Simulating Service Request Scheduling Using CPU Scheduling Algorithms

Patrick Killeen

Abstract—Understanding how to best meet service request deadlines on time is important. Smart health is an example of an application that meeting service request deadlines is important. This paper's goal is to analyze the performance, that is maximizing the number service request deadlines met, of CPU scheduling algorithms when used by a remote service provider for scheduling service requests. My simulation implementation is done in Java. The simulation has a client that sends many requests to a service provider over a simulated network connection. The requests are then scheduled and serviced. Upon receiving a response, the client logs the results. The results are then analyzed. The results indicate that in some cases some algorithms perform better than other algorithms in different contexts. The priority queue algorithm, for example, is found to have better performance, in terms of servicing high priority requests, in a system context with mostly relaxed deadlines. The results suggest that in general, prioritizing requests that are about to meet their deadline increases performance. Using the results of this paper, one can better choose a CPU scheduling algorithm for meeting service requests on time depending on the task and system context.

Index Terms—service request,cpu,scheduling,java,service provider,algorithm,iot

1 Introduction

The goal of this paper is to gain insight on the nature of meeting service request deadlines using CPU scheduling algorithms. The results will help make sense of the strength and weaknesses of the algorithms in different contexts, which will give insight to software designers, helping them decide what kind of scheduling algorithm should be used given the system's context and requirements.

This paper focuses on discussing and analyzing the results of the simulations I ran using the implementation of a simulated networked system I created (see Figure 1), which involves servicing service requests before their deadline using CPU scheduling algorithms. I compare CPU scheduling algorithms' performances when used by a service provider receiving a series of service requests over the network. The performance measure I chose for comparing the algorithms is the number of requests serviced before their deadline. That is, an algorithm's performance is better than another algorithm's performance if it services more requests by their deadlines. I will also be discussing details about my implementation, which I did using the Java programming language. I simulate request servicing by running a simulation of a network, client, service provider, and a request scheduler. To compare the algorithms and get insight on their performances I vary the context in which they are run. I make sense of the results and discuss interesting things I have noticed about the results found.

2 Related Work

[1] focuses on scheduling service requests with the goal of maximizing profit of the platform, which is based on revenue and cost. My paper also focuses on scheduling service requests, but the differences are my paper focuses on scheduling using CPU scheduling algorithms and with the goal of maximizing the number of requests serviced on time.

[2] focuses on comparing CPU scheduling algorithms to their proposed MIN-MAX algorithm, by analyzing various performance measures on CPU process scheduling. My paper similarly compares CPU scheduling algorithms, but the differences are my paper only focuses on the performance measure of meeting deadlines on time and instead of measuring performance on processes serviced by a CPU, this paper is measuring performance on service requests processed by a service provider.

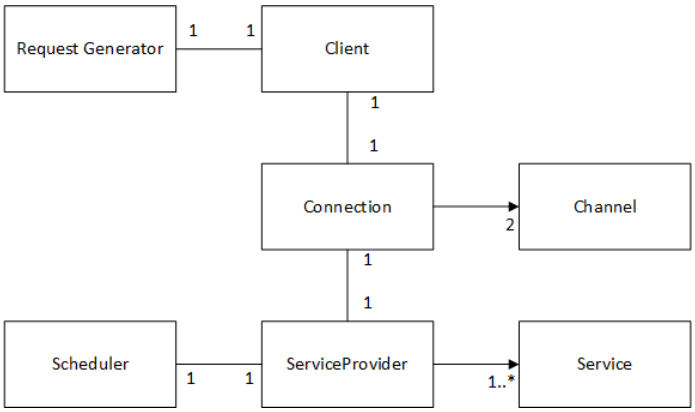

Fig. 1. UML Simple Class Diagram of Simulation Implementation

3 Smart City Applications

There are many applications to meeting service request deadlines on time. For example cost reductions in green

• *The author is with the Master of Computer Science program of at the University of Ottawa, Ottawa, ON, K1N 6N5, Canada.*
E-mail: pkill013@uottawa.ca

cloud computing [3]. Another example, [4] [5] mention that multimedia data can be classified into two categories, real-time and non-real-time. Real-time (or hard real-time) multimedia data is sensitive to delays, while non-real-time (or soft real-time) data isn't. For example, when streaming a video if there are delays then the video will lag, which is much more unacceptable for the user-experience than if a picture download takes a few more seconds to complete. The video is thus real-time data and the picture is non-real-time data in this example. The user-experience scenario may be important to website owners who focus on trying to create a more positive user-experience for their user base.
It may be mandatory that the deadline be met for some types of requests. For example, in a fleet management system, there may be a critical event that must be serviced on time [6], such as an event indicating that there was (or going to be) an accident. Another example could be a patient having a heart attack and a service request must be serviced to notify the hospital. Not meeting the critical service request would have great consequences.
It becomes important to service requests on time as system infrastructures begin to have costly consequences when service requests aren't met on time. Thus, with the rise of Internet of Things (IoT) and applications such as smart health, smart transport, etc., meeting service request deadlines will be critical and thus understanding service request scheduling becomes important.

