S-Que : Multi-class M/M/c based Smart Queues Ticketing System

Case Study : Bank Queuing System

Aris Prawisudatama
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
soedomoto@gmail.com

***Abstract*—The queue is the duration of time that someone has to spend waiting until finally gets the service, and usually characterized by using ticket. The conventional queuing system, which is commonly used at many service centers, only gives the consumer a queue numbers without providing any estimation about when the service will be given. Multi-class M/M/c, which is extends of M/M/c that is proposed by AK. Erlang, can be used to predict waiting time in the queue. On the other hand, SOA approach that is implemented using RESTful Web-service is proven to be flexible and scalable, so that can be used to monitor queue condition in real-time. This paper propose design and implementation of queuing system equipped with service time prediction and online checking using QR code, using Multi-class M/M/c theorem and SOA approach.**

Keywords—Queue; Ticket; Erlang; M/M/c; Multi-class; Multi-class M/M/c; Service Oriented Architecture; Service; REST; Python; QR Code

I. INTRODUCTION

The queue is a common situation that can be found almost every day and has become the part of our daily life activities such as purchasing a train ticket, saving money at the bank, creating official documents at government offices, and even eating at the restaurant. Generally, the queue can be described as the duration of time that someone has to spend waiting until he finally gets the service [1]. It is absolutely annoying, especially for those who have tight schedules. Sometimes, the queue could last all day without any prediction about when the service will be provided. By using the queuing theory, we could design a model to predict the queue length and the waiting time to get a service [2].

Queuing theory has been the subject of many researches for quite a long time. A mathematician from Denmark, A.K. Erlang, used a special class from gamma random variable, often called as Erlang-k random variable, in his research about the delay in phone traffic [2]. Now, the queue has also become a research subject in engineering, production, biological engineering, communication, computer system design, and other fields [3].

There are several theories related to queue:

1. M/D/1 used in cases where the arrivals are random and the service is deterministic and using a single channel.
2. M/M/1 used in cases with random arrivals, random services, and use single-channel service.
3. M/M/c, used in cases with random arrivals, random services, and multi-channel services.

The queue system that is implemented in bank service has more than one channel that can be used randomly by the customers. Thus, the most suitable solution to handle the queue problem is by implementing the M/M/C theory.

II. THEORETICAL BACKGROUND

A. M/D/1

In queueing theory, an M/D/1 queue represents the queue length in a system having a single server, where arrivals are determined by a Poisson process and job service times are fixed (deterministic). Agner Krarup Erlang first published on this model in 1909, starting the subject of queuing theory [4]. An M/D/1 queue is a stochastic process whose state space is the set {0,1,2,3,...} where the value corresponds to the number of customers in the system, including any that is currently in service.

- Arrivals occur at rate λ according to a Poisson process and move the process from state i to i + 1.
- Service times are deterministic time D (serving at rate $\mu = 1/D$).
- A single server serves customers one at a time from the front of the queue, according to a first-come, first-served discipline. When the service is complete, the customer leaves the queue and the number of customers in the system reduces by one.
- The buffer is of infinite size, so there is no limit on the number of customers it can handle.

B. M/M/1

An M/M/1 queue represents the queue length in a system having a single server, where arrivals are determined by a Poisson process and job service times have an exponential distribution. The model name is written in Kendall's notation [5].

- Arrivals occur at rate λ according to a Poisson process and move the process from state i to i + 1.
- Service times have an exponential distribution with rate parameter μ in the M/M/1 queue, where 1/μ is the mean service time.
- A single server serves customers one at a time from the front of the queue, according to a first-come, first-served discipline. When the service is complete the customer leaves the queue and the number of customers in the system reduces by one

The model can be described as a continuous time Markov chain with transition rate matrix

$$\begin{bmatrix} \lambda & 0 & 0 & \cdots & 0 \\ 0 & \mu+\lambda & 0 & \cdots & 0 \\ 0 & 0 & \mu+\lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & \cdots & \mu \end{bmatrix}$$

C. M/M/c

The M/M/c queue (or Erlang–C model) is a multi-server queuing model [6], where there are c servers and job service times that are exponentially distributed [7]. M/M/c is a generalization of the M/M/1 queue which considers only a single server. The characteristics of M/M/c are:

- Arrivals occur at rate λ according to a Poisson process and move the process from state i to i+1.
- Service times have an exponential distribution with parameter μ in the M/M/c queue.
- There are c servers, which serve from the front of the queue. If there are less than c jobs, some of the servers will be idle. If there are more than c jobs, the jobs queue in a buffer.
- The buffer is of infinite size, so there is no limit on the number of customers it can contain.

