performance comparing to the previous GASKY algorithm of the area skyline query problem for location recommendation on a map. In other words, the MRGASKY algorithm can handle "big data" well.

2.5 Concluding Remarks

Location-based recommendation systems are essential for mobile applications. Many map-based applications can filter the uninteresting locations for mobile users, and recommend a small number of areas for users. In general, an interesting or a good location should always be surrounded by preferable facilities, and should also be far away from unpreferable facilities. In the previous work [6], Annisa et al. has proposed area skyline queries based on a grid structure, called GASKY, to retrieve good locations on a map. However, the complexity and the average processing time of the GASKY algorithm is very large, since the Voronoi Diagram and $max - min$ table computation cost a lot.

In this work, we proposed a novel distributed area skyline query, which is based on the MapReduce framework. The novel algorithm, called MRGASKY, aims to improve the performance, i.e., the average processing time, comparing to the GASKY algorithm.

To confirm the effectiveness and efficiency of the MRGASKY algorithm, we conducted experiments. The experimental results illustrated that the average processing time of the MRGASKY algorithm tends to be stable with the number of grids, the number of facility types, and the number of objects increases. Overall, the proposed MRGASKY algorithm can handle the "big data" better and more effective than the GASKY algorithm.

For applications, the MRGASKY algorithm can be utilized in some specific scenarios as follows.

- In the travel filed: The utilization of map-based applications on mobile devices is very common for travelers during a trip. In general, travelers would like to travel to places which have convenient transportations and are close to famous sightseeing spots. Also, the places should be far from the pollution areas and wildernesses. In other words, a good location should be close to preferable facilities and be far from unpreferable facilities according to user preferences. The proposed MRGASKY algorithm can recommend such good locations to the tourists of mobile devices in a short time.

- In the business field: It is a typical case for people in business, who want to find some good locations for the company building. For example, a real estate developer would like to search for an area to build a community. In this case, the great area should be close to some convenient places, such as bus/train stations, malls, and schools. Moreover, the desirable area should also be far away from some unpopular places, such as pollution areas, wildnesses, and noisy factories. The proposed distributed MRGASKY algorithm can also be applied into the business field, to help such the real estate developers find the potential areas on a map. Also, the MRGASKY algorithm can help the company reduce the survey cost in general.

In the future, we will take some non-spatial properties, such as the prices of the areas,

the population densities of the regions into account. Also, we will consider the $k-$dominant problem, since the dimensions of data are more and more sophisticated in the age of "big data." Moreover, we would like to apply our algorithm in the map applications to help more people to make location recommendations.

Chapter 3

Parameter Estimation of Queueing Systems with Utilization Data

3.1 Introduction

3.1.1 Performance Evaluation

Performance evaluation plays a vital subject in many fields, such as the performance evaluation of traffic, the performance evaluation of computer systems, etc. For example, the performance evaluation of traffic can alleviate traffic pressure effectively, and help cities to carry out road planning and construction. Also, the performance evaluation of computer systems can help the designers determine the system configuration in the system design phase. For example, We can determine the number of CPU from the performance evaluation of computation cost. From the evaluation of storage, we can determine the memory size. However, with the advancement of computer technology, a tremendous amount of new data is generated every day. The improvement the computing power and computing capacity of computers are not economical and unrealistic sometimes. As a way to solve the problem, the recent design of system architecture tends to combine existing systems

as models, which is called the system of systems (SoS) [23]. The integration of SoS can effectively improve the utilization of computer systems, and can also reduce economic consumption. So, performance evaluation of the integrated systems, such as SoS, becomes more and more critical in recent years.

In general, there are three main performance evaluation methods for computer systems, measurement-based performance evaluation, simulation-based performance evaluation, and model-based performance evaluation, respectively. We usually use a measurement-based approach to evaluate the performance of existing systems. A simulation-based approach is more flexible and accurate for the systems under designing. However, the simulation-based method takes too much time in deriving models. The model-based approach is the most commonly used approach of performance evaluation, which can build a mathematical model, and analyze the model quantitatively.

3.1.2 Queueing Systems

Queuing-based model is the typical method for model-based performance evaluation. Statistical inference in queueing theory [24, 25, 26, 27, 28] has attracted many researchers in the past decades.

In general, there are several main components for a queueing system, the input, the queue, and the server, respectively. In a computer system, the input is a flow of jobs or tasks. The input jobs wait for service in the queue. The jobs or tasks leave the queueing system after service. In general, we represent a queueing system by using the notation $A/S/m/k$. Here, the symbol A represents the type of an arrival process, S represents the types of service distribution, m represents the number of servers, and k is the capacity of a queue. For example, the notation $M/M/1/k$ represents a typical queueing system. Jobs or tasks arrive at the queueing system following Poisson distribution, and service times of these jobs are following an exponential distribution. There is only one server in this

queueing system with a capacity of k.

The problem of estimation concerning the parameters of queueing systems such as arrival rate and service rate is an essential thing in the queueing systems [24, 29, 30, 31, 32, 33]. Thiruvaiyaru et al. [24] estimated the parameters of arrival and service rates. In this work, they assumed that the system is continuously observable over a fixed interval for inter-arrival times and service times as an open Jackson network. Clarke et al. [31] generated a $M/M/1$ queueing system, and estimated the Poisson arrival rate and exponential service distribution by collect the arrival times and departure times in a fixed time interval. Thiruvaiyaru et al. [32] generated an empirical Bayes model for a $M/M/1$ single queue and estimated the Poisson arrival rate and exponential service rate after the collection of data of observation. Basawa et al. [33] collected waiting times from n successive customers, and estimated the parameters of a $GI/G/1$ queueing system.

Most of the above literature [24, 31, 32] were interested in the experimental results that occur randomly for the counts of events within intervals of time. The Poisson distribution is a typical discrete distribution, which is usually used into the statistical inference of queueing theory. The Poisson distribution can observe the counts of the event in a given time interval. In general, the parameter of the Poisson distribution is the mean number of events per time interval. In other words, the arrival rate of the Poisson distribution is constant. However, for most the real cases in the world, the arrival processes are always changing over time. Such a constant assumption for the arrival rate cannot work well in the above cases. For example, the arrival process is always varying dynamically of the traffic. The arrival rate of transportations in the morning rush hour or evening rush hour is much higher than other time. Furthermore, the arrival rate at morning rush hour should also be different from evening rush hours. Also, the arrival rates of customers of a convenient store at different periods should be changed. The customers' arrival rate at mealtime should be much higher than other time, and the arrival rate at midnight should be the smallest.

3.1.3 Non-homogeneous Poisson Process

To generate a better model to satisfy the real-world situation, we use the dynamical changing arrival rate to model the arrival process, which we call Non-homogeneous Poisson Process (NHPP). The NHPP has already successfully been implemented to model the complex independent arrival processes in many past works [34, 35, 36]. An NHPP is a generalization of an ordinary Poisson process where events occur overtime randomly at a rate of λ events per unit time. In general, we define the rate of an NHPP, which varies over time, by the intensity function $\lambda(t)$. We can find that the intensity function is a function of arrival rate, which is respect with time t. In this work, we assume that the jobs or tasks of computer systems arrive at the queueing systems following an NHPP, i.e., $M_t/M/1/k$. It means that the queueing system has an NHPP arrival M_t, exponential service time M with one single server and queue capacity k.

On the other hand, all the queueing systems of the above literature are observable. In other words, we can collect the arrival times and the number of arrivals directly in a time interval. However, in some unobservable systems, we cannot estimate parameters by directly observing the arrival times and the number of arrivals. For example, in a computer system, CPU utilization is a widespread data type and can reflect the usage of the computing resource. CPU utilization data can be collected easily from a computer system. However, we cannot know the information on the arrival time and the service time of CPU-tasks from CPU utilization data directly. The observations of the arrival and service for the estimation are quite limited in this case.

In this work, we focus on generating a novel model to estimate parameters with utilization data. Utilization data are a kind of time-series data. Utilization data refer to the usage of processing resources or the amount of work handled by a computing node. In general, we define the utilization data as the time fractions of busy periods in fixed time intervals. Utilization data is parctically used to reprensent server conditions, such as CPU

utilization data, health care utilization data, etc. The definition of utilization data shows one of the properties that the data collection process of utilization data allocates in a series of discrete-time slices. Besides, there exists another slice adjacent to the collection slice, where we cannot monitor and collect information. Utilization data is the only data we can get in computer systems, and the estimation procedure should be developed.

The main contributions of this work are as follows:

1. We mainly estimate the arrival process of a queueing system from utilization data. To the best of our knowledge, this is the first paper to estimate the arrival process by using utilization data. In general, utilization data is defined as a fraction of a busy period in a fixed time interval. We cannot get exact arrival time and service time from utilization data. It is challenging for us to generate a novel model to estimate parameters from utilization data.

2. To better model the arrival process in the real world, we consider Non-homogeneous Poisson process as the input. The queueing model is noted as $M_t/M/1/K$. Since the intensity function of the NHPP is a function of rates, which is respected with time $\lambda(t)$, the NHPP is more realistic and complex than the conventional Poisson process. To simplify the problem, we make an approximation of the NHPP and transform the dynamical complex arrival process to a series of Homogeneous Poisson Process (HPP).

3. We aim to estimate the arrival process of the $M_t/M/1/K$ queueing system from incomplete utilization data. To make accurate estimations, we propose to apply the Maximum Likelihood Estimation (MLE) via the Expectation-Maximization (EM) algorithm.

4. In the experiments, we conduct an extensive performance evaluation based on simulated utilization data and real CPU utilization data respectively to verify the proposed

model.

The rest of the paper is organized into several sections. **Section 3.2** reviews the related works. **Section 3.3** shows the definitions in detail of the $M_t/M/1/K$ queueing systems. In **Section 3.4**, the author demonstrates the estimation approaches, such as the MLE method and the EM algorithm to estimate parameters. Next, in **Section 3.5**, the author conducts some experiments with the simulated utilization data and the real CPU utilization data to estimate the intensity function of an NHPP arrival process. Besides, the author demonstrates the response time of the queueing system to verify the proposed model. Finally, in **Section 3.6**, the author concludes this work.

3.2 Related Works

3.2.1 Queueing Systems

Statistical inference of queueing systems has attracted many researchers in recent decades [24, 25, 34, 37, 38]. A queueing system can model many processes. We can make the analogy to the queues we encounter in our daily life, such as patients waiting for lines in a hospital [39], phone call systems [40], and shipping container in a seaport [41]. In general, there are three typical components for a queuing system, arrival process, service process, and a queue, respectively. It is vital for a queueing system to estimate the parameters such as the arrival rate and service rate. [24] consider the case that travelers wait for service, as the objective queueing system. To simplify the queueing system, the authors proposed a simple $M/M/1$ queueing model. In the queueing system, customers arrive at the queue following Poisson distribution. The service time follows an exponential distribution with a single server. Basawa et al. [25] proposed two specific queues, named $M/M/1$ and $M/E_k/1$. $M/M/1$ is a typical and simplest queueing system. Different to the previous queueing system, in the queueing system of $M/E_k/1$, the author model the service time

followed Erlang distribution. They estimated parameters, i.e., the mean number of events per time interval λ, from waiting for time data of n successive customers. Since waiting time data are partial data, it is much simpler than inter-arrival times and service times. Fischer et al. [34] generated a $M/G/1$ queueing model, where the service time follows gamma distribution from waiting time data. They proposed an approximation approach to simplify the waiting time distribution. Specifically, they used the Laplace transform to transform the waiting time distribution instead of the waiting time distribution itself. In [37], Ross et al. proposed to estimate parameters of a $M/M/c$ queueing system with queue length data, which were collected by n successive time points. They generated density-dependent transition rates of a Markov process by taking the arrival rate to be of the same order as the number of servers. In [38], Liu et al. proposed a queueing system based on the queue length. They used the number of queueing vehicles to represent the queue length and used a real-time length estimation with probe vehicles' data to estimate parameters.

All the above papers considered the input follow a Homogeneous Poisson Process, which means the arrival rates λ represent the mean number of events per time interval. Since the Poisson distribution assumed that the arrival rate is a constant, it can only model some simple arrival process of customers or jobs of queueing systems. However, in real-world situations, the arrival processes are changing over time. In these cases, we consider an NHPP to represent the dynamical arrival process.

The intensity function $\lambda(t)$ of an NHPP is a function of arrival rate λ, which is respect with time t. The arrival process of such an NHPP is much complicated and challenging than an HPP. Rothkopf et al. [42] made an approximation for a queueing system with NHPP input. The proposed queueing system contained an arrival distribution with time-varying and service rates with a single server. Heyman et al. [43] generated a $M_t/G/c$ model, where the input followed the NHPP, the service distribution followed the gamma distribution with c servers. Green et al. [44] proposed a queueing model with the NHPP

input. Since the nonhomogeneous process rate was a function of time t, it is too challenging to estimate such time-continuous rate. They made an approximation to the non-stationary intensity function. In [45], Pant et al. modeled the waiting traffic system by the annotation $M_t/M/1$. On the formulating of the mathematical model, they got the mean waiting time of customers, mean time spent, the mean number of customers iteratively.

3.2.2 Maximum Likelihood Estimates

The Maximum Likelihood Estimates (MLE) and the Moment Estimate (ME) are two typical estimation methods for the arrival and service parameters of queueing models. In statistics, the MLE is a method of estimating the parameters by maximizing a likelihood function. The primary assumption of the MLE is that the observed data is most probable.

Clarke et al. [31] estimated the arrival and service parameters of a $M/M/1$ queueing model by using the MLE. The $M/M/1$ queueing model, proposed by Thiruvaiyaru et al. [32], have mentioned on the above, used an empirical Bayes approach to determine the arrival and service parameters. Benes [46] generated a queueing model for a telephone exchange system, which had an infinite number of trunks. The parameters were estimated from a standard queueing system by the ME. Wang et al. [47] considered a $M/M/R/N$ queue, which the arrival and service distribution were the Poisson and exponential distribution with multi-servers. They developed the confidence interval formula for the estimated results by using the MLE. In [48], Amit et al. estimated the NHPP intensity function of a traffic system by observing the number of customers from a $M_t/M/1$ queueing system.

Most of the above works need a complete collection of data in a fixed time. In practice, however, missing data or incomplete data always occur. Therefore, researchers usually use the Expectation-Maximization (EM) algorithm to process the statistical inference under missing data.

3.2.3 Expectation Maximization Algorithm

The EM algorithm is a technique that finds the MLE in parametric models for incomplete data. In general, given a set of incomplete data, the EM algorithm is an iterative procedure to find the MLE of the parameter vector by repeating the following steps.

1. **The expectation E-step:** Given a set of parameter estimates, the E-step calculates the conditional expectation of the complete data log-likelihood given the observed data and parameter estimates.

2. **The maximization M-step:** Given a complete data log-likelihood, the M-step aims to find the parameter estimates to maximize the estimates form E-step.

The two steps are iterated until the iterations coverage.

The EM algorithm is first proposed by Dempster et al. [49]. Based on this work, Wu [50] discussed the convergence of the EM algorithm. Wu et al. considered that an EM sequence coverages to a unique MLE with a unimodal and differentiable likelihood function. Besides, Wu found that the EM algorithm was powerful to process the multi-parameter stochastic model. Rydén [51] estimated parameters of a model, called Markov modulated Poisson process (MMPP) by applying an EM algorithm. In [52], Basawa et al. derived arrival parameters of a $GI/G/1$ queue by applying the EM algorithm with missing data. In this work, Basawa collected partial waiting times as the observed data with the "First Come First Served" (FCFS) discipline. Okamura et al. [53] collected partial group data and estimated parameters of a Markovian arrival process (MAP). They first calculated the log-likelihood by using the MLE method and applied the EM algorithm based on the results of the log-likelihood function.

In general, the EM algorithm can guarantee the likelihood increase for every iteration operation. Also, the E-step and M-step are often pretty simple for many problems in terms of implementation. In this work, the author considers applying the EM algorithm

to estimate the $M_t/M/1$ queueing model based on incomplete data.

On the other hand, all the mentioned past works estimated the parameters from the exact arrival time and service time. For example, the waiting times have a direct relation to the service distribution. Also, after observing in a fixed time, the arrival related data can directly be collected, such as the traffic volume or telephone calls. In other words, the past works assumed that we could observe the arrival time or the number of arrivals in a time interval. However, in a real-world situation, such the data cannot be observed, such as utilization data. For example, CPU utilization data is the most common data in computer systems. CPU utilization data reflect the computation cost of the CPU. However, the arrival process of the CPU queueing system cannot be observed directly. So the parameter estimation from CPU utilization data is complicated and challenging for researchers.

In this work, the author aims to estimate the NHPP arrival process of $M_t/M/1$ queueing systems, by using an MLE via the EM algorithm from incomplete utilization data. Firstly, the author makes several assumptions for utilization data. Then, the author generates an $M_t/M/1$ queueing system, whose jobs arrive at the system following an NHPP. Next, to simplify the non-stationary arrival process, the author makes an approximation of the NHPP input. Specifically, the author transforms the intensity function to a series of HPPs. For the estimation of HPPs, the author applies the MLE via the EM algorithm. In the experiments, the author first generates the utilization data through simulation. Also, the author collects real CPU utilization data. The evaluation results investigate the effectiveness of the proposed method.

3.3 $Mt/M/1/k$ Queue

Assume that the queueing system only has one server, called $M_t/M/1/k$, with the First Come First Served (FCFS) discipline. Suppose the tasks arrive at computer systems following an NHPP and represent the intensity function by the notation $\lambda(t)$. Since there

is only one server in the system, the first arrived task can be served directly. The next task waits in the queue if the previous job is being served. The queueing system performs a similar process for the next arrived tasks until the tasks waiting for service exceeds the capacity of k. The service time follows the exponential distribution, where the service rate is μ.