4 Simulation Work

My simulation was implemented using the Java programming language (see Figure 1 for a UML class diagram of the system). I implemented a client and a service provider as the nodes communicating over the network. The client generates requests and sends them via a simulated network connection to the service provider. The service provider schedules the requests to be serviced and sends a response once the work is done. The client logs the results.

4.1 Sources of Error

Upon completing my implementation, running simulations, and analyzing the results, I noticed there are a couple sources of error that could affect the results' accuracy. One of the sources of error I found is that I made the assumption the Round Robin (RR) algorithm has no latency during context switching. Usually RR has latency when context switching [7]. Thus, the results I found make this assumption since it didn't occur to me to add context switching overhead during implementation. I only realized this could be a source of error upon finishing all the simulations, and observing the results (see Appendix A, Figure 11,12, and 13). I noticed on average the lower time quantum RR simulations had better performance on average, which contradicted my hypothesis that very small time quanta would have too much over head from context switching too frequently, which should be the case according to [7].
Another source of error I found is the relatively small number of simulations I ran when analyzing averages of each algorithm (see Appendix A, Figure 11,12, and 13). I only ran 5 simulations for each algorithm in each context, since running simulations for each algorithm and for each context many times took at least half an hour. It would have taken too much time to run many more simulation instances to get better average results.

4.2 Implementation

I use Java as the language for implementing the simulation. All the parameters of the simulation instances are specified using a key-value pair XML configuration file, which I use for varying the context (system state) parameters. I vary the following contexts for comparing algorithms:

- **Request average deadline** (see Section 5.1): I varied the average request deadlines from more relaxed to stricter deadlines.
- **Network connection latency** (see Section 5.2): I varied the latency/bandwidth from low to high.
- **Request arrival frequency** (see Section 5.3): I varied the rate at which service requests are sent by the client from few to many.
- **Service provider request handling rate** (see Section 5.4): I varied the rate at which requests, those next in line to be serviced, are retrieved from scheduler's queue, from many requests per second to a few requests per second. That is the service provider's rate at which it handles requests, simulating its resources available.
- **Request average burst time**: I varied the average request burst times from short to long, which I don't discuss about in this paper since I didn't find any interesting results.

During the preprocessing phase I compute all the requests along with their parameters by gathering the mean and variances for the following parameters:

- Request deadline
- Service id (reference to request's burst time)
- Priority
- Time between service request creation/sending

4.2.1 Client

The client is the component that sends requests to the service provider at varying frequencies. The client computes the time between sending each service requests. Once all the requests have been created, they are all sent over the simulated network connection at varying time intervals between requests sent. Upon receiving a response, the client decides if the request was met on time based on the response's arrival time and request's sent time. The results are logged into a log file using the comma-separated-value format.

4.2.2 Service Provider

The service provider initializes all services given a burst time, such that the burst times are uniformly distributed between a burst time maximum and minimum. Upon service request arrival the service provider looks up a service given a service id, and determines the request's burst time sending the request to the scheduler (see Section 4.2.3). Impossible requests are rejected immediately, which are requests with a deadline smaller than their burst time. The service provider

services the next service request to be serviced (determined by the scheduler) at a specified frequency (which simulates the service provider's computational resources). The service provider simulates servicing requests by simply sleeping for an amount of time determined by the request's burst time, which is done sequentially for each request received from the scheduler. Once it wakes up, the service provider replies to the client with a response associated to the request the client initially sent over the simulated network connection (see Section 4.2.4).

4.2.3 Scheduler

The scheduler determines the order in which service requests are serviced by using the below CPU scheduling algorithms.

- **First in First Out (FIFO)/First Come First Serve**: requests that arrive first are serviced first [2] [7].
- **Priority Queue (PQ)**: requests with higher priorities are serviced first [2] [7]. See Appendix A, Figure 14 for an example of my simulation's priority queue scheduling.
- **Shortest Job First(SJF)**: requests with smaller burst times are serviced first [2] [7].
- **Nearest Deadline First(NDF)**: requests that are closest to not meeting their deadline are serviced first. That is the requests with their burst time nearest to their deadline.
- **Round Robin(RR)**: requests are serviced for a specific amount of time (time quantum) before being preempted, in which case they are placed back into the scheduler's request queue if they haven't been fully serviced yet [2] [7]. Note that I will be referring to the RR algorithm with an arbitrary time quantum as such, RR(x) where x is the length in milliseconds of the time quantum.