D. Multi-class M/M/c

The simple M/M/c queue is like using c servers at the same service rate. In fact, for example at bank service, there are several service desks that handle different service types, so that each service type could have different service rate. This condition is called multi-class services.

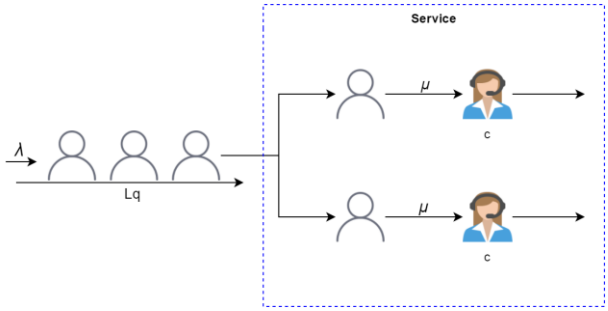

Fig. 3. M/M/c Queuing Model

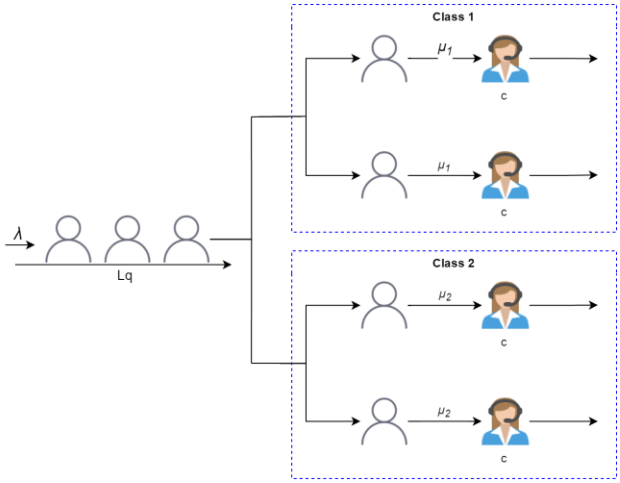

Fig. 1. Multi-class M/M/c Queuing Model with Two Servers

Wang, et.al, conducted a research to discover a model for queue handling in Two Priority Class [8]. Two class M/M/c service rate at state (q1, q2) is μ1 min(q1, c); independent of q2. Thus, the distribution of Class-1 jobs' sojourn time is [9]:

$$P\{S_1 < t\} = 1 - e^{-\mu_1 t} - \frac{(e^{-(c\mu_1-\lambda_1)t} - e^{-\mu_1 t})}{1-(c-\frac{\lambda_1}{\mu_1})} \frac{\lambda_1^c}{\mu_1^c c!}((1-\frac{\lambda_1}{c\mu_1})\sum_{i=0}^{c-1}\frac{\lambda_1^i}{\mu_1^i i!} + \frac{\lambda_1^c}{\mu_1^c c!})^{-1}$$

E. Service Oriented Architecture

Service Oriented Architecture (SOA) is a way of designing software as interoperable software services. A service is a software component that is available over a network and can act either as a client or a server or be composed by combining other services [10]. Services in SOA are lightweight, loosely-coupled and communicate using messages over well-defined interfaces. Today, most SOAs are built using Web Services.

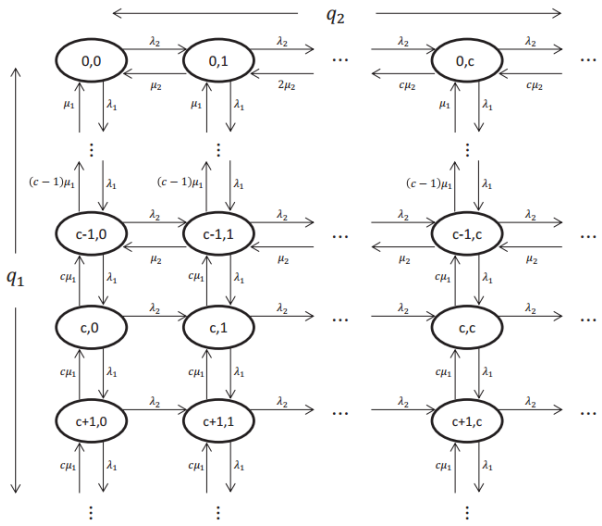

Fig. 2. MC of the M/M/c queue with two priority classes

Because SOAs are built on light-weight and interoperable services, they are inherently flexible and scalable. The flexibility comes from the fact that services can be replaced according to changing needs and that new services can be created by combining other services. SOAs scale well thanks to their parallel nature as services can easily be duplicated or migrated as needed [11].