We can use a nonhomogeneous Markov Chain can represent the system behavior. The infinitesimal generator can be shown as follows.

$$\boldsymbol{Q}(t) = \begin{pmatrix} -\lambda(t) & \lambda(t) & & \\ \mu & -(\mu+\lambda(t)) & \lambda(t) & \\ & \ddots & \ddots & \\ & & \mu & -\mu \end{pmatrix} \tag{3.1}$$

3.3.1 Approximation of an NHPP

An NHPP is similar to an ordinary Poisson distribution, except that the mean rage of arrivals is allowed to be dynamically changed concerning time t. Since the intensity function $\lambda|(t)$ of an NHPP is a function of time t, it is not easy for the estimation of the arrival process. In general, a discrete model is more simple than the time related continuous model. It motived us to make an approximation to transform the intensity function $\lambda|(t)$ into a series of Poisson distributions. And, each Poisson distribution has a piecewise constant arrival rate λ.

Explicitly, we define the total observation time of the arrival process as $[0, T]$. Then we divide the observation time $[0, T]$ into n $(n \geq 1)$ equal time slices. We represent the equal time slices by the symbol Δt. Then, for the i-th $(1 \leq i \leq n)$ period, the jobs arrive at the system following an HPP, and the arrival rate is λ_i. If the length of the equal time slice is small enough, the series of the HPPs can be integrated to be a continuous NHPP.

The approximation of the NHPP can be formulated as follows.

$$\lambda(t) = \begin{cases} \lambda_1 & (0 \leq t \leq \Delta t) \\ \lambda_2 & (\Delta t < t \leq 2\Delta t) \\ \vdots & \vdots \\ \lambda_n & ((n-1)\Delta t < t \leq T) \end{cases} \tag{3.2}$$

3.3.2 Utilization Data

Define the utilization data as the time fractions of busy periods in fixed time intervals. The definition of utilization data shows one of the properties that the data collection process of utilization data allocates in a series of discrete-time slices. Assume that the arrival process in the i-th time slice $((i-1)\Delta t, i\Delta t]$ follows an HPP, where the arrival rate is λ_i. For the utilization data, we make several assumptions:

- Utilization is computed at every fixed time interval.
- Each time interval consists of two periods: the unobserved period and the observed period.
- Utilization on a time interval can be computed as a time fraction of busy time over total time only in the observed period.
- In an observed period, there is at most one change from busy (idle) to idle (busy).

The first three assumptions illustrate that utilization data could be monitored as a time fraction only at the observed period. The fourth assumption demonstrates that the observed period is sufficiently small so that more than one state change cannot occur in one observation period.

Figure 3.1 demonstrates the possible behavior of the system state of the observed and unobserved periods for utilization data. Let t_u and t_o be the time lengths of unobserved

and the observed periods, respectively. Let B_t represent the length of a busy time, and I_t represent the length of idle time in a time slot. Based on the mentioned assumptions, the utilization in a time slice $t_u + t_o$ can be formulated by $B_{t_o}/(B_{t_o} + I_{t_o})$ with $t_o \ll t_u$.

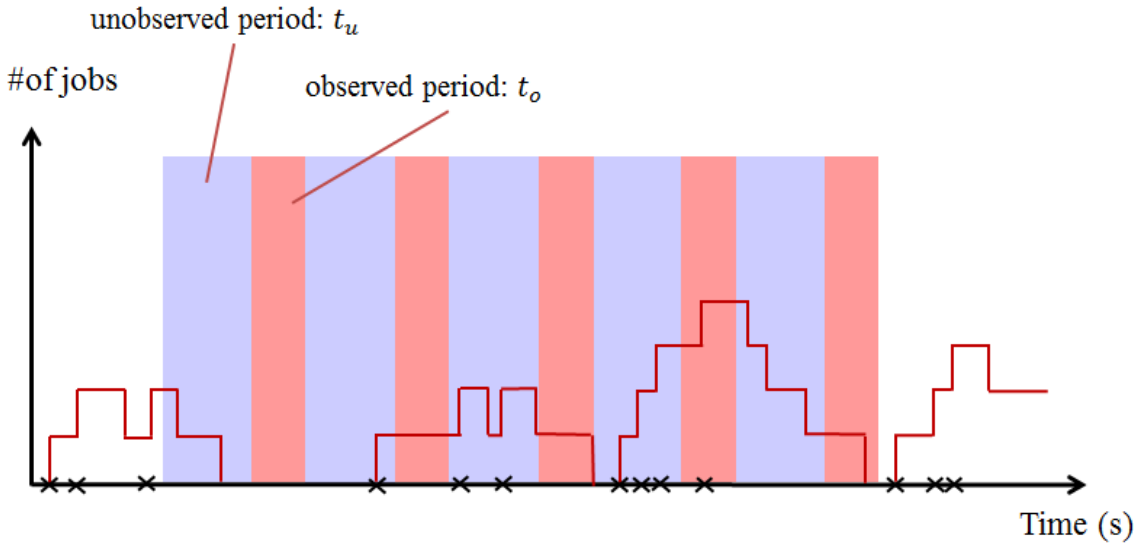

Figure 3.1: Possible behavior of system state and observed and unobserved periods for utilization.

3.4 Parameter Estimation

In this section, the author aims to estimate arrival parameters by using MLE. However, based on the above assumptions of utilization data, we cannot monitor the utilization data at the unobserved periods. In other words, the lack of data at the unobserved periods causes utilization data being incomplete data. The author applies the EM algorithm to resolve the problem of incomplete data.

3.4.1 Likelihood Function

In statistics, the MLE is a method of estimating the parameters by maximizing a likelihood function. The primary assumption of the MLE is that the observed data is most probable.

In this work, the author formulates the likelihood function from utilization data by using the MLE.

Define the two infinitesimal generators of i-th period as follows.

$$\boldsymbol{Q}_i^0 = \left(\begin{array}{c|cccc} -\lambda_i & & & & \\ \hline & -(\mu+\lambda_i) & \lambda_i & & \\ & & \ddots & \ddots & \\ & & & \mu & -\mu \end{array}\right), \tag{3.3}$$

$$\boldsymbol{Q}_i^1 = \left(\begin{array}{c|c} & \lambda_i \\ \hline \mu & \\ & \boldsymbol{O} \\ & \end{array}\right), \tag{3.4}$$

where $\boldsymbol{O}$ is the zero matrix. Note that $\boldsymbol{Q}_i = \boldsymbol{Q}_i^0 + \boldsymbol{Q}_i^1$.

Let utilization be $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n)$. The utilization data in $i-$th time period Δt is defined as $\mathcal{D}_i = (u_i^1, u_i^2, \ldots, u_i^k)$, $0 \leq u_i^j \leq 1$. We formulate the likelihood function as follows.

$$\mathrm{L}(\lambda_1, \lambda_2, \cdots, \lambda_n; \mathcal{D}) = \boldsymbol{p}\mathrm{L}_1(\lambda_1; \mathcal{D}_1)\mathrm{L}_2(\lambda_2; \mathcal{D}_2)\cdots\mathrm{L}_n(\lambda_n; \mathcal{D}_n)\mathbf{1}, \tag{3.5}$$

$$\mathrm{L}_i(\lambda_i; \mathcal{D}_i) = \mathcal{L}_i(u_i^1)\mathcal{L}_i(u_i^2)\cdots\mathcal{L}_i(u_i^k), \tag{3.6}$$

$$\mathcal{L}_i(u) = \exp(\boldsymbol{Q}_i t_u)\boldsymbol{\Lambda}_0 \exp(\boldsymbol{Q}_i^0(1-u)t_o)\boldsymbol{Q}_i^1 \exp(\boldsymbol{Q}_i^0 u t_o)$$

$$+ \exp(\boldsymbol{Q}_i t_u)\boldsymbol{\Lambda}_1 \exp(\boldsymbol{Q}_i^0 u t_o)\boldsymbol{Q}_i^1 \exp(\boldsymbol{Q}_i^0(1-u)t_o),$$

$$\text{if } 0 < u < 1 \tag{3.7}$$

$$\mathcal{L}_i(u) = \exp(\boldsymbol{Q}_i t_u)\boldsymbol{\Lambda}_0 \exp(\boldsymbol{Q}_i^0 t_o), \quad \text{if } u = 0, \tag{3.8}$$

$$\mathcal{L}_i(u) = \exp(\boldsymbol{Q}_i t_u)\boldsymbol{\Lambda}_1 \exp(\boldsymbol{Q}_i^0 t_o), \quad \text{if } u = 1, \tag{3.9}$$

where $\boldsymbol{p}$ is the initial probability vector. Also, $\boldsymbol{\Lambda}_0$ and $\boldsymbol{\Lambda}_1$ are $(K+1)$-by-$(K+1)$ block matrices;

$$\boldsymbol{\Lambda}_0 = \begin{pmatrix} 1 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{pmatrix}, \tag{3.10}$$

$$\boldsymbol{\Lambda}_1 = \begin{pmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}. \tag{3.11}$$

3.4.2 EM Algorithm

We apply the EM algorithm to resolve the problem of incomplete data and to improve the scalability of the estimation algorithm based on the previous computation result of the likelihood function.

In general, there are two steps, named E-step and M-step, for the EM algorithm:

- E step: compute the expected log-likelihood function with the posterior probability

of the hidden variables. The formula can be expressed:

$$E[\log p(\mathcal{D}, \mathcal{U}; \boldsymbol{\theta}^{'})], \tag{3.12}$$

where $\mathcal{D}$ is the observed data, $\mathcal{U}$ is the missing data in the unobserved period and $\boldsymbol{\theta}^{'}$ is the parameter vector.

- M step: maximize the expected log-likelihood function to find the parameter $\boldsymbol{\theta}$

$$\boldsymbol{\theta} = argmax E[\log p(\mathcal{D}, \mathcal{U}; \boldsymbol{\theta}^{'})], \tag{3.13}$$

For example, EM formula for the arrival rate from i state to j state as $\lambda_{i,j}$ is given by

$$\lambda_{i,j} = \frac{E[N_{i,j}]}{E[S_i]} = \frac{E[N_{i,j}^{\boldsymbol{U}} + N_{i,j}^{\boldsymbol{O}} \mid \mathcal{D}]}{E[S_i^{\boldsymbol{U}} + S_i^{\boldsymbol{O}} \mid \mathcal{D}]}, \tag{3.14}$$

where $N_{i,j}$ is the number of transition from state i to state j, S_i is the sojourn time in state i, $N_{i,j}^{\boldsymbol{U}}$ is the number of transition from state i to state j at unobserved time period, $N_{i,j}^{\boldsymbol{O}}$ is the number of transition from state i to state j at observed time period, $S_i^{\boldsymbol{U}}$ is the sojourn time in state i at unobserved period and $S_i^{\boldsymbol{O}}$ is the sojourn time in state i at observed period.

Let the symbol T be the total monitor time, and the symbol n be the number of equal time slices of the time T. If the n is too small, the length of a time slice is too large to estimate the intensity function well. For example, when $n = 1$, the intensity function $\lambda(t) = \lambda$. The NHPP is transformed into an HPP with the arrival rate λ. If n is too large, the length of a time slice tends to be too small, which may cause the overfitting problem that the estimation $\lambda(t)$ overreacting to minor fluctuations.

In this work, n is the only hyper-parameter. The author utilizes the Akaike Information Criterion (AIC) to quantify the goodness of the model. The AIC is an estimator of the

relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection. When AIC takes the minimum value, the n may the best.

$$AIC = -2LLF + 2(\sharp\ of\ parameters) \tag{3.15}$$

where LLF is the maximum value of the log-likelihood function.

So we can use this method to select the minimum value of AIC as the number of parameters is from 1 to n ($n \geq 1$).

3.5 Numerical Experiments

In this section, the author conducts two numerical experiments, the experiment on simulation and the experiment on real CPU utilization, respectively.

3.5.1 Simulation

We set the intensity function of the NHPP is as follows.

$$\lambda(t) = sin(0.004t) + 1 \tag{3.16}$$

For the $M_t/M/1/k$ queueing system, we set the exponential service rate to $\mu = 10$ with the system capacity of $k = 10$. Also, we define the unobserved time length and the observed time length as $t_u = 0.95$, $t_o = 0.05$. We set the total observation time is $T = 2000$, and monitor the utilization data per second. Figure 3.2 demonstrates the utilization data of simulation.

The x-axis represents the observation times. Since the utilization data are collected per second, we can collect 2000 utilization data. The y-axis on the left represents the utilization data. The utilization changes rapidly from 0 to 1. The y-axis on the right is the intensity function, shown in formula 3.16. The utilization tends to be high when the value of the intensity function is large; while the utilization is changed to be zero when the intensity function is zero.

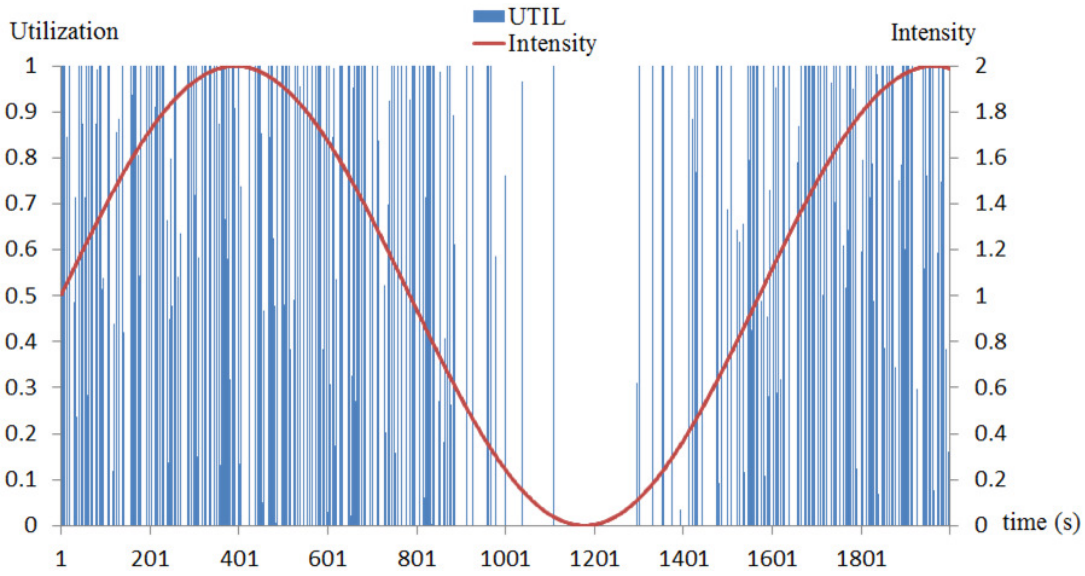

Figure 3.2: Utilization of simulation.

The estimated intensity function changes when the values of n change. To select the best model concerning n, we utilize the AIC approach. Table 3.1 exhibits the values of AIC with respect to n, where $1 \leq n \leq 30$. To examine the effect of a large value of n, we also estimate the intensity function, when $n = 100$.

From the table, we find the optimal number of the time interval is $n = 12$. The minimum value of the AIC is 1280.997. The estimation results in the table are further depicted from Figure 3.7 to Figure 3.37.

In Figure 3.7, the estimation of the intensity function is a constant over the totoal $T = 2000$. It means that the intensity function of the NHPP is same to the HPP. The

Table 3.1: AIC of the model.

n	LLF	AIC
1	-712.987	1427.973
2	-698.927	1401.854
3	-681.246	1368.492
4	-648.228	1304.455
5	-662.962	1335.924
6	-639.015	1290.030
7	-644.716	1303.431
8	-635.233	1286.466
9	-632.422	1282.844
10	-634.796	1289.592
11	-634.506	1291.011
12	-628.498	1280.997
13	-633.896	1293.792
14	-626.879	1281.758
15	-629.794	1289.588
16	-626.644	1285.288
17	-629.651	1293.302
18	-625.456	1286.913
19	-625.239	1288.479
20	-627.769	1295.537
21	-625.390	1292.780
22	-625.998	1295.995
23	-624.723	1295.447
24	-624.933	1297.865
25	-621.882	1293.765
26	-623.207	1298.413
27	-623.246	1300.491
28	-624.711	1305.421
29	-623.426	1304.851
30	-618.796	1297.592
100	-590.283	1380.566

value of the AIC equals to 1427.973, and it is much higher than other models because the HPP cannot model the arrival process well. This result also illustrates that the HPP cannot fit the real-world case. The $M_t/M/1/k$ queueing system, which has the NHPP arrival process is better than the ordinary $M/M/1/k$ queueing system in this case.

Similarly, the estimation results when $n = 2, 3, ..., 11$ exhibit from Figure 3.8 to 3.17. As the value of time slice n increase, the value of the AIC tends to be smaller. It means that the model tends to be better when the time slice n is more significant from 1 to 11. When $n = 12$, the AIC is the smallest, and the estimated result exhibits in Figure 3.18.

As the value of n is larger than 12, the AIC is larger. The reults exhibit from Figure 3.19 to 3.36. In these cases, although the estimated intensity functions approach the exact intensity function, they tend to cause the complexity of the model increasing. When $n = 100$ in Figure 3.37, the time slice n tends to be too large to estimate the optimal intensity function.

3.5.2 Real CPU Utilization

In this experiment, we utilize the real CPU utilization data to estimate the intensity function of the $M_t/M/1/k$ queueing system. Specifically, we monitor the CPU utilization data from two servers, named Server A and Server B, in a laboratory in Hiroshima University. We exhibit the evaluation of the average response time for the integrated system.

We monitor the CPU utilization data for about one week (7 days). We collect the CPU utilization data per ten minute and collect it for 1000 times. The capacity of the queueing system is set to $k = 20$. The observed time length and the unobserved time length are set to $t_u = 599s$, and $t_o = 1s$. We set service time of the Server A, and the Server B follow an exponential distribution, and service rates of A and B are $\mu_A = 3$ and $\mu_B = 2.5$, respectively. The collected real CPU utilization data exhibit in Figure 3.3 and Figure 3.4.