4.2.4 Simulated Network Connection

The network connection is simulated via 2 communication channels between client and service provider. Each channel contains a service request buffer to simulate upstream and downstream bandwidth. Each node on the end of the network connection can send requests through its respectful upstream channel and read incoming requests from the downstream channel. The channels periodically flush the request buffers to simulate bandwidth and latency. I used a thread-safe queue as a synchronization mechanism to ensure thread safety.

5 Results

I will discuss the results I found from the simulation instances and the conclusions I have drawn from them (if any). I compare the results of each context by comparing two simulation instances that have interesting results. I then compare algorithms' performance on average in different contexts. Table 1 is the summary of the results of each algorithm's performance (number of request deadlines met) in each context (see Section 4.2 for the context list) according to all the simulation instances I ran when varying the context. I compiled Table1 by digging through many result diagrams that I haven't included in this report.

TABLE 1
Summary of Algorithm Performances in Various Contexts

Context	State	High	Decent	Poor	Horril
RAD	Relaxed	PQ, RR, NDF, SJF	-	FIFO	-
RAD	Strict	-	PQ, RR, NDF, SJF	FIFO	-
NCL	Low	PQ, SJF, RR, NDF	-	FIFO	-
NCL	High	-	PQ, SJF	RR, NDF	FIFO
RABT	Low	RR	NDF, SJF, PQ	FIFO	-
RABT	High	-	NDF, SJF, RR	PQ	FIFO
SPRHR	High	PQ, SJF, RR, NDF	-	FIFO	-
SPRHR	Poor	-	PQ, SJF, NDF	RR	FIFO
RAR	Low	PQ, SJF, RR, NDF	-	FIFO	-
RAR	High	-	PQ, SJF	RR, NDF	FIFO

RAR: Requests' Average Deadline
NCL: Network Connection Latency
RABT: Requests' Average Burst Time
SPRHR: Service Provider Request Handling Rate
RAR: Request Arrival Rate

5.1 Deadline

5.1.1 Relaxed Deadline - Comparing PQ and NDF

I found interesting results when comparing two simulation instances of the PQ and NDF algorithms in a context with relaxed deadlines. Both algorithms perform similarly well, servicing many requests on time (see Appendix A, Figure 6). Figure 2 illustrates that NDF serves most requests with stricter deadlines and longer burst times, while PQ serves requests with a variety of deadlines and burst times. Appendix A, Figure 7 illustrates s that PQ serves high priority (low values) requests and starves the low priority (high values) requests. This isn't entirely desirable as PQ behavior, since one of the objectives of a PQ is to service high pirority requests without starving low priority requests [8].

5.1.2 Strict Deadline - Comparing PQ and NDF

In a context with stricter deadlines, both NDF and PQ lose performance (see Appendix A, Figure 6), and the requests

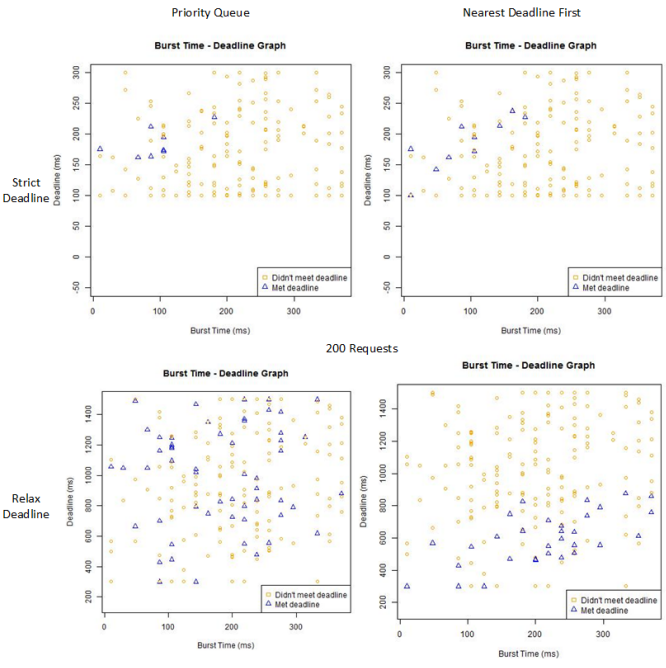

Fig. 2. Deadline Variation, Deadline - Burst Time Plot

served are those with a shorter burst time (see Figure 2). What I found interesting, is that PQ loses its effectiveness of servicing high priority requests over low priority requests (see Appendix A, Figure 7). According to [8] this is undesirable PQ behavior.

5.1.3 Varying Deadline - Comparing All Algorithms on Average

The results I found varying the deadline contexts suggest that RR is the slightly better choice for a stricter deadline context and SJF is the better choice for a more relaxed deadline context (see Appendix A, Figure 12).