F. SOAP and REST

A SOA can be rendered through the use of any service-based technology, such as SOAP or REST. Compared to each other, SOAP is more mature and standardized, widely supported and transport-protocol independent while REST gives developers larger freedom of choice and is easy to get started with thanks to the agility of the design and the lightweight approach. REST also works well for applications with a Web Interface since the basic REST operations are supported by any Web Browser. However, REST lacks standards for security, quality of service (QoS), etc and support for business automation languages whereas SOAP is highly standardized and supported. A more in-depth discussion of these technologies is given by Pautasso et al [12].

From a performance point of view, REST has better cache support, lightweight requests and responses as well as easier response parsing. REST also supports a large number of parallel clients and servers and reduces network traffic due to less overhead [6].

III. S-QUE SYSTEM DESIGN

S-Que is a system that is used to manage the queuing service. S-Que is an answer to the conventional queuing system that is commonly used at many service centers. The conventional system only gives the consumer a queue numbers without providing any estimation about when the service will be given. This kind of system is really inconvenient, especially for the busy consumers, because they have to wait in an uncertain time and cannot leave the service centers until their number is called.

On the other hand, S-Que gives a certainty to the consumer trough two channels: estimation and real-time. Thus, when a consumer takes his queue number, he is also given an estimation about when he will be served. Moreover, the consumer could also check the current queue status at any time through the website. S-Que adopted the queue theory of Multi-class Erlang-C (M/M/c) model that has been developed by Wang et al [8] as well as the SOA approach that has been implemented in RESTful Web Service.

A. Use Case

There are three types of the system users: customer, service officer, and administrator. Customers are people who get in the queue line, get the ticket, and frequently check the current queue status. Service officers are people who give service to the customers. The administrator is someone who manages the system, including add, edit, and delete service channels and service officers. The Use case for the system is shown in Fig. 4.

First, the customer needs to get the queue number. Before sending the response, the system will assign a number and calculate the estimation of the service time based on the number of queues and available service counters. Then, the queuing number, along with the service time estimation, will be printed on the queuing ticket. This process is conducted by the queuing server that communicates with the client (in the form of a button) using web service technology.

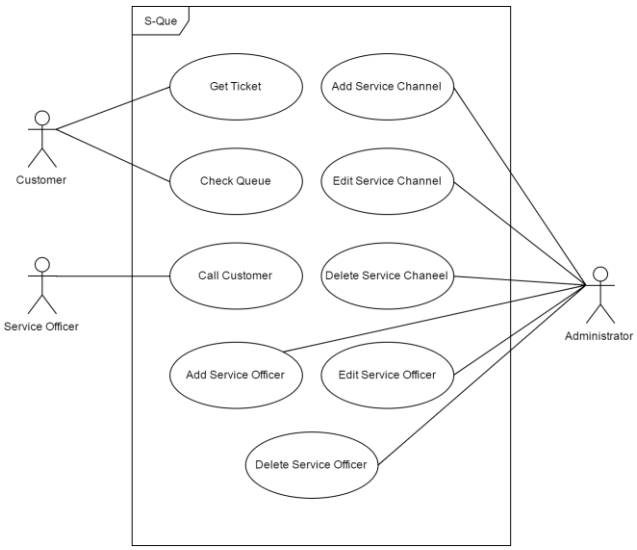

Fig. 4. S-Que Use Case

The customer could also check the real-time queuing status using a browser installed on a computer or smartphone. In the meantime, the service office will call the customers using the client button (the start and stop button). The start and end time of the service are recorded to count the average service time for each service class.

B. SOA Design

The use case in Fig. 4 will be decomposed into Services and Components that will be used in the system. Fig. 5 illustrates the relationship between services, components, and functional components. Services can be distinguished into six

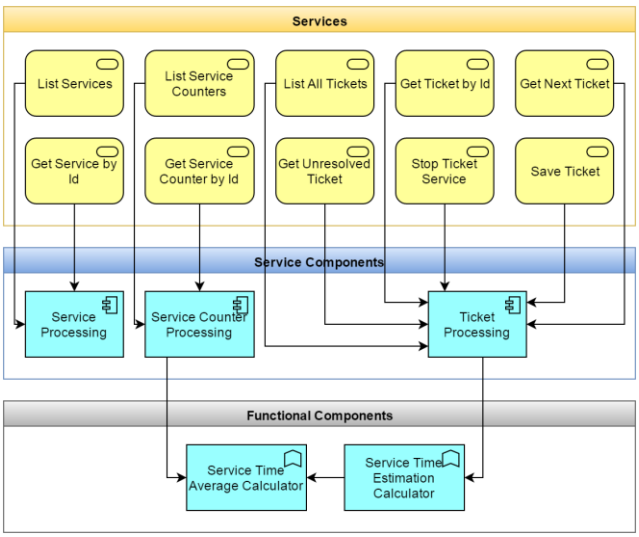

Fig. 5. SOA Architecture Layer

types which are grouped into three modules: service, counter, and ticket. On the other hand, the component provides functionalities for every service module such as connection to the database, consume other services, as well as computing a logic calculation.