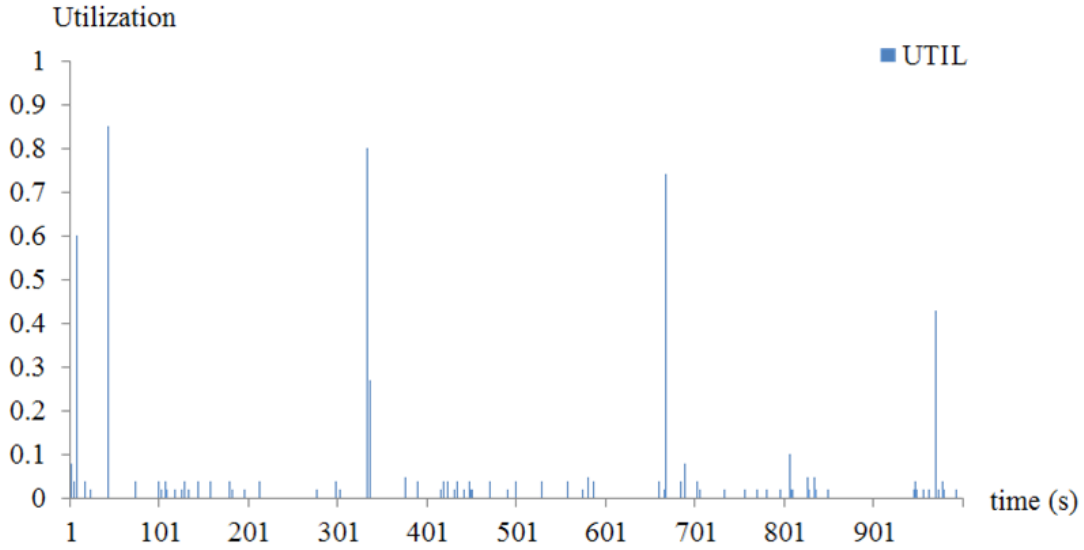

Figure 3.3: CPU utilization of server A.

Table 3.2 exhibits the values of AIC with respect to n, where $1 \leq n \leq 30$.

From the table, we find the optimal number of the time interval of Server A is $n = 17$. The minimum value of the AIC is 511.220. The estimation result when $n = 17$, is depicted in Figure 3.5.

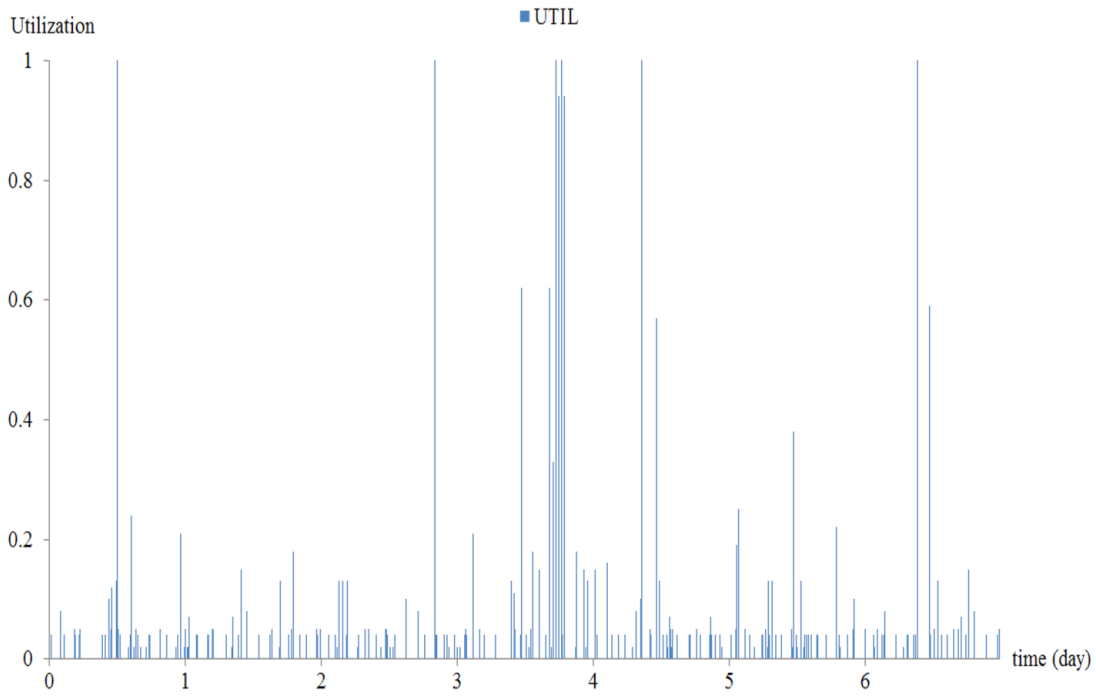

Figure 3.4: CPU utilization of server B.

Table 3.3 exhibits the values of AIC with respect to n, where $1 \leq n \leq 30$ of Server B.

From the table, we find the optimal number of the time interval of Server B is $n = 1$. The minimum value of the AIC is 1075.980. The estimation result when $n = 1$, is depicted in Figure 3.6. It means that the optimal arrival process of Server B is an HPP.

To evaluate the performance of the integrated system, we integrate both Server A and Server B. In other words, in an integrated system, after knowing the arrival intensity function and service distribution of Server A and Server B, we integrate these two servers into a system and deliver the overarching performance.

We set the integrated intensity function of Server A and Server B to $\lambda_{all} = \lambda_A + \lambda_B$. To indicate the performance, we utilize the average response time of the integrated system. The average response time is the time from a job arriving at the system to the job responding to a request after service. We represent the average response time by the symbol T_{res}.

We set the service rate from 0.5 to 20.0. The performance of the integrated system

Table 3.2: AIC of the real CPU utilzation from Server A.

n	LLF	AIC
1	-258.486	518.972
2	-258.261	520.522
3	-258.262	522.524
4	-257.093	522.186
5	-254.924	519.848
6	-255.252	522.504
7	-255.614	525.228
8	-253.444	522.888
9	-255.874	529.748
10	-252.190	524.380
11	-251.220	524.440
12	-249.700	523.400
13	-249.126	524.252
14	-245.136	518.272
15	-244.162	518.324

n	LLF	AIC
16	-247.170	526.340
17	-238.610	511.220
18	-248.421	532.842
19	-244.532	527.064
20	-238.210	516.420
21	-242.499	526.998
22	-237.127	518.254
23	-237.382	520.764
24	-238.376	524.752
25	-242.572	535.144
26	-238.602	529.204
27	-237.785	529.570
28	-236.595	529.190
29	-235.270	528.540
30	-237.437	534.874

Table 3.3: AIC of the real CPU utilzation from Server B.

n	LLF	AIC
1	-536.990	1075.980
2	-536.586	1077.172
3	-536.922	1079.844
4	-536.503	1081.006
5	-535.041	1080.082
6	-535.187	1082.374
7	-536.296	1086.592
8	-535.955	1087.910
9	-536.378	1090.756
10	-534.285	1088.570
11	-534.443	1090.886
12	-534.075	1092.150
13	-534.352	1094.704
14	-534.008	1096.016
15	-533.239	1096.478

n	LLF	AIC
16	-530.934	1093.868
17	-532.934	1099.868
18	-530.202	1096.404
19	-530.308	1098.616
20	-529.203	1098.406
21	-531.481	1104.962
22	-525.890	1095.780
23	-531.605	1109.210
24	-530.523	1109.046
25	-528.260	1106.520
26	-531.467	1114.934
27	-528.686	1111.372
28	-529.612	1115.224
29	-527.595	1113.190
30	-527.436	1114.872

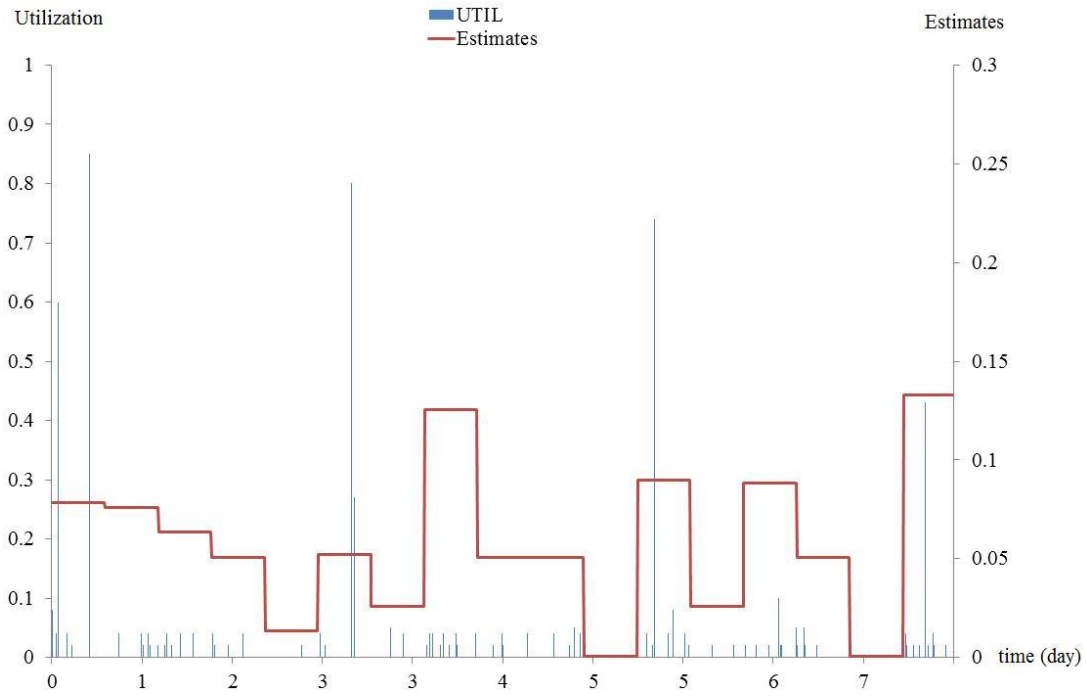

Figure 3.5: The optimal estimation result for intensity function (n=17) of server A.

shows in Table 3.4.

We can observe that the average response time reduces when the service rate μ increases. The average response time tends to be changed slowly when the service rate is high enough.

In addition, we discuss the integrated system of two cases:

- **Case 1**: We set $n = 1$ of Server A, and set $n = 1$ of Server B. In this case, both Server A and Server B are HPPs. In other words, we integrate two $M/M/1/k$ queueing systems.

- **Case 2**: We set $n = 17$ of Server A, and set $n = 1$ of Server B. In this case, both the two models of Server A and Server B are the optimal models based on the previous experiments. In other words, we integrate the $M_t/M/1/k$ and the $M/M/1/k$ queueing systems into an integrated system.

For the above two cases, we calculate the average response times and compare the

Table 3.4: Response time of integrated system.

$ServiceRate\ \mu$	$T_{res}\ (s)$
0.5	4.285
1.0	1.271
1.5	0.758
2.0	0.545
2.5	0.426
3.0	0.350
3.5	0.298
4.0	0.259
4.5	0.229
5.0	0.206
5.5	0.186
6.0	0.171
6.5	0.157
7.0	0.146
7.5	0.136
8.0	0.127
8.5	0.120
9.0	0.113
9.5	0.107
10.0	0.101

$ServiceRate\ \mu$	$T_{res}\ (s)$
10.5	0.096
11.0	0.092
11.5	0.088
12.0	0.084
12.5	0.081
13.0	0.078
13.5	0.075
14.0	0.072
14.5	0.070
15.0	0.067
15.5	0.065
16.0	0.063
16.5	0.061
17.0	0.059
17.5	0.058
18.0	0.056
18.5	0.054
19.0	0.053
19.5	0.052
20.0	0.050

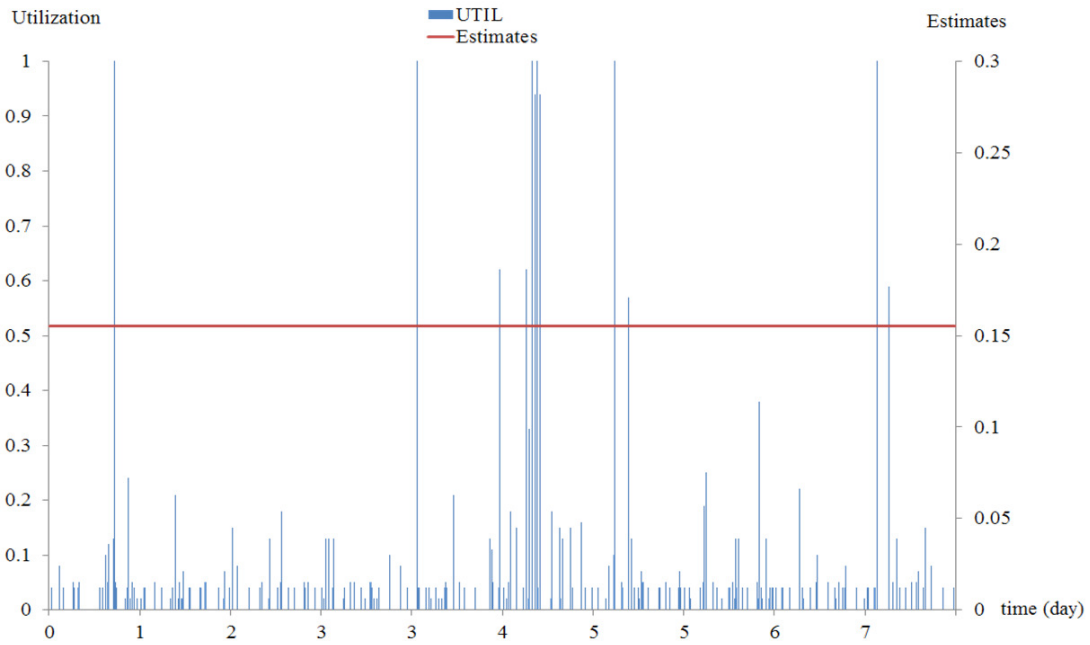

Figure 3.6: The optimal estimation result for intensity function (n=1) of server B.

differences.

Table 3.5 demonstrates the average response time of case 1 (T_{res}^{HPP}) and case 2 (T_{res}^{NHPP}) of the integrated system.

Table 3.5: Comparison of response time between HPP and NHPP for the integrated system.

μ	T_{res}^{HPP}	T_{res}^{NHPP}
0.3	17.129	24.517
0.4	6.502	7.260
0.5	3.817	4.046
0.6	2.706	2.756
0.7	2.052	2.113
0.8	1.667	1.705
0.9	1.411	1.435
1.0	1.225	1.240
1.1	1.083	1.097

μ	T_{res}^{HPP}	T_{res}^{NHPP}
1.2	0.971	0.980
1.3	0.881	0.888
1.4	0.807	0.813
1.5	0.742	0.750
1.6	0.692	0.695
1.7	0.644	0.649
1.8	0.604	0.608
1.9	0.570	0.573
2.0	0.540	0.540

The author has already confirmed that the proposed queueing system can model the arrival process better than the typical $M/M/1/k$ system. When the service rate μ is small,

the difference between T_{res}^{HPP} and T_{res}^{NHPP} is large. Also, with the service rate μ being more extensive, the difference tends to be smaller, and the different kinds of arrival processes have a little effect on the response time..

3.6 Concluding Remarks

In this chapter, we generate a novel queueing system to estimate the arrival process with utilization data. In particular, since the Poisson distribution cannot fit the real-world situations, we generate the queueing system with the arrival process following an NHPP. An NHPP can model the dynamical changing process of arrival process better than an HPP. However, since the intensity function of an NHPP is a function of time t, it is more complicated and challenges to estimate the intensity function from such a continuous process.

To simplify the problem, the author made an approximation of the NHPP arrival input. After the approximation, the author transformed the continuous arrival process into a series of discrete piecewise HPPs.

On the other hand, the author defined that utilization is a fraction of busy time at a fixed time interval. The definition illustrated that there existed the observed time intervals and unobserved time intervals of the utilization data. In other words, utilization data is a kind of incomplete data. For the estimation of the intensity function, the author proposed to use the MLE via the EM algorithm.

In the experiments, the author first generated the simulated utilization data for an $M_t/M/1/k$ queueing system. Then, the author investigated the effectiveness of the proposed approach. Furthermore, the author utilized the real CPU utilization data again, to estimate the intensity function of the two servers. Besides, the author used the average response time to examine the performance evaluation of an integrated system.

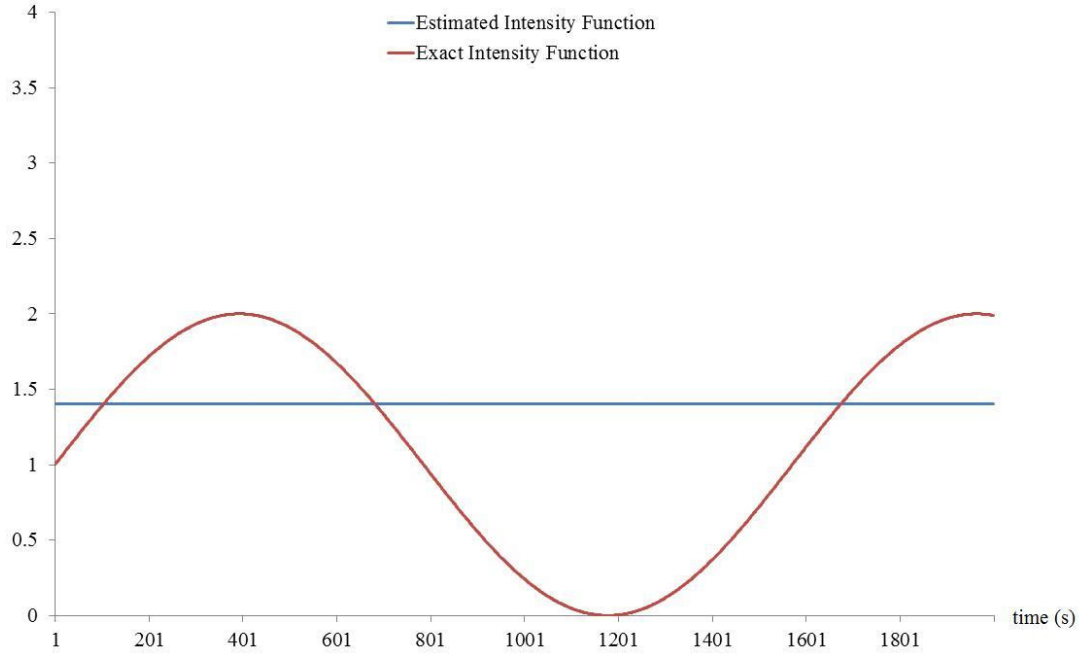

Figure 3.7: Estimation results for intensity function (n=1).