5.1.4 Findings

The results found in varying deadline contexts suggest that if service request priorities are important, the PQ algorithm should be used in a more relaxed deadline context, and it shouldn't be used in a stricter deadline context. In a context with more relaxed deadlines, NDF serves most requests with stricter deadlines and longer burst times, which is its usual behavior according to the results found (see Section 5.1.1, 5.3.3 , and 5.4.1). What is worth noting is that in a strict deadline context NDF behaves opposite of its usual behavior. Instead, in a context with stricter deadlines NDF performs better on mostly requests that have shorter burst times and more relaxed deadlines.

5.2 Network Connection Latency

5.2.1 Low Latency - Comparing RR(25) and SJF

Both RR(25) and SJF algorithms perform well in a low latency context (see Appendix A, Figure 8). What I found interesting, according to results illustrated in Figure 3, is that RR(25) is able to better service the requests with stricter deadlines and longer burst times compared to SJF. SJF better services the requests that have more relaxed deadlines and lower burst times.

5.2.2 High Latency - Comparing RR(25) and SJF

According to the results shown in Appendix A, Figure 8, SJF does better than RR(25ms) in a high latency context.

5.2.3 Varying Latency - Comparing All Algorithms on Average

According to simulation results shown in Appendix A, Figure 11, SJF is the better performing algorithm when latency of the network is the varied parameter.

5.2.4 Findings

The results found regarding latency suggest that when latency is high, SJF is the best candidate algorithm. In an arbitrary latency context, when requests with more relaxed deadlines and shorter burst times are prioritized, then the SJF is the best candidate algorithm. In the case that requests have stricter deadlines and longer burst times, RR(25) is the best candidate.

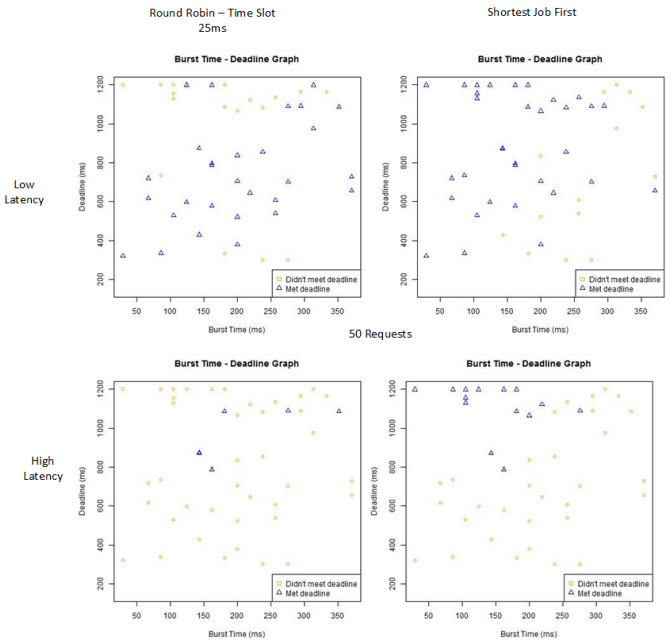

Fig. 3. Latency Variation, Deadline - Burst Time Plot

5.3 Request Arrival Rate

5.3.1 Comparing SJF and NDF

According to the simulation results shown in Appendix A, Figure 5 and Figure 4, both SJF and NDF perform well in a context where the service request arrival rate is low, and in a context with high request arrival rate SJF out-performs NDF.

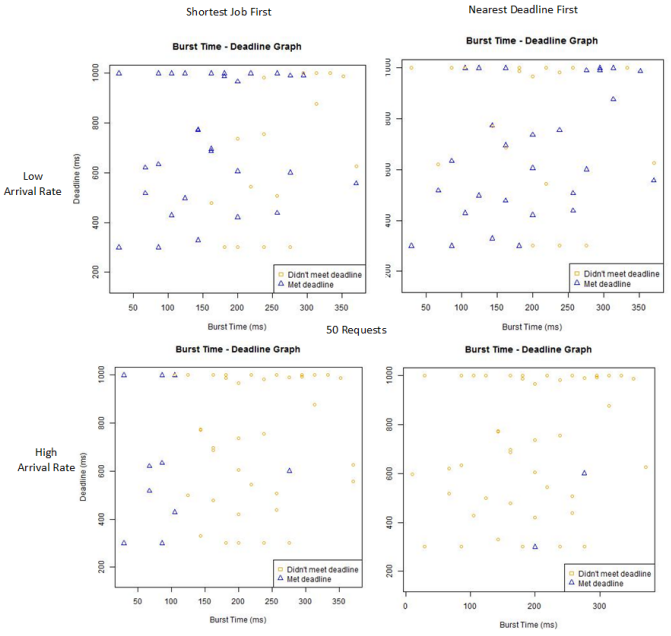

Fig. 4. Arrival Rate Variation, Deadline - Burst Time Plot

5.3.2 Varying Request Arrival Rate - Comparing All Algorithms on Average

According to the simulation results shown in Appendix A, Figure 13, SJF out-performs all the other algorithms on average mostly in a relatively high request arrival rate context. Most of the algorithms perform similarity when there aren't many incoming requests.