Functional components have two functions that are service time average calculator and service time estimation calculator. Service time average is used to calculate the average time for each service type based on available historical data, while the service time estimation calculator is used to predict when the service will be given based on the multi-class M/M/c method.

C. *User Interface*

While it is possible for users to use the REST service interface to work with the S-Que System, we think most people would appreciate not having to interact directly with Web Services. We, therefore, provide a Web Application on top of the REST interface which clients and reviewers use to interact with the system.

IV. Implementation and Deployment

A. *Web Service*

RESTful Web Service in the S-Que is implemented using Python programming language and Flask framework. Flask framework provides basic methods for HTTP communication such as POST, GET, PUT, PATCH, and DELETE.

B. *All S-Que User Interface*

The user interface is implemented as a Web-based interface. The web-based interface is aimed to be simpler, platform independent, and responsive. The user interface is built on top of twitter bootstrap library.

C. *S-Que Ticket*

The S-Que ticket is still using a manual system. It means the customers must be present at the service location to push the "service-types" button, which cannot be done remotely. Physically, the ticket is printed in a postcard size and contains more information including customer queue number, the current queue number (the queue number that is being served right now), the type of service needed, service time estimation, and QR code to check the current queue status in the real time.

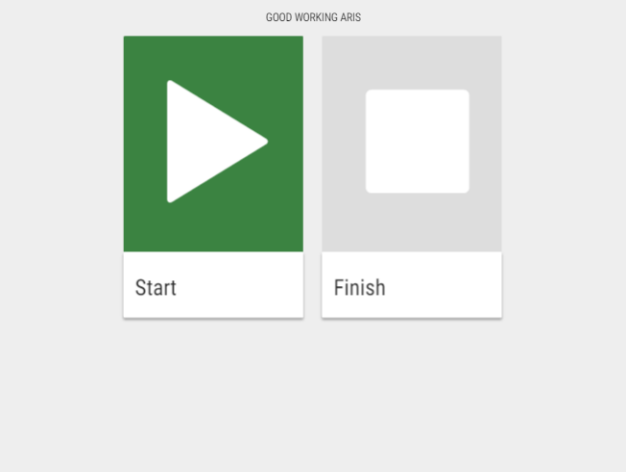

Fig. 7. Screenshot of S-Que Service Counter on Tablet Screen

```
@self.route('/api/list/', methods=['GET'])
def api_list():
   tickets = Ticket.query.all()
   return Response(json.dumps([i.serialize for i
        in tickets]),
        mimetype='application/json')

@self.route('/api/id/<id>/', methods=['GET'])
def api_by_id(id):
   ticket =
        Ticket.query.filter_by(ticket_id=id)\
      .first()
   return Response(json.dumps(ticket.serialize),
        mimetype='application/json')

@self.route('/api/next/<counter>',
        methods=['GET'])
def api_next_ticket(counter):
   next_ticket = Ticket.query.filter_by(\
        service_date=datetime.datetime.now()\
        .date(), service_start_time=None)\
        .order by(Ticket.ticket order.asc())\
        .first()

   if next_ticket:
      next ticket.service counter = int(counter)
      next ticket.service start time = datetime\
        .datetime.now()

      db.session.commit()

      return Response(json.dumps(next_ticket\
        .serialize if next_ticket else None),\
        mimetype='application/json')

@self.route('/api/unresolved/<counter>',\
   methods=['GET'])
def api_unresolved_ticket(counter):
   unresolved_ticket =
        Ticket.query.filter(and_(\
      Ticket.service start time!=None,\
      Ticket.service finish time==None,\
      Ticket.service_date==datetime.datetime\
        .now().date(),\
      Ticket.service counter==int(counter)\
   )).first()

   return Response(json.dumps(unresolved_ticket\
      .serialize if unresolved_ticket else None)\
      , mimetype='application/json')

@self.route('/api/stop/<id>', methods=['GET'])
def api_stop_ticket(id):
   finished ticket = Ticket.query.filter by(\
      ticket id=id).first()

   if finished_ticket:
      finished_ticket.service_finish_time =\
        datetime.datetime.now()
      db.session.commit()

   return Response(json.dumps({'success' :\
      True}), mimetype='application/json')
```