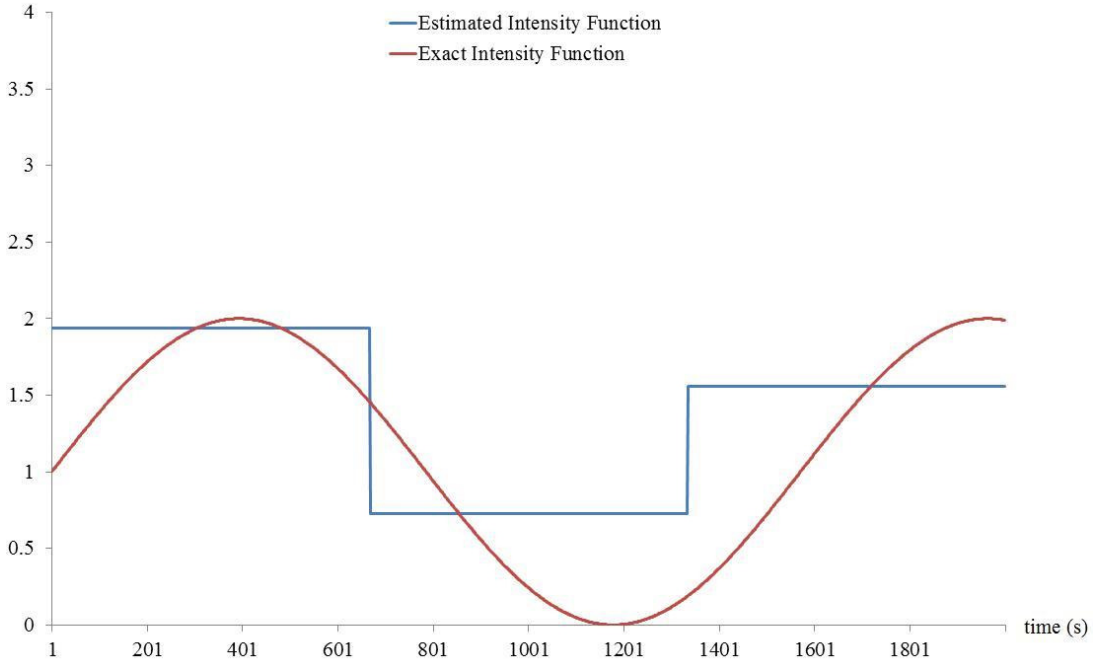

Figure 3.8: Estimation results for intensity function (n=2).

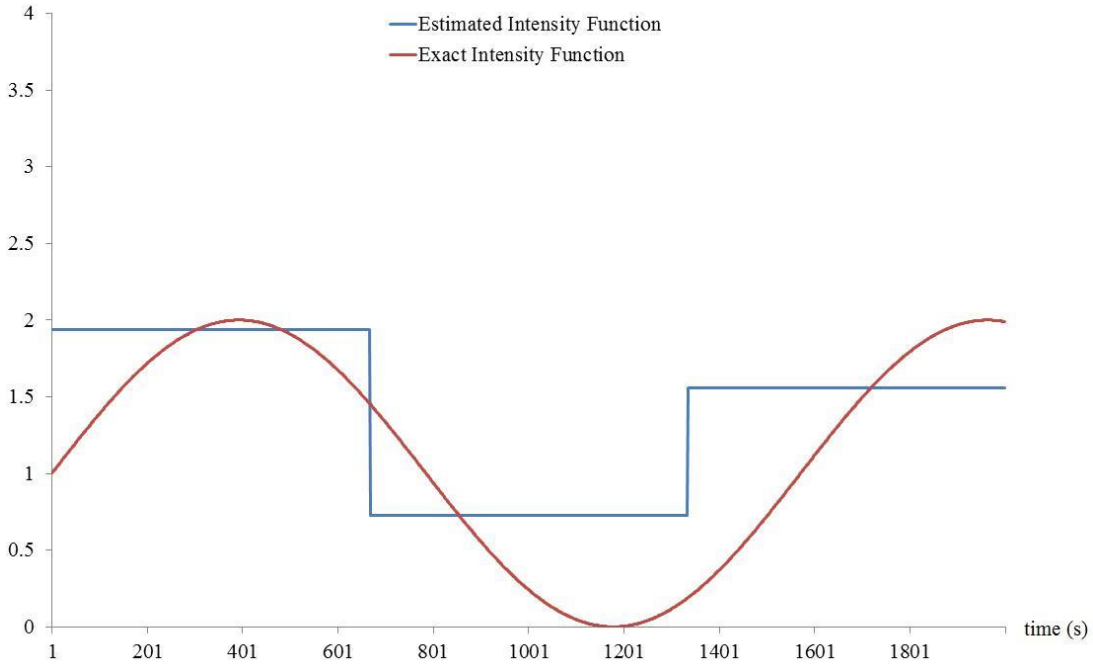

Figure 3.9: Estimation results for intensity function (n=3).

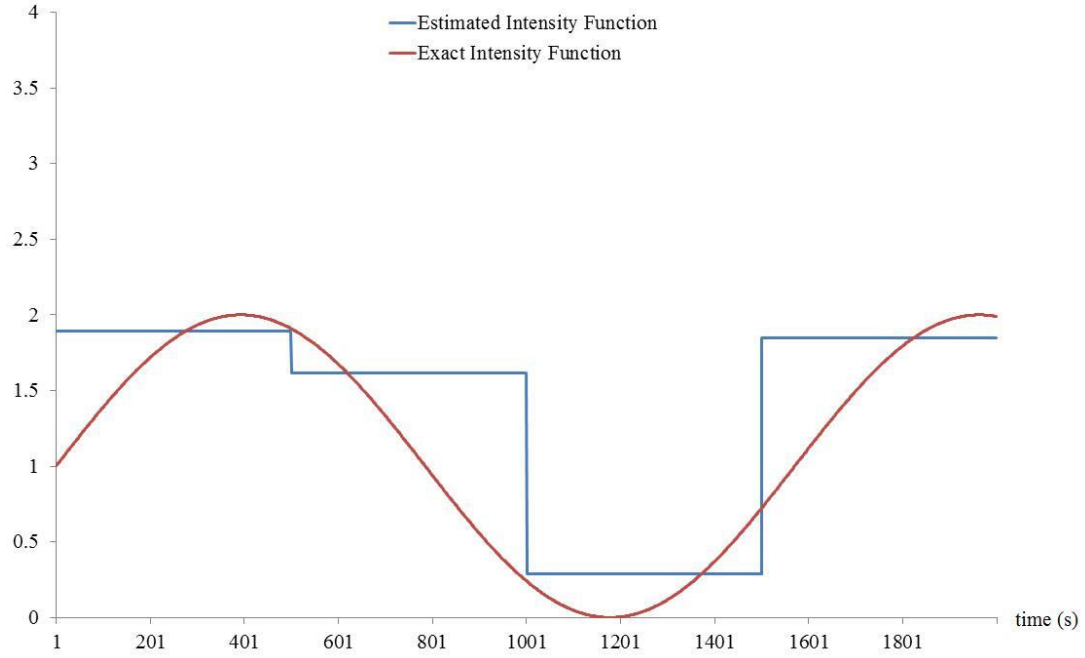

Figure 3.10: Estimation results for intensity function (n=4).

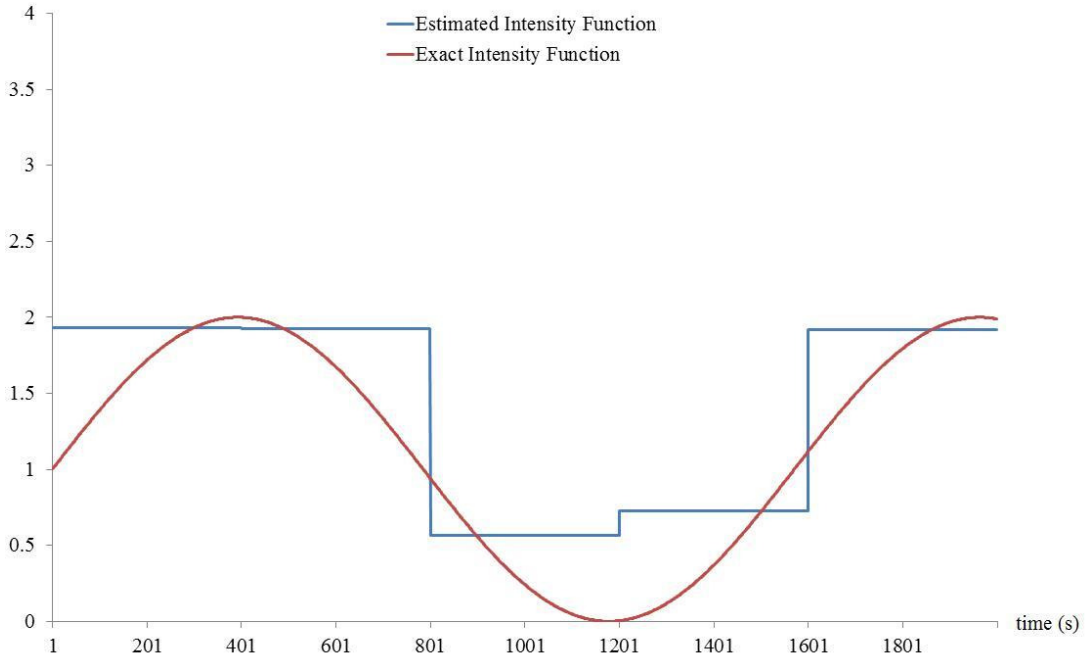

Figure 3.11: Estimation results for intensity function (n=5).

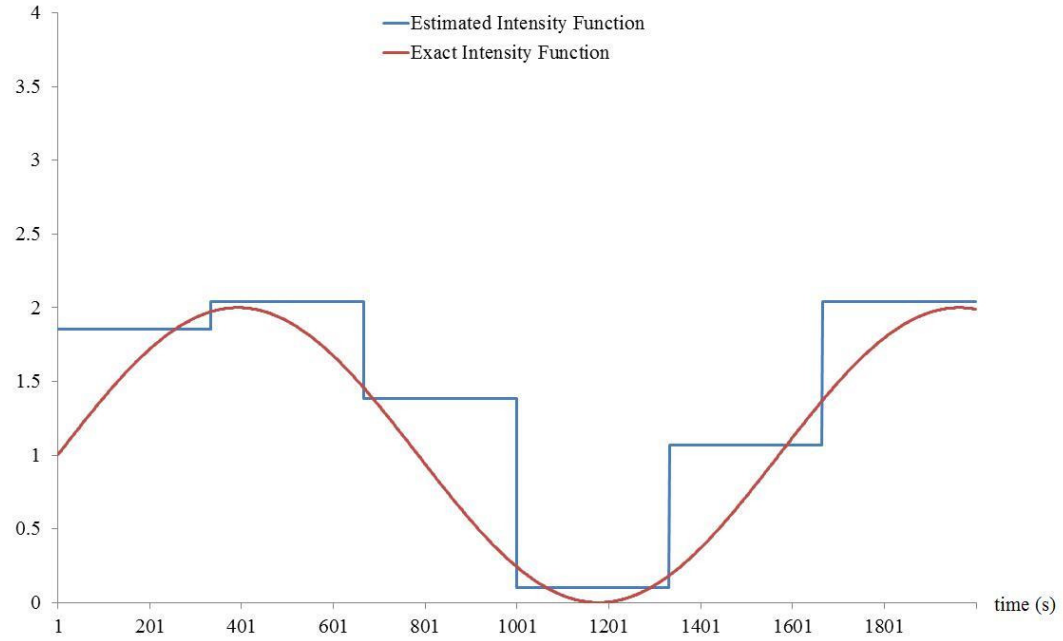

Figure 3.12: Estimation results for intensity function (n=6).

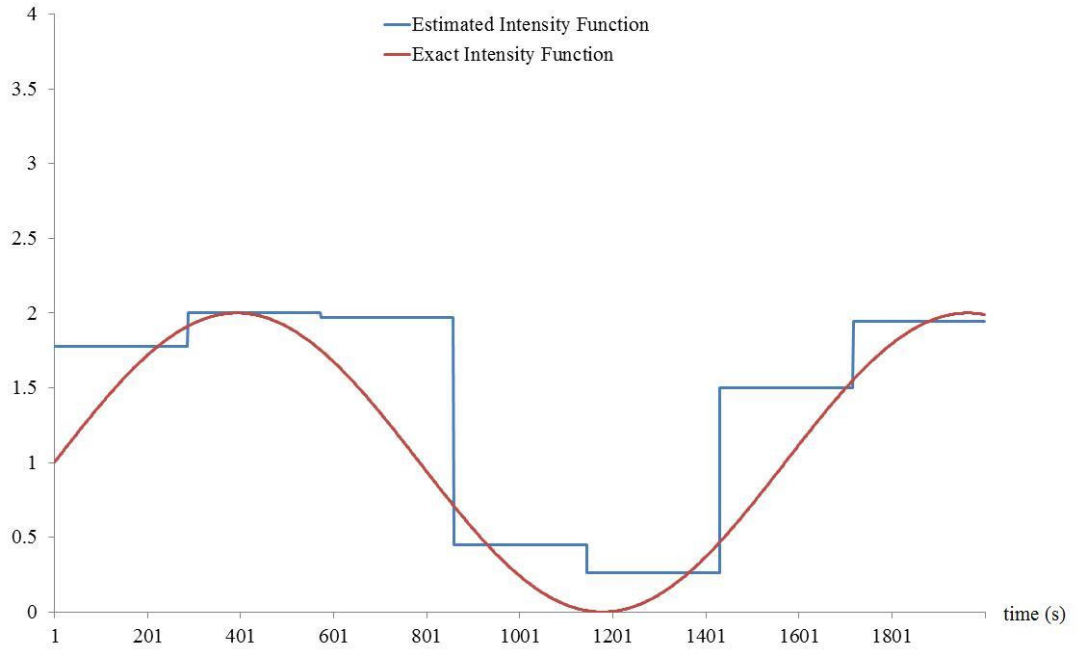

Figure 3.13: Estimation results for intensity function (n=7).

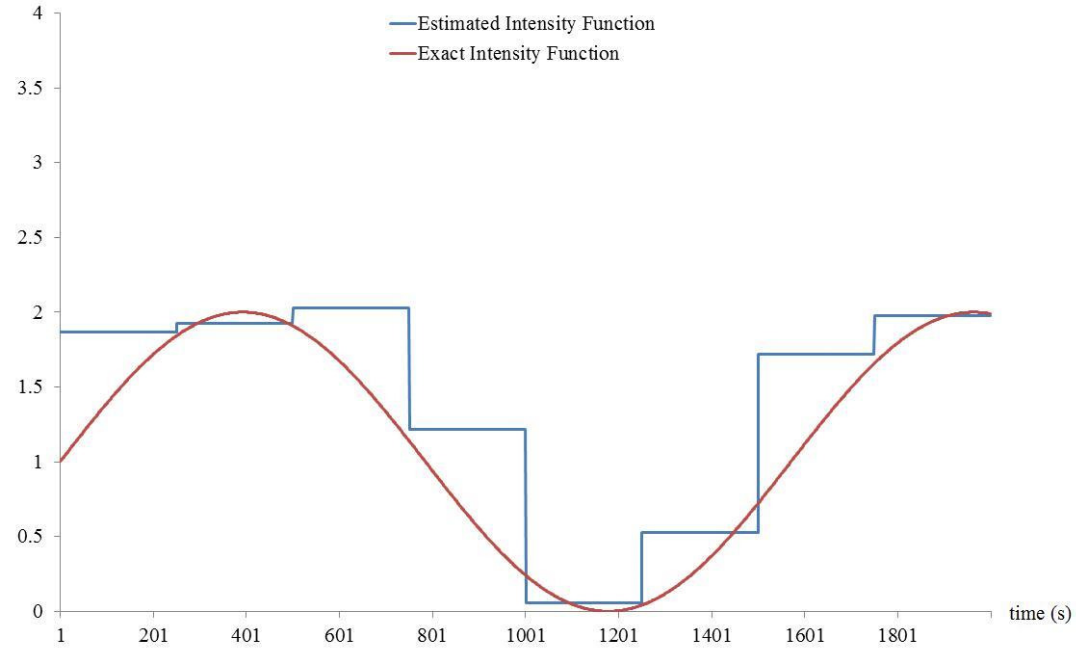

Figure 3.14: Estimation results for intensity function (n=8).

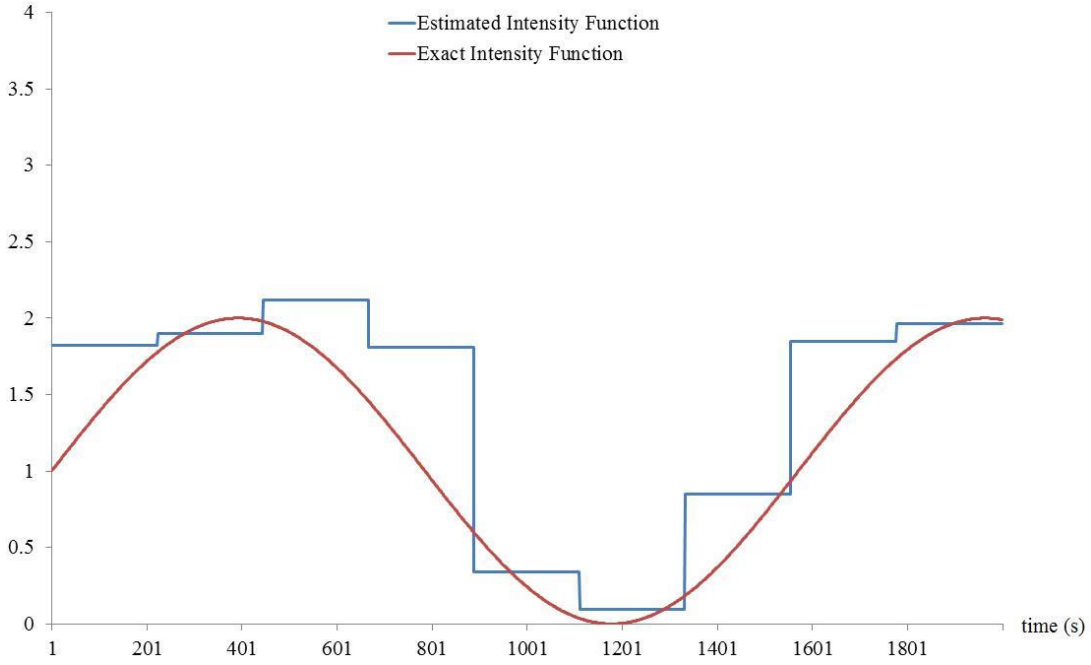

Figure 3.15: Estimation results for intensity function (n=9).