5.3.3 Findings

According to the results found regarding request arrival rates, SJF performs best when requests have more relaxed deadlines and lower burst times. This is similar to the results discussed in Section 5.2.4. Furthermore, NDF performs better when requests tend to have longer burst times and stricter deadlines, which is similar to results found in Section 5.1.4 and 5.4.1.

5.4 Service Provider Request Handling Rate

5.4.1 Comparing SJF and NDF

According to simulation results from Appendix A, Figure 9, both SJF and NDF perform well in a high request handling rate context and equally as poorly when the service provider's request handling rate is poor. According to simulation results from Appendix A, Figure 10, SJF performs the best when requests have more relaxed deadlines and lower burst times (which is similar to the results discussed in Section 5.2.4 and 5.3.3), and NDF tends to perform better in contexts with stricter deadlines and longer burst times (which is similar to the results discussed in Section 5.1.4 and 5.3.3).

6 Summary and Concluding Remarks

After running my simulations, although there may be some sources of error that affect the accuracy of the results (see Section 4.1), the results have given me insight into the performance of various CPU scheduling algorithms when used by a remote service provider to schedule service requests. What I have concluded is that SJF tends to perform best when requests have more relaxed deadlines and lower burst times. On the other hand, NDF tends to service most requests that have stricter deadlines and longer burst times. Intuitively these results make sense. On average, I found the SJF algorithm has better performance in most cases.
The most interesting results I have found running these simulations is that in a strict deadline context, the PQ algorithm may not be the best choice when it is important to service requests with higher priorities before lower priority requests. The PQ fails to serve mostly high priority requests instead of the low priority requests in a strict deadline context, which is contradictory to the nature of the PQ algorithm (see Section 5.1.4). The results found give insight into how scheduling can be accomplished in general. With the results of this simulation, I envision future work on the topic could involve a machine algorithm to classify the context/state of the system to choose the most appropriate scheduling algorithm in real-time, which is similar to the research done in [9] and mentionned in [3].

References

[1] Z. Liu, S. Wang, Q. Sun, H. Zou, and F. Yang, "Cost-aware cloud service request scheduling for saas providers," *The Computer Journal*, vol. 57, no. 2, pp. 291 – 293, 2014.
[2] K. Sukhija, N. Aggarwal, and M. Jindal, "An optimized approach to cpu scheduling algorithm: Min-max," *Journal of Emerging Technologies in Web Intelligence*, vol. 6, no. 4, pp. 420 – 422, 2014.
[3] J. Bi, H. Yuan, W. Tan, and B. H. Li, "Trs: Temporal request scheduling with bounded delay assurance in a green cloud data center," *Information Sciences*, vol. 360, pp. 57 – 58, 2016.
[4] S. Kim, "Adaptive online processor management algorithms for multimedia data communication with qos sensitivity," *International Journal of Communication Systems*, vol. 22, pp. 469 – 470, 2009.
[5] X. Hei and D. H. Tsang, "The earliest deadline first scheduling with active buffer management for real-time traffic in the internet," *In: Lorenz P. (eds) Networking ICN 2001. ICN 2001. Lecture Notes in Computer Science*, vol. 2093, pp. 45 – 54, 2001.
[6] ukasz Kruk, J. Lehoczky, K. Ramanan, and S. Shreve, "Heavy traffic analysis for edf queues with reneging," *The Annals of Applied Probability*, vol. 21, no. 2, pp. 484 – 485, 2011.
[7] A. Silberschatz, P. Galvin, and G. Gagne, *Operating System Concepts*. Hoboken New Jersey: Wiley, 2008.
[8] D. avdar, R. Birke, L. Y.Chen, and F. Alagzb, "A simulation framework for priority scheduling on heterogeneous clusters," *Future Generation Computer Systems*, vol. 52, p. 44, 2015.
[9] C.-M. Tien, C.-J. Lee, P.-W. Cheng, and Y.-D. Lin, *SOAP Request Scheduling for Differentiated Quality of Service*. Berlin Heidelberg: M. Dean et al. (Eds.): WISE 2005 Workshops, LNCS 3807, Springer, 2005.