Fig. 6. Code Snippet for Python Flask Web Service

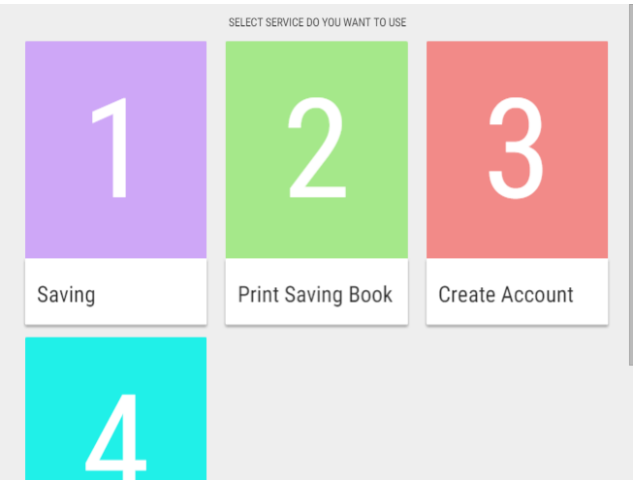

Fig. 8. Screenshot of S-Que Ticketing Service on Tablet Screen

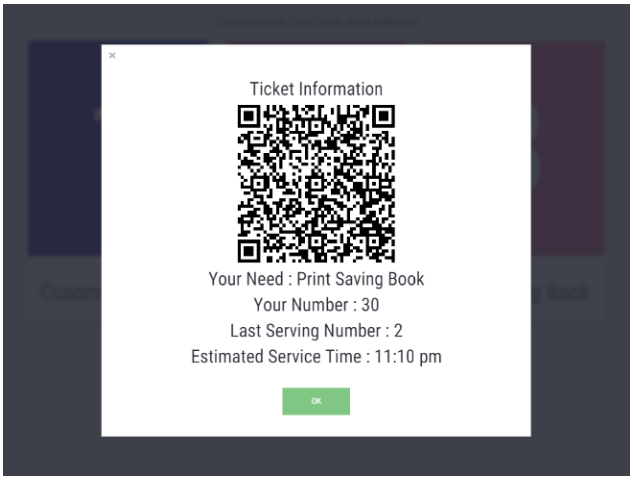

Fig. 10. Screenshot of S-Que Ticket Information after Selecting Service on Tablet Screen

Fig. 9. Conventional vs S-Que Queuing Ticket

REFERENCES

[1] C. Yin, "Simulation of the queue system in ticket business based on Monte Carlo method," in 2010 International Conference on Environmental Science and Information Application Technology (ESIAT), 2010, vol. 3, pp. 141–144.

[2] Sundarapandian, Probability, Statistics and Queuing Theory. PHI Learning Pvt. Ltd., 2009.

[3] D. A. Menascé, V. A. F. Almeida, L. W. Dowdy, and L. Dowdy, Performance by Design: Computer Capacity Planning by Example. Prentice Hall Professional, 2004.

[4] J. F. C. Kingman, "The first Erlang century—and the next," Queueing Syst., vol. 63, no. 1–4, pp. 3–12, Dec. 2009.

[5] J. R. Sturgul, Mine Design: Examples Using Simulation. SME, 2000.

[6] P. G. Harrison and N. M. Patel, Performance Modelling of Communication Networks and Computer Architectures (International Computer S, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992.

[7] D. G. Kendall, "Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain," Ann. Math. Stat., vol. 24, no. 3, pp. 338–354, Sep. 1953.

[8] J. Wang, O. Baron, and A. Scheller-Wolf, "M/M/c Queue with Two Priority Classes," Oper. Res., vol. 63, no. 3, pp. 733–749, Jun. 2015.

[9] J. A. Buzacott and J. G. Shanthikumar, Stochastic models of manufacturing systems. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[10] P. F. Brown and R. M. Hamilton, "Reference model for service oriented architecture 1.0. Tech. rep.," 12-Oct-2006. [Online]. Available: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf. [Accessed: 01-May-2016].

[11] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[12] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful Web Services vs. 'Big'' Web Services: Making the Right Architectural Decision," in Proceedings of the 17th International Conference on World Wide Web, New York, NY, USA, 2008, pp. 805–814.