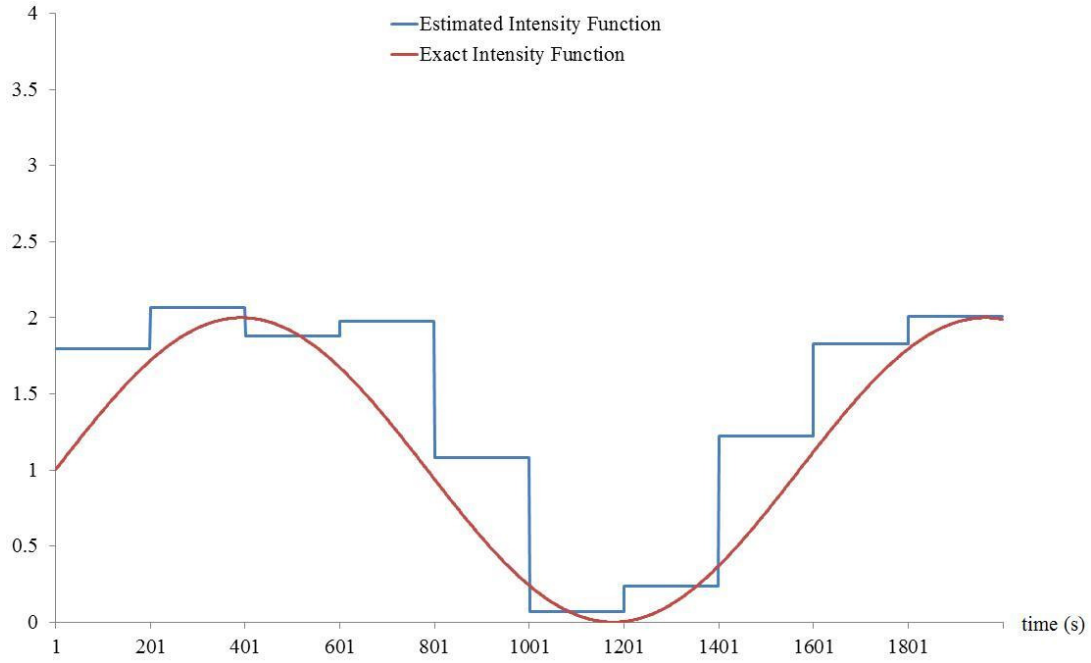

Figure 3.16: Estimation results for intensity function (n=10).

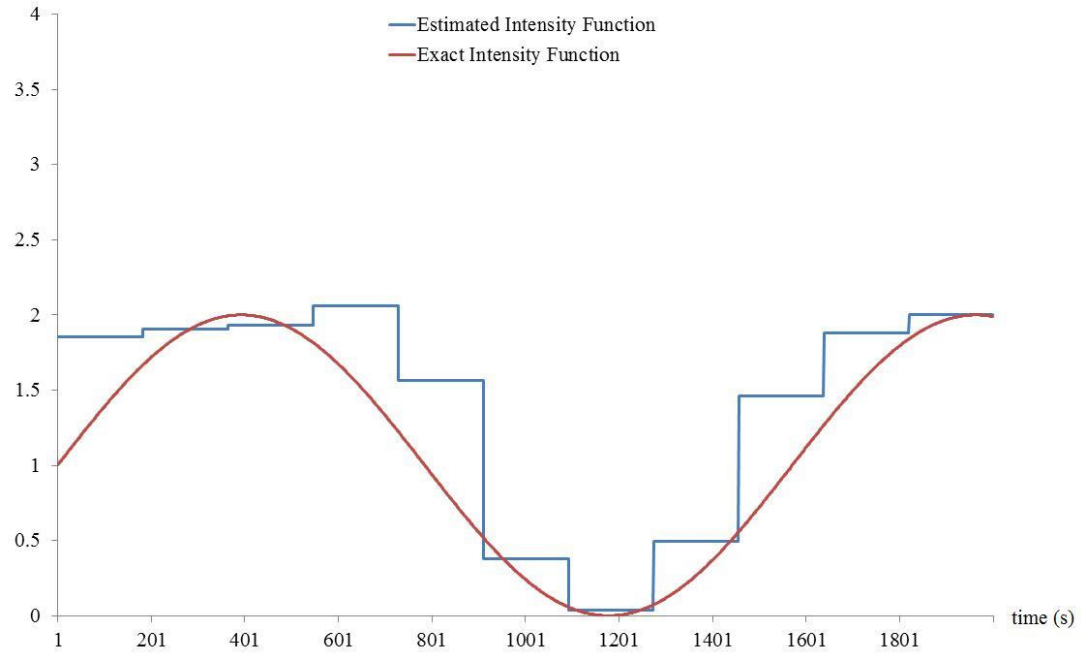

Figure 3.17: Estimation results for intensity function (n=11).

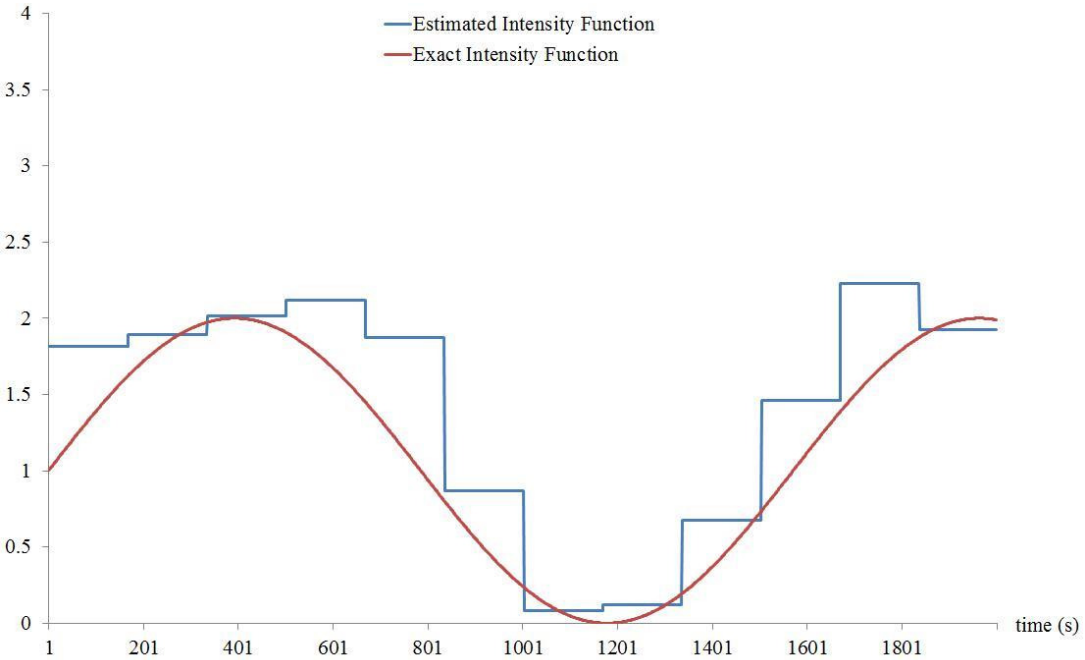

Figure 3.18: Estimation results for intensity function (n=12).

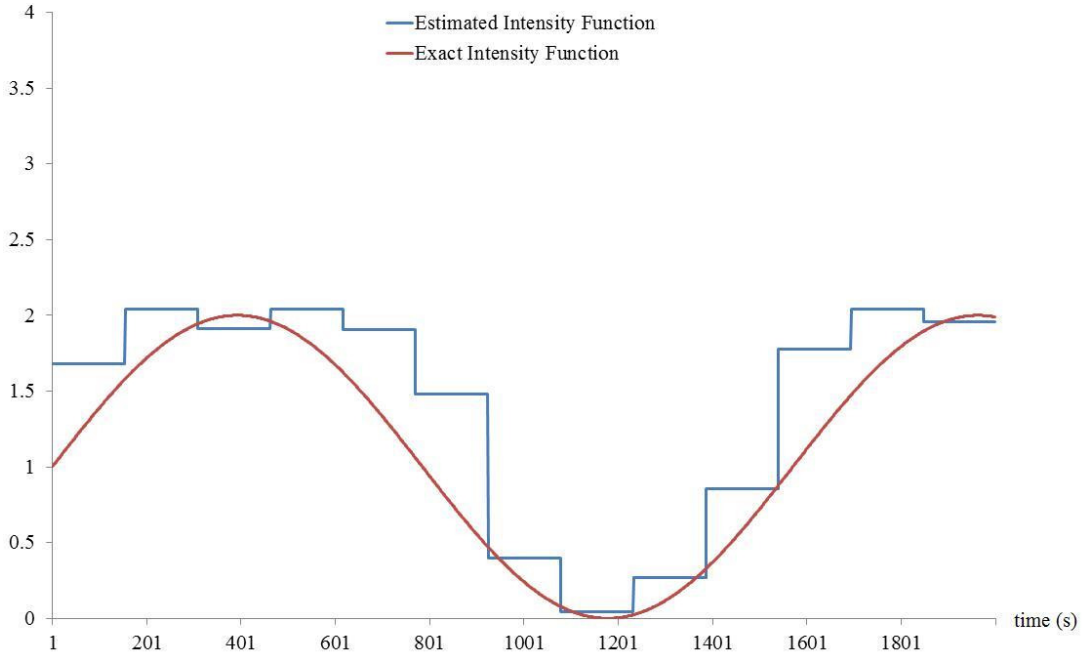

Figure 3.19: Estimation results for intensity function (n=13).

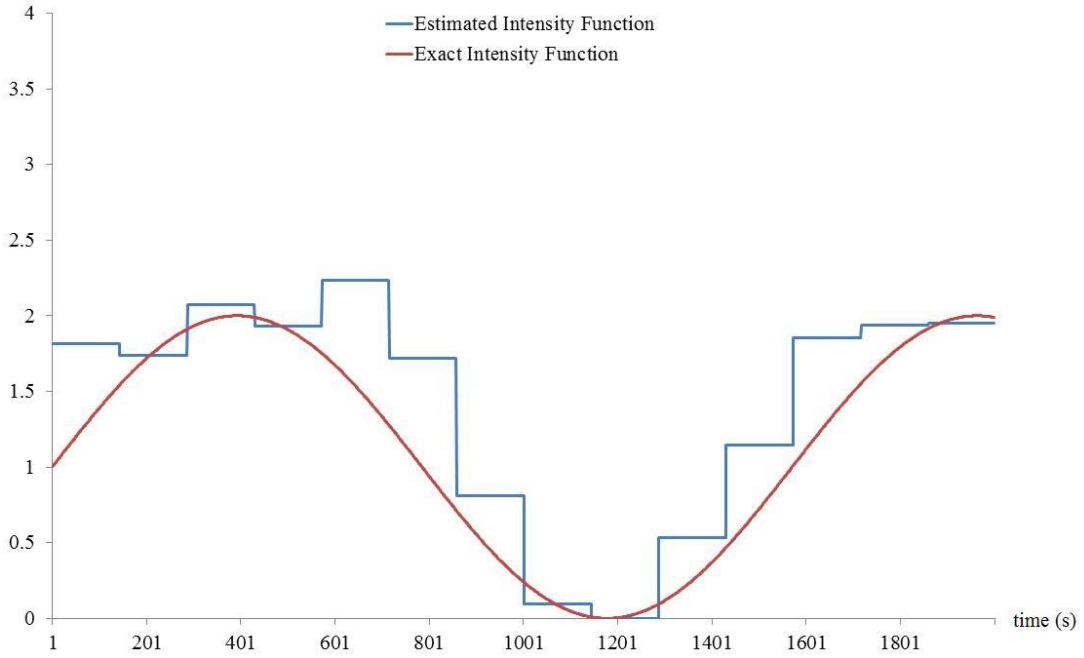

Figure 3.20: Estimation results for intensity function (n=14).

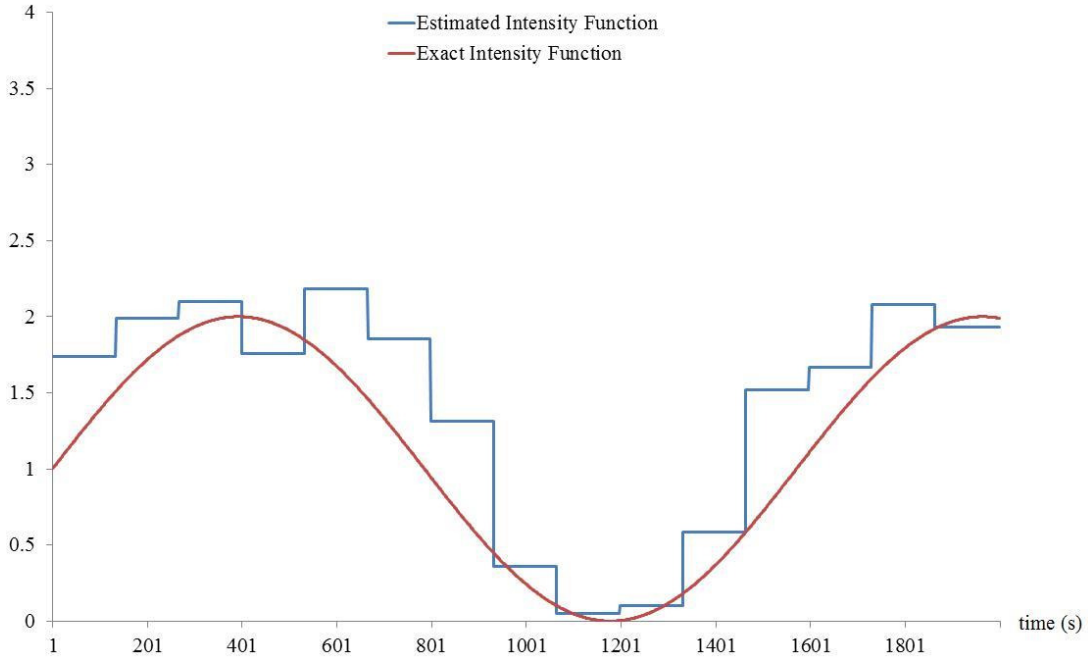

Figure 3.21: Estimation results for intensity function (n=15).

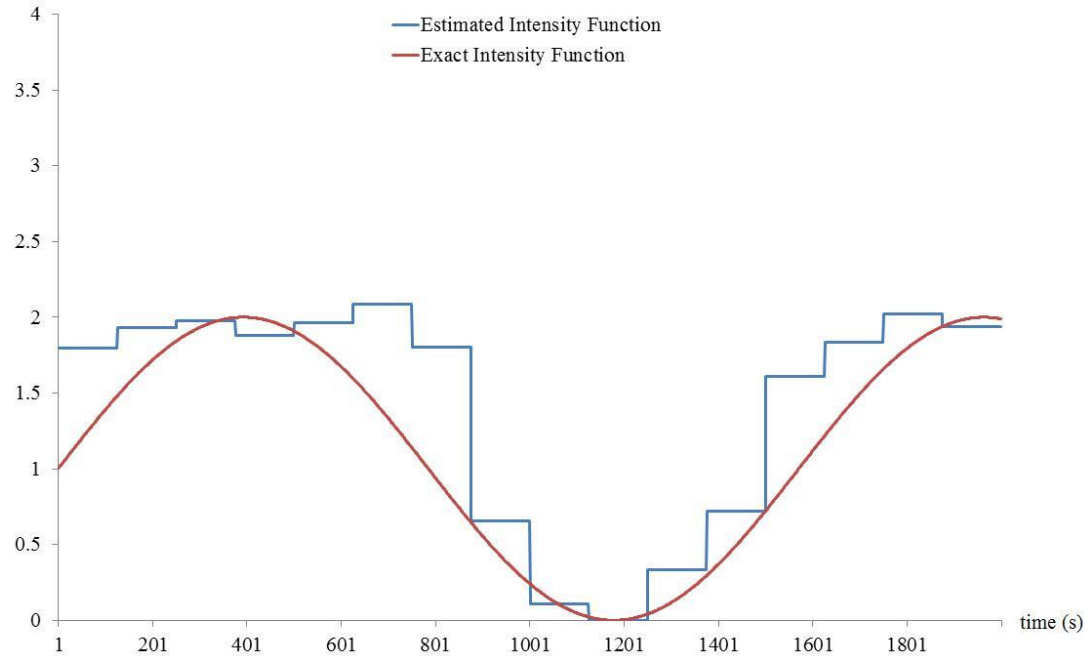

Figure 3.22: Estimation results for intensity function (n=16).

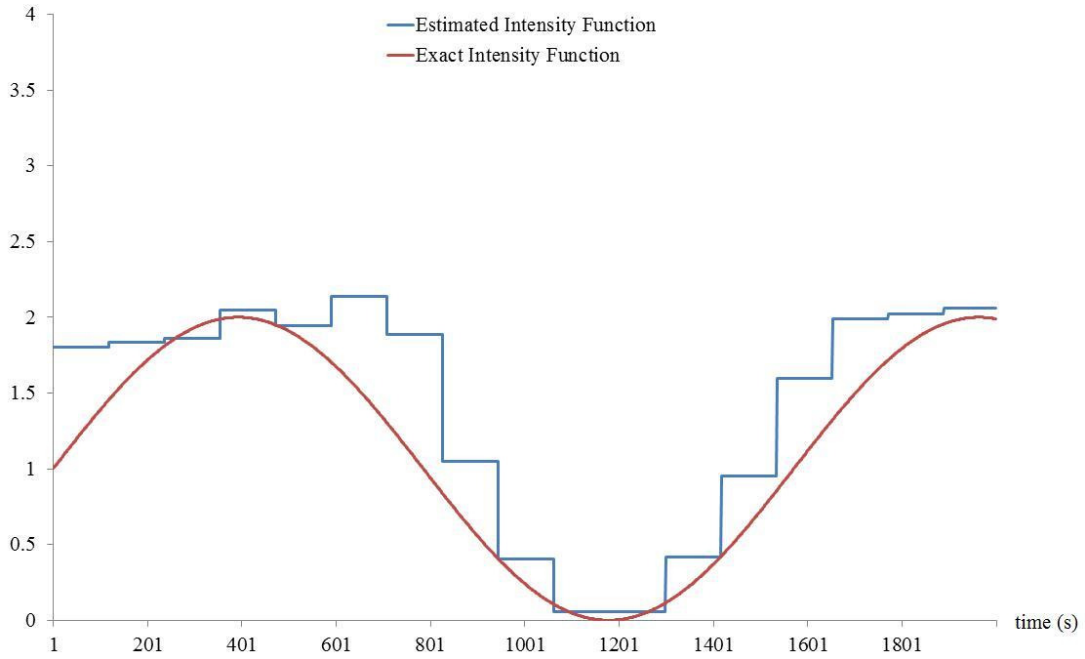

Figure 3.23: Estimation results for intensity function (n=17).