7 Appendix A

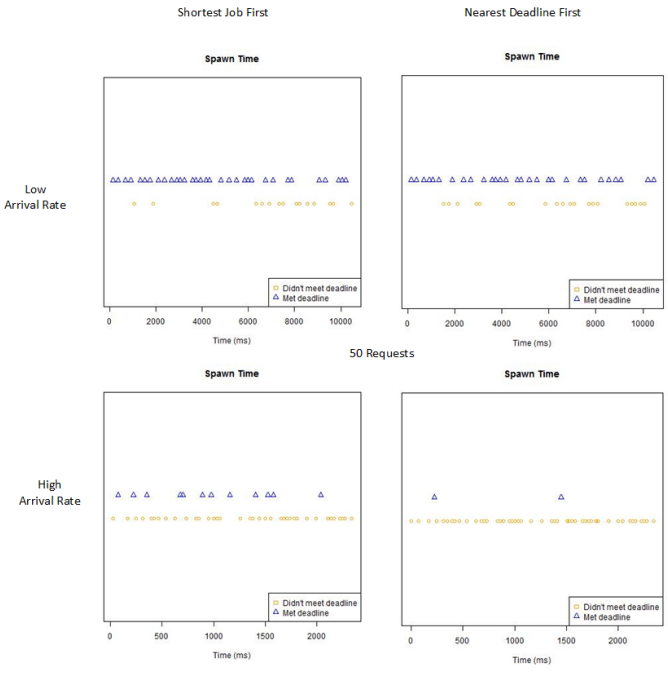

Fig. 5. Arrival Rate Variation, Request Spawn Time Plot

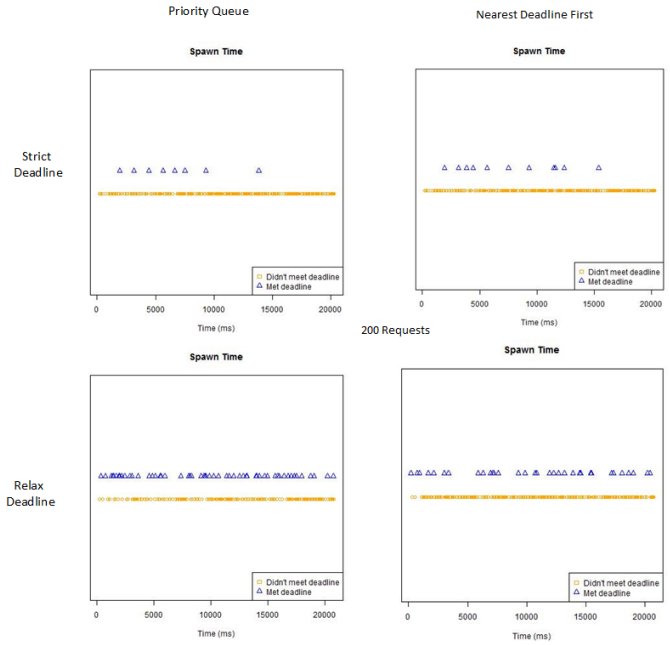

Fig. 6. Deadline Variation, Deadline Request Spawn Time Plot

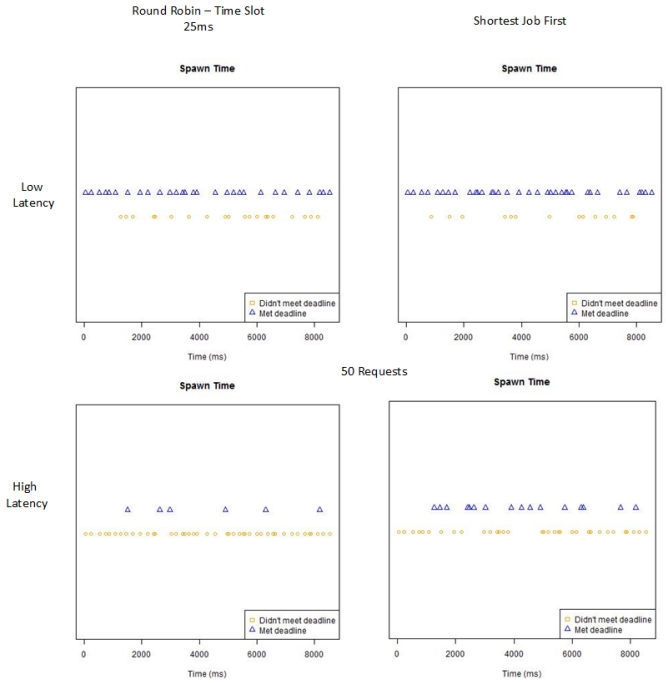

Fig. 8. Latency Variation, Request Spawn Time Plot

Fig. 7. Deadline Variation, Priority Histogram