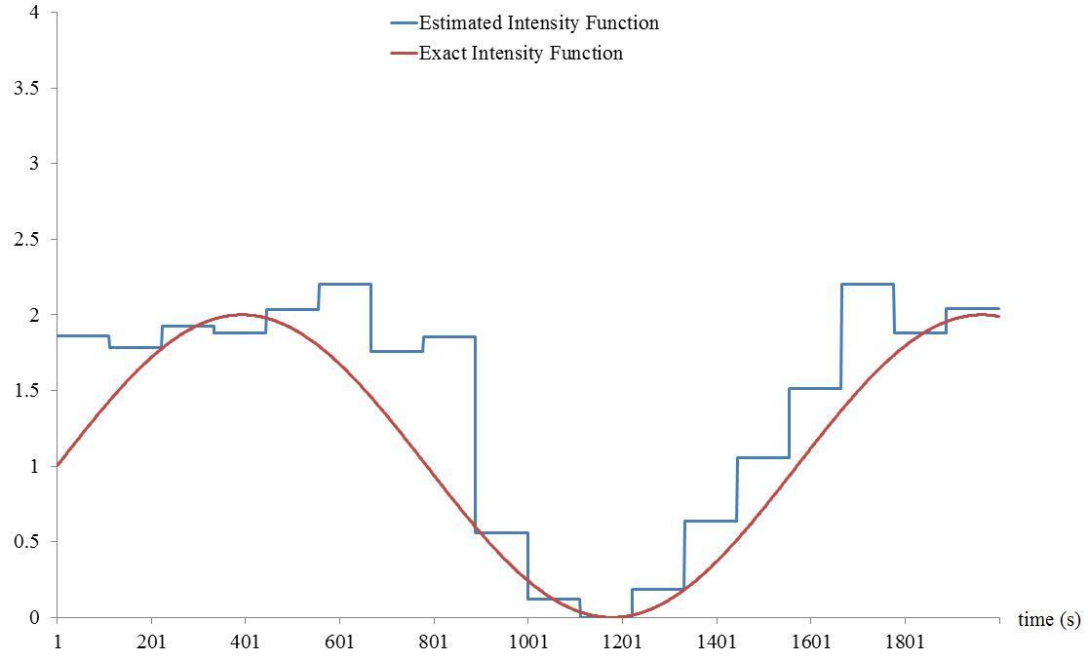

Figure 3.24: Estimation results for intensity function (n=18).

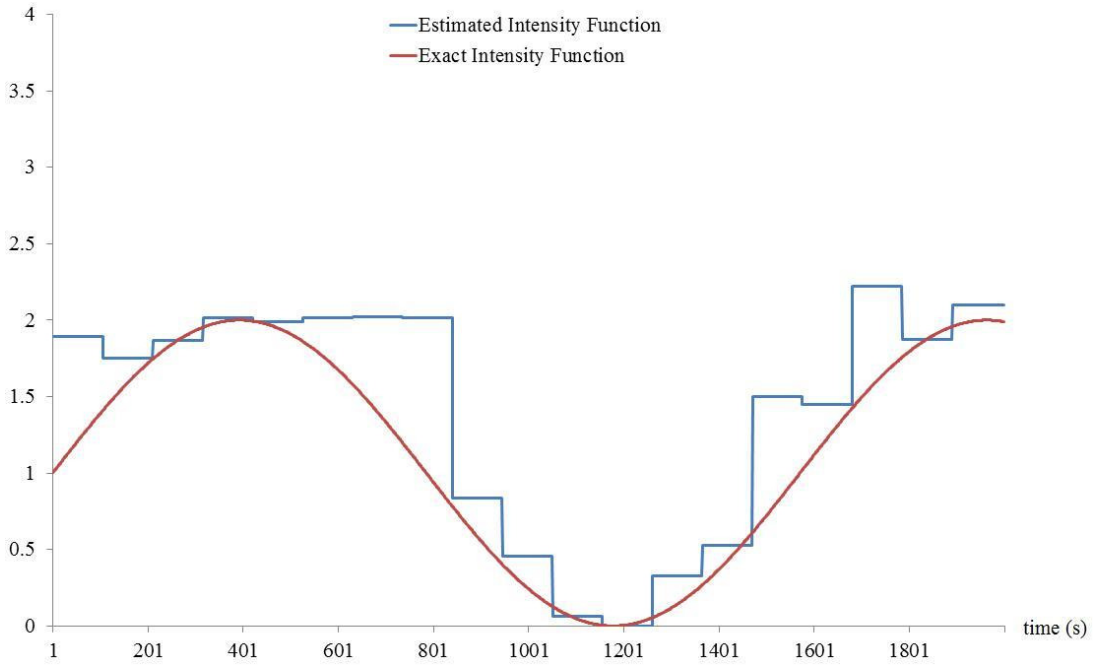

Figure 3.25: Estimation results for intensity function (n=19).

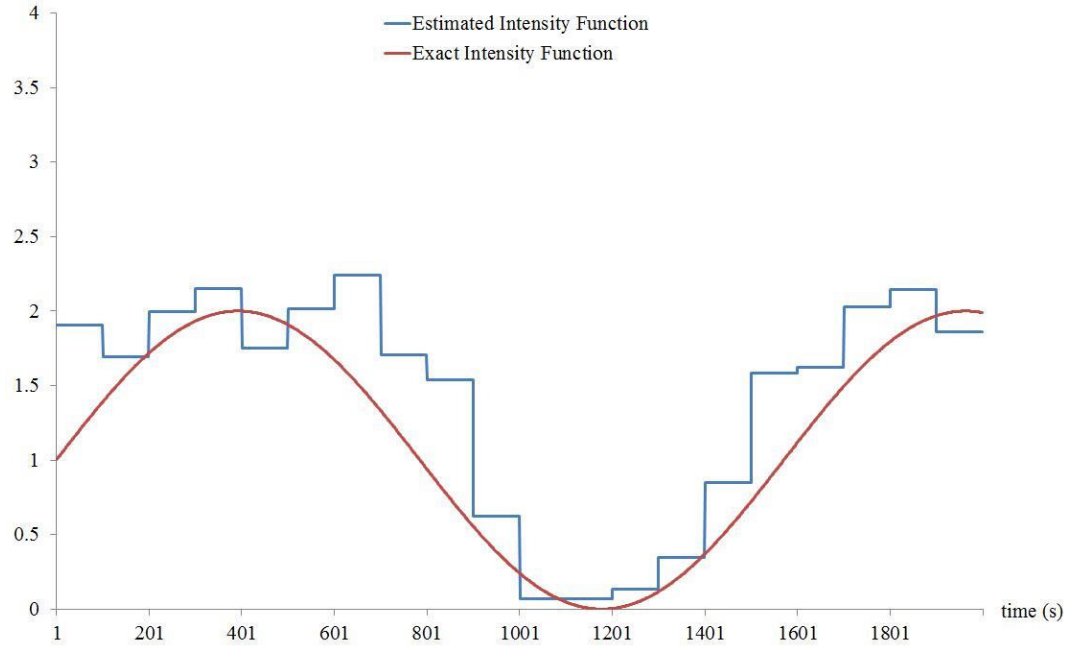

Figure 3.26: Estimation results for intensity function (n=20).

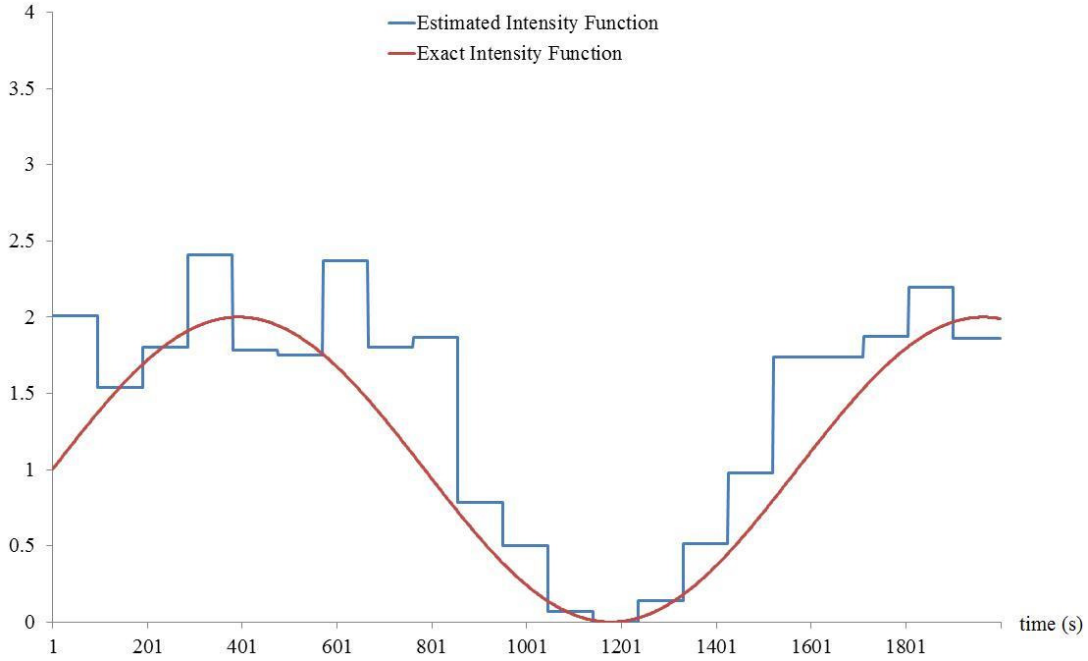

Figure 3.27: Estimation results for intensity function (n=21).

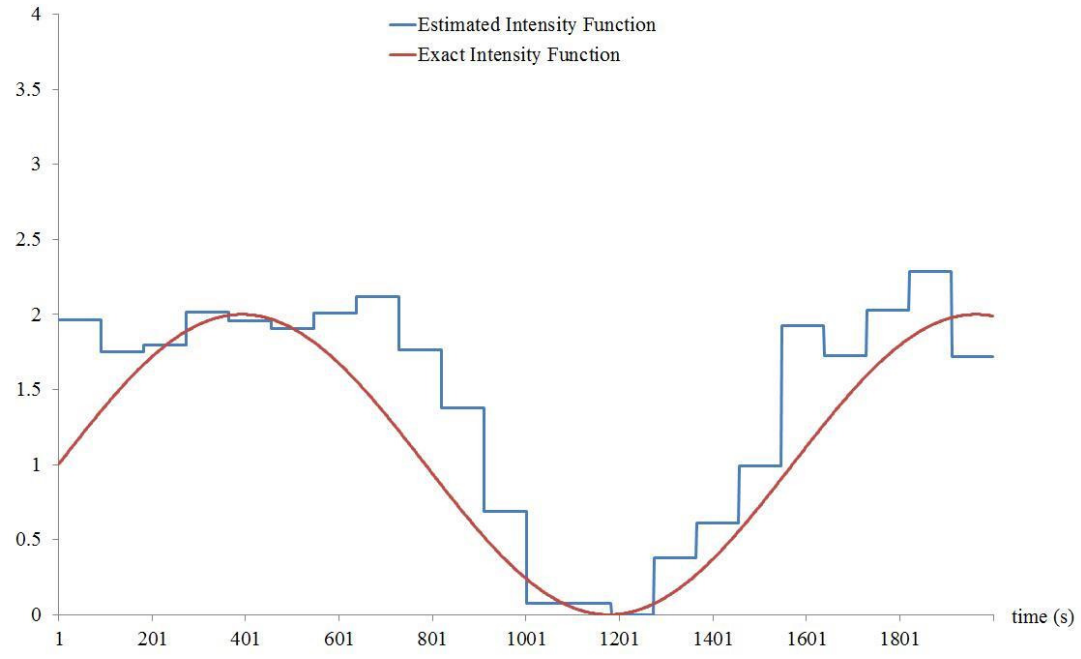

Figure 3.28: Estimation results for intensity function (n=22).

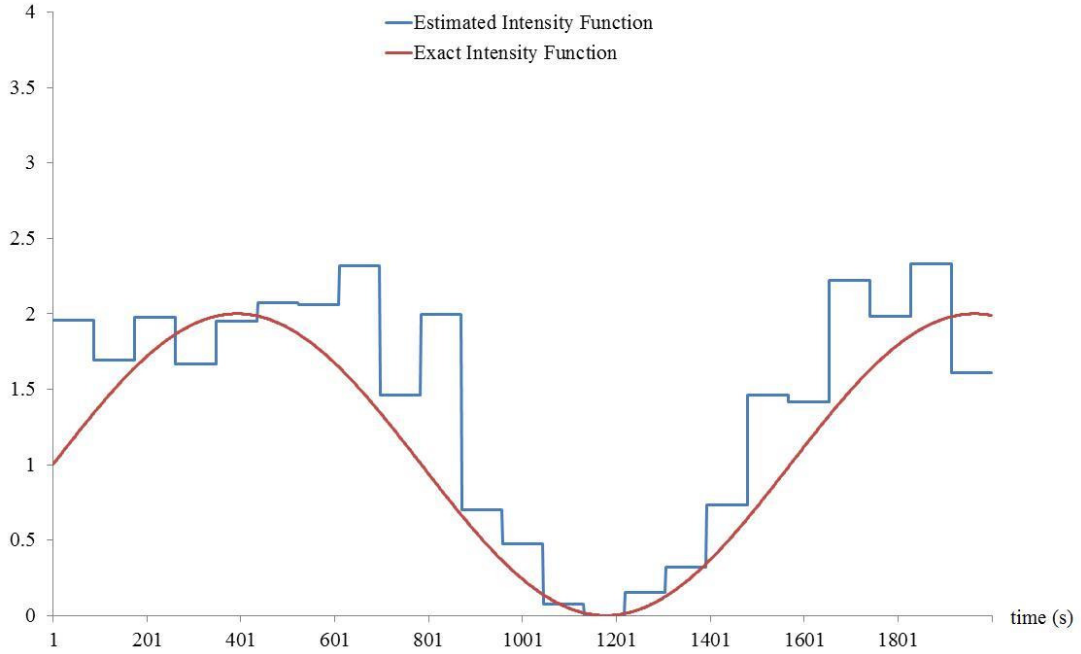

Figure 3.29: Estimation results for intensity function (n=23).

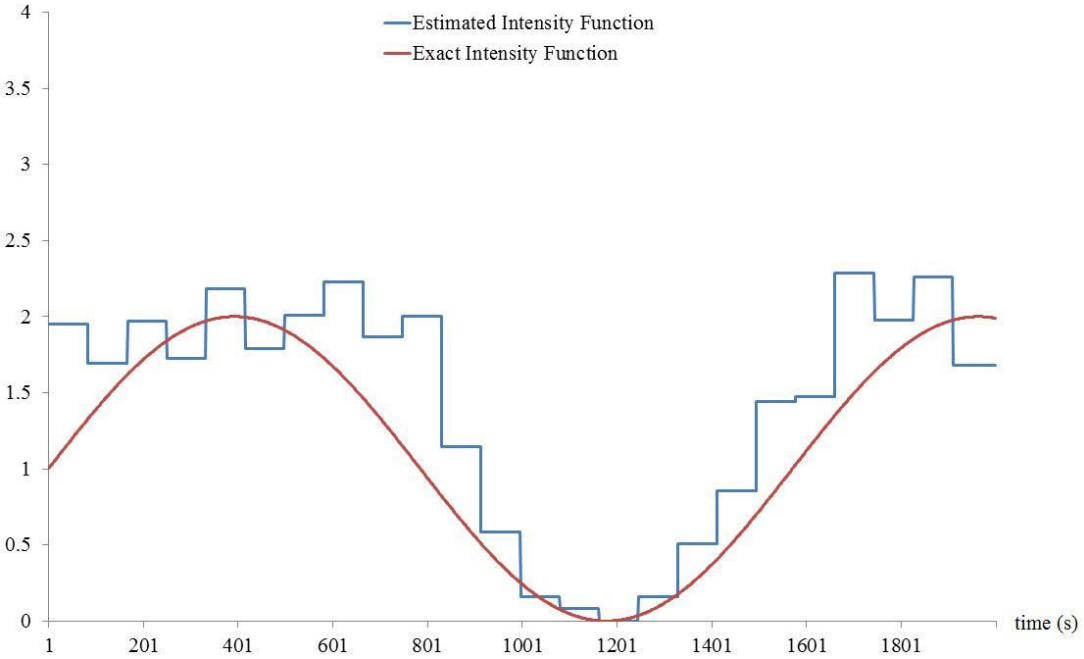

Figure 3.30: Estimation results for intensity function (n=24).

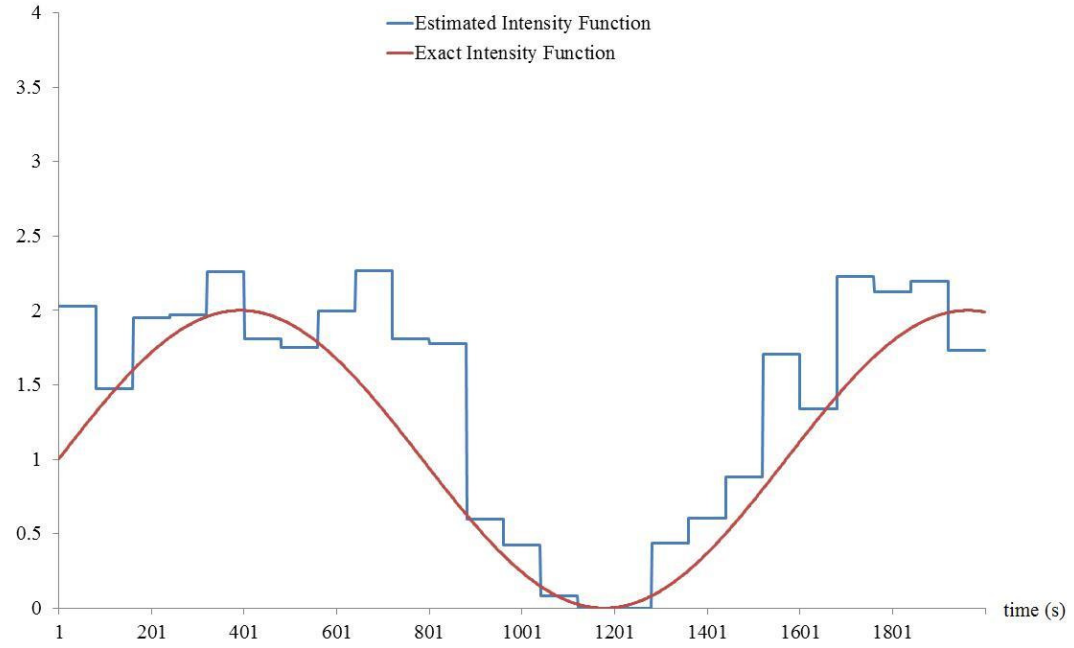

Figure 3.31: Estimation results for intensity function (n=25).

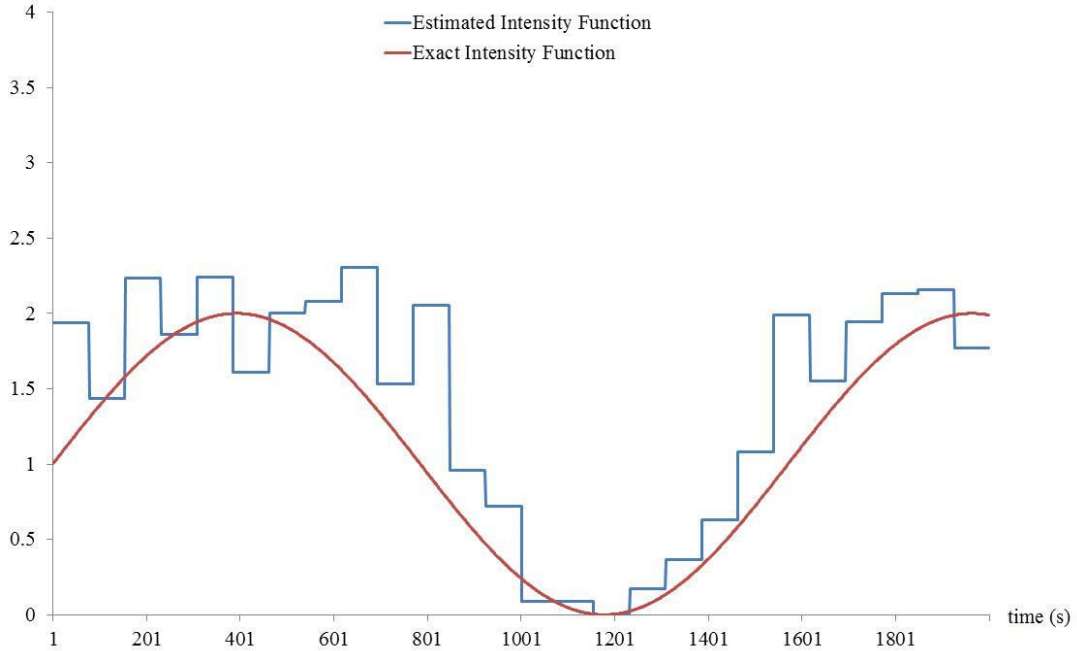

Figure 3.32: Estimation results for intensity function (n=26).

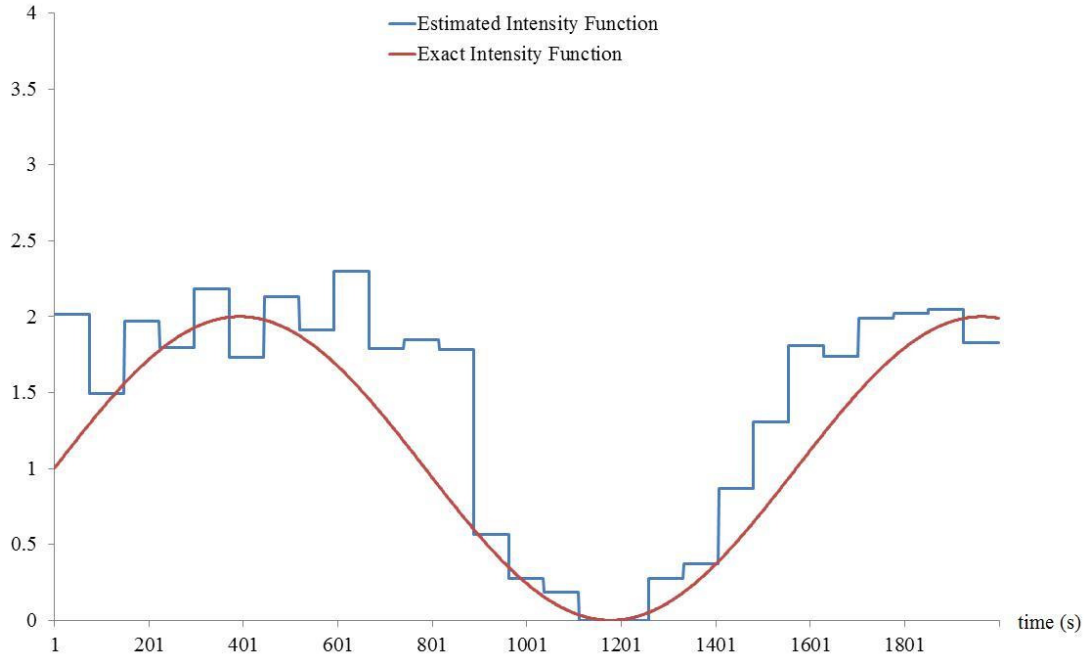

Figure 3.33: Estimation results for intensity function (n=27).

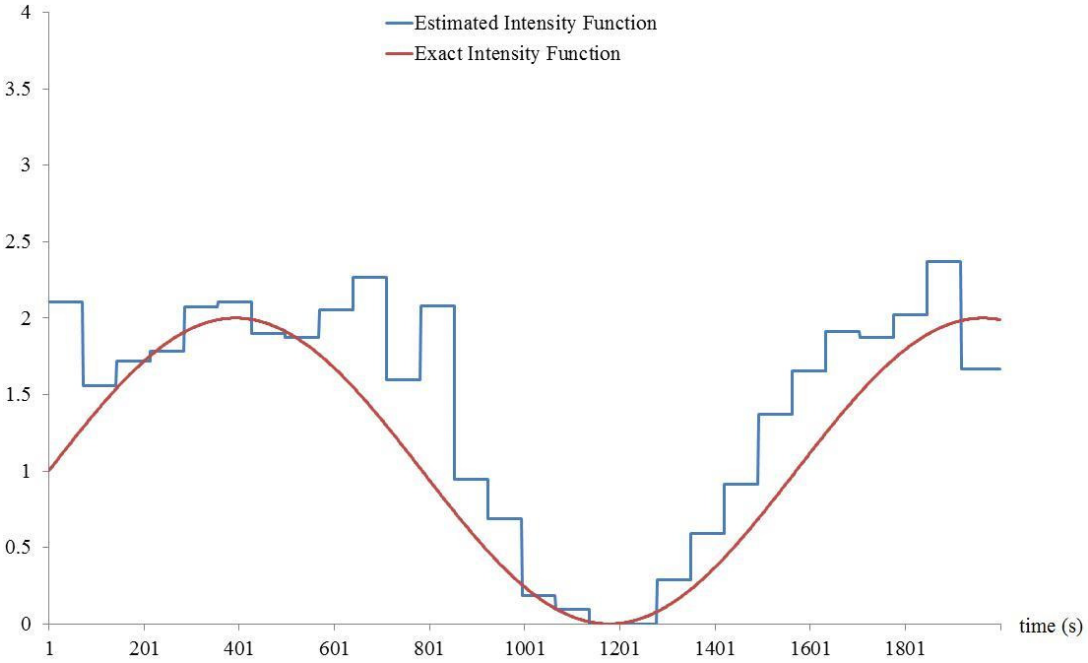

Figure 3.34: Estimation results for intensity function (n=28).

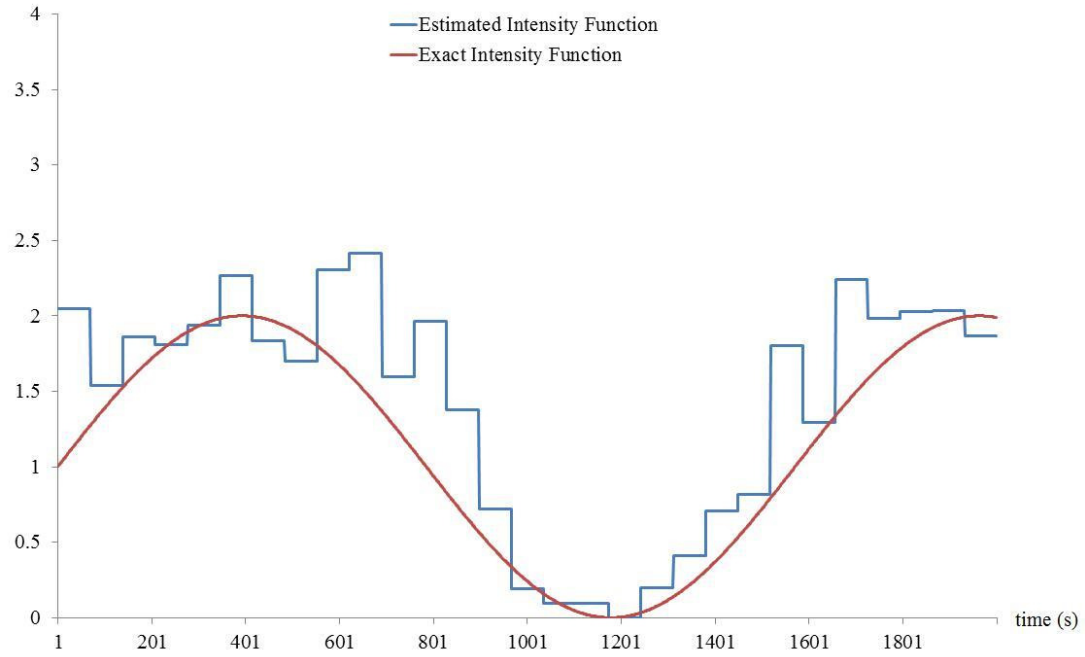

Figure 3.35: Estimation results for intensity function (n=29).

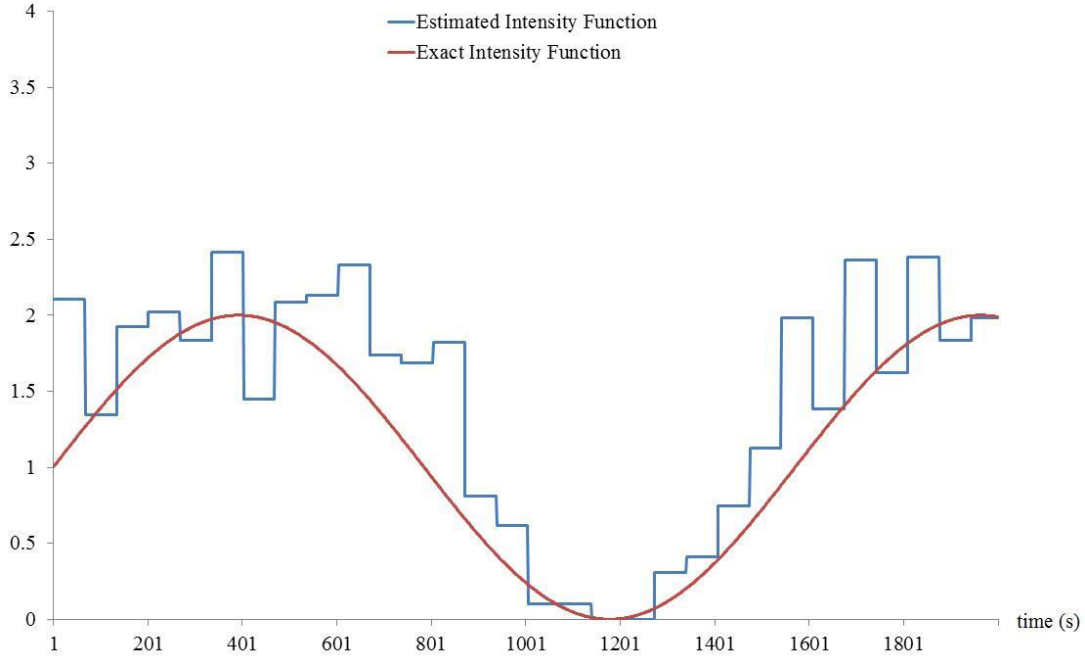

Figure 3.36: Estimation results for intensity function (n=30).

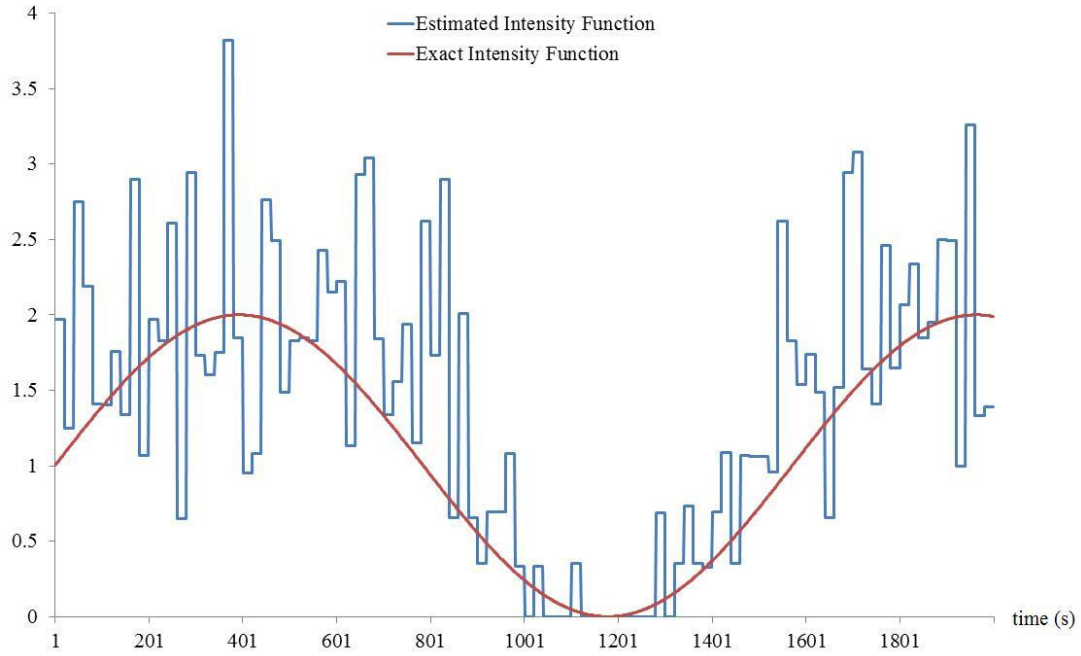

Figure 3.37: Estimation results for intensity function (n=100).

Chapter 4

Conclusion

In this dissertation, the author discusses two kinds of research works, MapReduce based computation of area skyline query for location recommendation in **Chapter 2**, and parameter estimation of queueing systems with utilization data in **Chapter 3**, respectively.

4.1 MapReduce Based Area Skyline Query

Location-based recommendation systems are essential for mobile applications. Many map-based applications can filter the uninteresting locations for mobile users, and recommend a small number of areas for users. In general, an interesting or good location should always be surrounded by excellent facilities, and should also be far away from unpreferable facilities. In the previous work [6], Annisa et al. has proposed area skyline queries based on a grid structure, called GASKY, to retrieve good locations on a map. However, the complexity and the average processing time of the GASKY algorithm is vast, since the Voronoi Diagram and $max - min$ table computation cost a lot.

In this work, we proposed a novel distributed area skyline query, which is based on the MapReduce framework. The novel algorithm, called MRGASKY, aims to improve the performance, i.e., the average processing time, comparing to the GASKY algorithm.

To confirm the effectiveness and efficiency of the MRGASKY algorithm, we conducted experiments. The experimental results illustrated that the average processing time of the MRGASKY algorithm tends to be stable with the number of grids, the number of facility types, and the number of objects increases. Overall, the proposed MRGASKY algorithm can handle the "big data" better and more effective than the GASKY algorithm.

4.1.1 Applications of the MRGAKSY Model

For applications, the MRGASKY algorithm can be utilized in some specific scenarios as follows.

- In the travel filed: The utilization of map-based applications on mobile devices is very common for travelers during a trip. In general, travelers would like to travel to places which have convenient transportations and are close to famous sightseeing spots. Also, the places should be far from the pollution areas and wildernesses. In other words, a good location should be close to preferable facilities and be far from unpreferable facilities according to user preferences. The proposed MRGASKY algorithm can recommend such good locations to the tourists of mobile devices in a short time.

- In the business field: It is a typical case for people in business, who want to find some good locations for the company building. For example, a real estate developer would like to search for an area to build a community. In this case, the great area should be close to some convenient places, such as bus/train stations, malls, and schools. Moreover, the desirable area should also be far away from some unpopular places, such as pollution areas, wildnesses, and noisy factories. The proposed distributed MRGASKY algorithm can also be applied into the business field, to help such the real estate developers find the potential areas on a map. Also, the MRGASKY algorithm can help the company reduce the survey cost in general.

4.1.2 Contributions of MRGAKSY Model

The main contributions of this work are as follows.