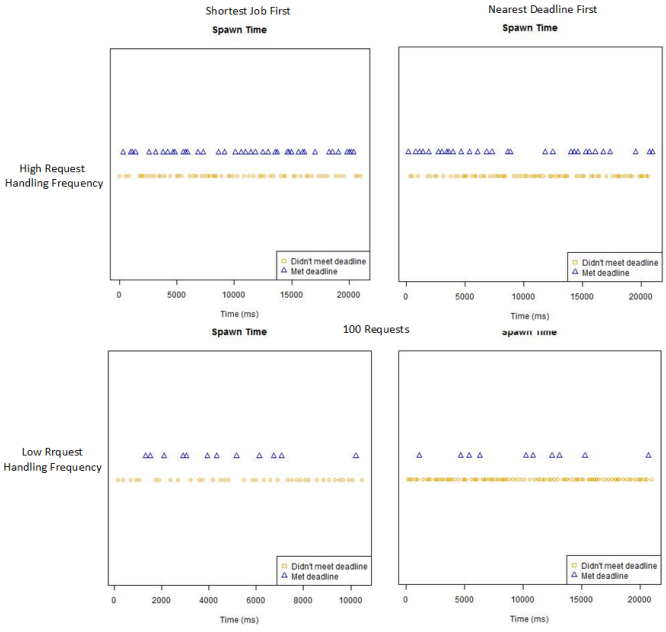

Fig. 9. Request Handling Rate Variation, Request Spawn Time Plot

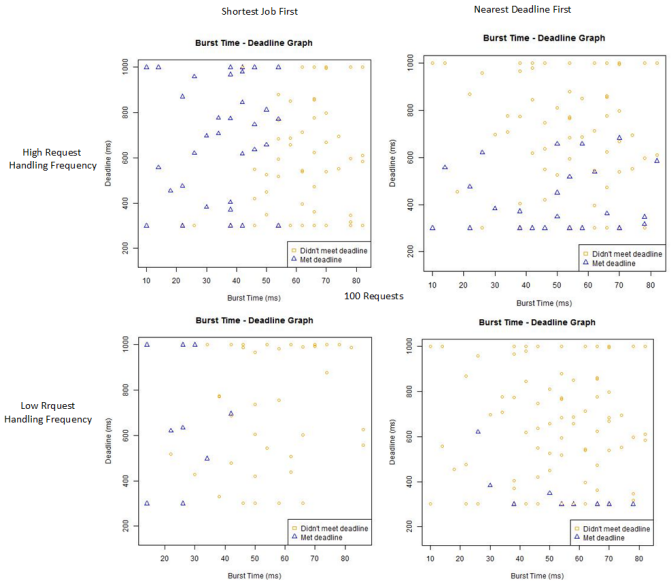

Fig. 10. Request Handling Rate Variation, Deadline - Burst Time Plot

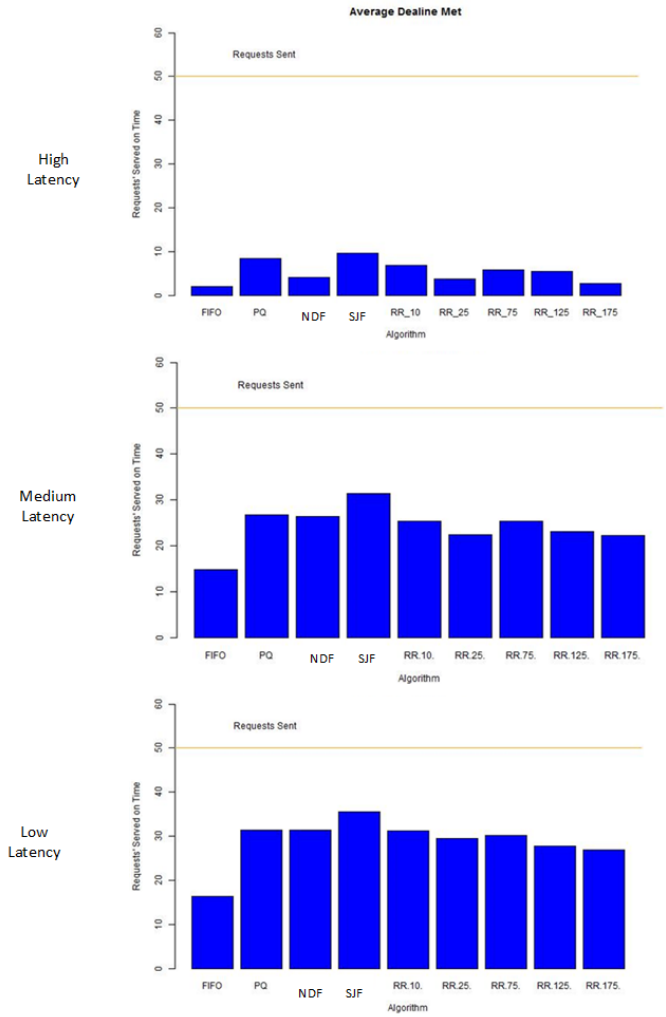

Fig. 11. Latency Variation Average Algorithm Comparison Plot, 5 Simulation Iterations