1. In our previous work, we have proposed the algorithm for area skyline query computation. In this work, to reduce the high cost on complexity and time processing, we propose a novel distributed algorithm for the area skyline query.

2. We propose a MapReduce-based area skyline query computation, which can reduce the complexity, and efficiently reduce the cost of time processing.

3. We conduct an extensive performance evaluation, which shows the high efficiency and scalability of our proposed algorithm.

4.1.3 Future Direction

In the future, we will take some non-spatial properties, such as the prices of the areas, the population densities of the regions into account. Also, we will consider the $k-$dominant problem, since the dimensions of data are more and more sophisticated in the age of "big data." Moreover, we would like to apply our algorithm in the map applications to help more people to make location recommendations.

4.2 Parameter Estimation of Queueing Systems

Statistical inference of queueing theory plays a vital subject in many fields, such as the performance evaluation of traffic, the performance evaluation of computer systems, etc. For example, the performance evaluation of traffic can alleviate traffic pressure effectively, and help cities to carry out road planning and construction. Also, the performance evaluation of computer systems can help the designers determine the system configuration in the system design phase. For example, We can determine the number of CPU from the performance evaluation of computation cost. From the evaluation of storage, we can determine

the memory size. However, with the advancement of computer technology, a tremendous amount of new data is generated every day. The improvement the computing power and computing capacity of computers are not economical and unrealistic sometimes. As a way to solve the problem, the recent design of system architecture tends to combine existing systems as models, which is called the system of systems (SoS) [23]. The integration of SoS can effectively improve the utilization of computer systems, and can also reduce economic consumption. So, performance evaluation of the integrated systems, such as SoS, becomes more and more critical in recent years.

In this work, the author generates a novel queueing system to estimate the arrival process with utilization data. In particular, since the Poisson distribution cannot fit the real-world situations, we generate the queueing system with the arrival process following an NHPP. An NHPP can model the dynamical changing process of arrival process better than an HPP. However, since the intensity function of an NHPP is a function of time t, it is more complicated and challenges to estimate the intensity function from such a continuous process.

To simplify the problem, the author made an approximation of the NHPP arrival input. After the approximation, the author transformed the continuous arrival process into a series of discrete piecewise HPPs.

On the other hand, the author defined that utilization is a fraction of busy time at a fixed time interval. The definition illustrated that there existed the observed time intervals and unobserved time intervals of the utilization data. In other words, utilization data is a kind of incomplete data. For the estimation of the intensity function, the author proposed to use the MLE via the EM algorithm.

In the experiments, the author first generated the simulated utilization data for an $M_t/M/1/k$ queueing system. Then, the author investigated the effectiveness of the proposed approach. Furthermore, the author utilized the real CPU utilization data again, to

estimate the intensity function of the two servers. Besides, the author used the average response time to examine the performance evaluation of an integrated system.

4.2.1 Applications of the $M_t/M/1/l$ Queueing System

For applications, the proposed $M_t/M/1/l$ queueing system can be utilized in some specific scenarios, such as the computer design phase.

In a computer system, CPU utilization is a widespread data type and can reflect the usage of the computing resource. CPU utilization data can be collected easily from a computer system. However, we cannot know the information on the arrival time and the service time of CPU-tasks from CPU utilization data directly. The observations of the arrival and service for the estimation are quite limited in this case. The proposed queueing system can effectively resolve the above problem. We can estimate the scalable arrival process only from CPU utilization data.

4.2.2 Contributions of the $M_t/M/1/l$ Queueing System

The main contributions of this work are as follows:

1. We mainly estimate the arrival process of a queueing system from utilization data. To the best of our knowledge, this is the first paper to estimate the arrival process by using utilization data. In general, utilization data is defined as a fraction of the busy period in a fixed time interval. We cannot get exact arrival time and service time from utilization data. It is challenging for us to generate a novel model to estimate parameters from utilization data.

2. To better model the arrival process in the real world, we consider Non-homogeneous Poisson process as the input. The queueing model is noted as $M_t/M/1/K$. Since the intensity function of the NHPP is a function of rates, which is respected with time $\lambda(t)$, the NHPP is more realistic and complex than the conventional Poisson process.

To simplify the problem, we make an approximation of the NHPP and transform the dynamical complex arrival process to a series of Homogeneous Poisson Process (HPP).

3. We aim to estimate the arrival process of the $M_t/M/1/K$ queueing system from incomplete utilization data. To make accurate estimations, we propose to apply the Maximum Likelihood Estimation (MLE) via the Expectation-Maximization (EM) algorithm.

4. In the experiments, we conduct an extensive performance evaluation based on simulated utilization data and real CPU utilization data respectively to verify the proposed model.

4.2.3 Future Direction

In the future, we will take the Maximum A Posterior (MAP) into account. The estimation results of the intensity function in this work tend to be changed a little violently from the previous experiments.

To resolve the problem, we will consider applying the MAP algorithm into the MLE. Also, we will generate a hierarchical Bayesian estimation (HBE) to further consideration.

Bibliography

[1] O. Celma, P. Lamere, "Music recommendation tutorial," *In International Conference on Music Information Retrieval (ISMIR 2007)*, Vienna, 2007.

[2] S. Borzsonyi, D. Kossmann, K. Stocker, "The skyline operator," *In Proceedings of the 17th International Conference on Data Engineering (ICDE)*, pp. 421-430, Heidelberg, Germany, April 2-6, 2001.

[3] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, "Skyline with presorting," *In Proceedings of the 19th International Conference on Data Engineering (ICDE)*, pp. 717-719, Bangalore, India, March 5-8, 2003.

[4] K. L. Tan, P. K. Eng, B. C. Ooi, "Efficient progressive skyline computation," *In Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pp. 301-310, Rome, Italy, September 11-14, 2001.

[5] T. Xia, D. Zhang, Y. Tao, "On skylining with flexible dominance relation," *In Proceedings of the 24th International Conference on Data Engineering (ICDE)*, pp. 1397-1399, Cancun, Mexico, April 7-12, 2008.

[6] A. Zaman, Y. Morimoto, "Area skyline query for selecting good locations in a map," *J. Inf. Process*, pp. 946-955, 2016.

[7] C. Y. Chan, H. Jagadish, K. L. Tan, A. K. Tung, Z. Zhang, "On high dimensional skylines," *In Proceedings of the 10th International Conference on Extending Database Technology*, pp. 478-495, Munich, Germany, March 26-31, 2006.

[8] C. Y. Chan, H. Jagadish, K. L. Tan, A. K. Tung, Z. Zhang, "Finding k-dominant skylines in high dimensional space," *In Proceedings of the International Conference on Management of Data and Symposium on Principles Database and Systems*, pp. 444-457, Chicago, IL USA, June 27-29, 2006.

[9] X. Lin, Y. Yuan, Q. Zhang, Y. Zhang, "Selecting stars: The k most representative skyline operator," *In Proceedings of the 23rd International Conference on Data Engineering*, pp. 86-95, Istanbul, Turkey, April 11-15, 2007.

[10] M. Sharifzadeh, C. Shahabi, "The spatial skyline queries," *In Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, pp. 751-762, Seoul, Korea, September 12-15, 2006.

[11] K. Kodama, Y. Iijima, X. Guo, Y. Ishikawa, "Skyline queries based on user locations and perferences for making location-based recommendations," *In Proceedings of the 19th International Workshop on Location Based Social Networks (LBSN)*, pp. 9-16, Washington, DC, USA, November 3, 2009.

[12] M. Arefin, J. Xu, Z. Chen, Y. Morimoto, "Skyline query for selecting spatial objects by utilizing surrounding objects," *J. Softw.*, pp. 1742-1749, 2013.

[13] G.W. You, M. W. Lee, H. Im, S.W. Hwang, "The farthest spatial skyline queries," *Inf. Syst.*, pp. 286-301, 2013.

[14] Y.W. Lin, E. T. Wang, C. F. Chiang, A. L. P. Chen, "Finding targets with the nearst favor neighbor and farthest disfavor neighbor by a skyline query," *In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC)*, pp. 821-826, Gyeongju, Korea, March 24-28, 2014.

[15] M.A. Siddique, A. Zaman, Y. Morimoto, "A method for selecting desirable unfixed shape areas from integrated geographic information system," *In Proceedings of the 4th International Congress on Advanced Applied Informatics*, pp. 195-200, Okayama, Japan, July 12-16, 2015.

[16] K. Hose, A. Vlachou, "A survey of skyline processing in highly distributed environments," *Int. J. Very Large Data Bases*, pp. 359-354, 2012.

[17] B. Zhang, S. Zhou, J. Guan, "Adapting skyline computation to the mapreduce framework: algorithms and experiments," *In Proceedings of the 16th International Conference on Database Systems for Advanced Applications*, pp. 403-414, Hong Kong, China, April 22-25, 2011.

[18] L. Chen, K. Hwang, J. Wu, "MapReduce skyline query processing with new angular partitioning approach," *In Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum'*, pp. 2262-2270, Shanghai, China, May 21-25, 2012.

[19] D. Papadias, Y. Tao, G. Fu, B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, pp. 41-82, 2005.

[20] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal, A. Abbadi, "Parallelizing skyline queries for scalable distribution," *In Proceedings of the 10 International Conference on Extending Database Technology*, Munich, Germany, March 26-31, 2006.

[21] W. Wang, J. Zhang, M. T. Sun, W. S. Ku, "Efficient parallel spatial skyline evaluation using MapReduce," *In Proceedings of the 20th International Conference on Extending Database Technology*, Venice, Italy, March 21-24, 2017.

[22] D. Man, K. Uda, Y. Ito, K. Nakano, "Accelerating computation of Euclidean distance map using the GPU with efficient memory access," *Int. J. Parallel Emergent Distrib. Syst.*, pp. 383-406, 2012.

[23] S. Popper, S. Bankes, R. Callaway and D. DeLaurentis, "System-of-systems symposium: report on a summer conversation," *Potomac Institute for Policy Studies*, Arlington, VA, July 21-22, 2004.

[24] D. Thiruvaiyaru and I.V. Basawa, "Estimation for a class of simple queueing networks," *Queueing Systems*, 9, 301-312, 1991.

[25] I.V. Basawa, "Maximum likelihood estimation for single server queues from waiting time data," *Queueing systems*, 24, 155-167, 1996.

[26] K. S. Trivedi, *"Probability and statistics with reliability, queueing, and computer sciences applications,"*, John Wiley & Sons, New York, 2nd edition, 2001.

[27] S. Dharmaraja and K. S. Trivedi and D. Logothetis, "Performance modeling of wireless networks with generally distributed handoff interarrival times," *Computer Communications*, 26(15):1747-1755, 2003.

[28] M. Stasiak, M. Gabowski, A. Wisniewski and P. Zwierzykowski, *Modelling and dimensioning of mobile networks: from GSM to LTE*, Wiley, 2011.

[29] J. Medhi, *Stochastic models in queueing theory (Second Edition)*, Academic Press. 2003.

[30] U. N. Bhat, *An Introduction to queueing theory-modeling and analysis in applications*, Birkhauser Boston, 2008.

[31] AB. Clarke, "Maximum likelihood estimaties in a simple queue," *Ann. Math. Statist.*, 28, 1036-1040, 1957.

[32] D. Thiruvaiyaru and I.V. Basawa, "Empirical Bayes estimation for queueing systems and networks," *Queueing Systems*, 11, 179-202, 1992.

[33] I. Basawa,"Maximun likelihood estimation for single server queues from waiting time data," *Queueing systems*, 24, 155-167, 1996.

[34] M. J. Fischer, D. Gross, D. M. Bevilacqua Masi, et al., "Analyzing the waiting time process is internet queuing systems with the transform approximation method," *The Telecommunication Review*, 12:21-32, 2001.

[35] I. Shams and K. Shahanaghi, "Analysis of nonhomogeneous input data using likelihood ratio test," *in Proc IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Hong Kong, 2009.

[36] R. P. Trinatha, R. K. Srinivasa and K.V.V.S. Reddy, "Performance of non-homogeneous communication with Poisson arrivals and dynamic bandwidth allocation," *International Journals of Systems, Control and Communication (IJSCC)*, 4(3): 164-182, 2012.

[37] J. V. Ross, T. Taimre and P. K. Pollett, "Estimation for queues from queue length data," *Queueing Sytems*, 131-138, 2007.

[38] H. Q. Liu, W. L. Liang, L. Rai, K. Teng and S. L. Wang, "A real-time queue length estimation method based on probe vehicles in CV environment," *IEEE Access*, 20825-20839, 2019.

[39] H. Takagi, Y. Kanai, and K. Misue, "Queueing network model for obstetric patient flow in a hospital," *Health Care Management Science*, 20, 433–451, 2017.

[40] N. Gans, G. Koole, and A. Mandelbaum, "Telephone call centers: Tutorial, review, and research prospects," *Manufacturing & Service Operations Management*, 5, 79-141, 2003.

[41] E. Kozan, E, "Comparison of analytical and simulation planning models of seaport container terminals," *Transportation Planning and Technology*, 20, 235-248, 1997.

[42] M. H. Rothkopf and S. S. Oren, "A closure approximation for the non-stationary M/M/S queue," *Management Science*, 25 (6): 522-534, 1979.

[43] D. P. Heyman and W. Whitt, "The asymptotic behavior of queues with time-varying arrival rates," *Journal of Applied Probability*, 21: 143-156, 1984.

[44] L. Green, P. Kolesar and A. Svoronos, "Some effects of non-stationarity on multi-server Markovian queueing systems," *Operations Research*, 39 (3): 502-511, 1991.

[45] A. P. Pant and R. P. Ghimire, "$M_t/M/1$ queueing system with sinusoidal arrival rate," *Journal of Institute of Engineering*, 11 (1): 120-127, 2015.

[46] V. E. Benes, "A sufficient set of statistics for a simple telephone exchange model," *Bell System Technology Journal*, 36, 939-964, 1957.

[47] L. H. Wang, S. C. Chen and J. C. Ke, "Maximum likelihood estimates and confidence intervals of an M/M/R queue with heterogeneous servers," *Mathematical Methods Operational Research*, 63, 371-384, 2006.

[48] C. Amit and B. Arpita, "Statistical inference on traffic intensity in an M/M/1 queueing system," *International Journal of Management Science and Engineering Management*, vol. 13, 2018.

[49] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc. B*, vol. B-39, pp. 1-38, 1977.

[50] C. F. J. Wu, "On the convergence properties of the EM algorithm," *Ann. Stat.*, vol. 11, pp. 95-103, 1983.

[51] T. Rydén, "An EM algorithm for estimation in Markov-modulated Poisson processes," *Computational Statistics Data Analysis*, 21(4), 431-447, 1996.

[52] I. V. Basawa, U. N. Bhat and J. Zhou, "Parameter estimation in queueing systems using partial information," *Statist. Probab. Lett.*, 2008.

[53] H Okamura, T. Dohi and K. S. Trivedi, "Markovian arrival process parameter estimation with group data," *IEEE/ACM Transactions on Networking*, 17(4), 1326-1339, 8 2009.

List of Referred Publications

Referred Journals

J-1 **Chen Li**, Annisa, Asif Zaman, Mahboob Qaosar, Saleh Ahmed and Yasuhiko Morimoto, *"MapReduce algorithm for location recommendation by using area skyline query"*, Algorithms, 2018, 11(12):191, Basel, Switzerland.

J-2 **Chen Li**, Hiroyuki Okamura and Tadashi Dohi, *"Parameter estimation of $M_t/M/1/K$ queueing systems with utilization data"*, IEEE Access, Vol 7, Pages 42664-42671, March 2019.

Referred International Conferences

C-1 **Chen Li**, Annisa, Asif Zaman and Yasuhiko Morimoto, *"MapReduce-based computation of area skyline query for selecting good locations in a map"*, Proceedings of 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, December 11-14, 2017, doi: 10.1109/BigData.2017.8258540.

Other Publications (not in dissertation)

Referred Journals

J-3 Saleh Ahmed, Mahboob Qaosar, Asif Zaman, Md. Anisuzzaman Siddique, **Chen Li** and Yasuhiko Morimoto *"Privacy aware MapReduce based multi-party secure skyline computation"* , Information, 2019, 10(3):119, Basel, Switzerland.

Referred International Conferences

C-2 **Chen Li**, Minjia He, Mahboob Qaosar, Saleh Ahmed and Yasuhiko Morimoto, *"Capturing temporal dynamics of users' preference from purchase history big data for recommendation system"*, Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, USA, December 10-13, 2018, doi: 10.1109/BigData.2018.8622411.

C-3 **Chen Li**, Xu Zhang, Mahboob Qaosar, Saleh Ahmed, Kazi Md. Rokibul Alam and Yasuhiko Morimoto, *"Multi-factor based stock price prediction using hybrid neural networks with attention mechanism"*, Proceedings of the 5th IEEE International Conference on Cloud and Big Data Computing (CBDCom 2019), Fukuoka, Japan, August 5-8, 2019.

C-4 Mahboob Qaosar, Saleh Ahmed, **Chen Li** and Yasuhiko Morimoto, *"Hybrid sensing and wearable smart device for health monitoring and medication: opportunities and challenges"*, Proceedings of the 2018 AAAI Spring Symposium Series, pages 269-274, March 2018.

C-5 Xu Zhang, **Chen Li** and Yasuhiko Morimoto, *"A multi-factor approach for stock price prediction by using recurrent neural networks"*, Proceedings of Bulletin of Networking, Computing, Systems, and Software (BNCSS), Vol 8, Number 1, pages 9-13, January 2019.

Non-Referred International Conferences

N-1 **Chen Li**, Chao Luo, Hiroyuki Okamura and Tadashi Dohi, *"A note on performance evaluation of system with CPU utilization data"*, IEICE-Assurance System conference, Hiroshima, Japan, 2015.

N-2 **Chen Li**, Annisa, Asif Zaman and Yasuhiko Morimoto, *"MapReduce-based computation of area skyline query for selecting good locations in a map"*, DEIM conference, Fukui, Japan, 2018.

N-3 Mahboob Qaosar, Saleh Ahmed, **Chen Li** and Yasuhiko Morimoto, *"Privacy-preserving multi-party skyline computation framework based on homomorphic encryption with data perturbation and anonymization"*, the 4th International Symposium on Big Data Analytics in Science and Engineering (BASE 2019), Fukushima, Japan, 2019.