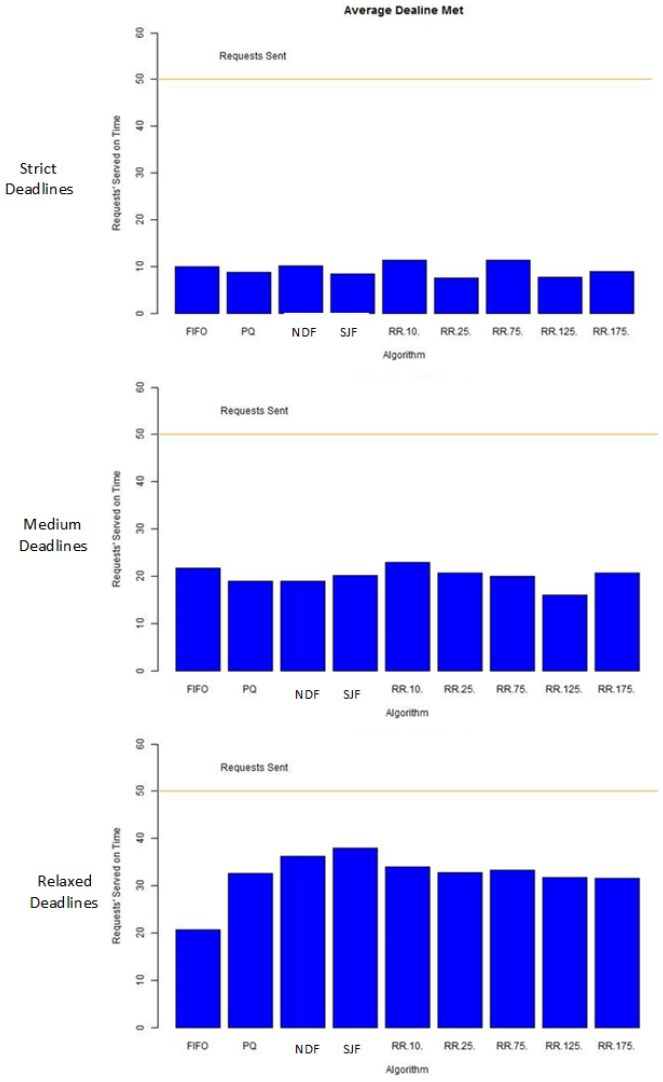

Fig. 12. Deadline Variation Average Algorithm Comparison Plot, 5 Simulation Iterations

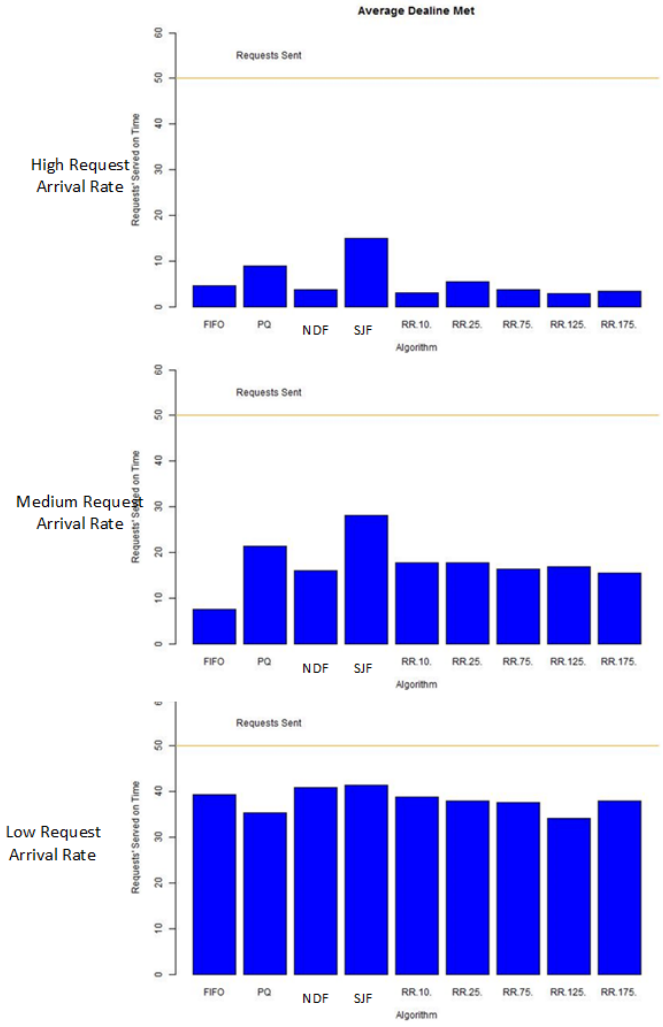

Fig. 13. Arrival Rate Variation Average Algorithm Comparison Plot, 5 Simulation Iterations

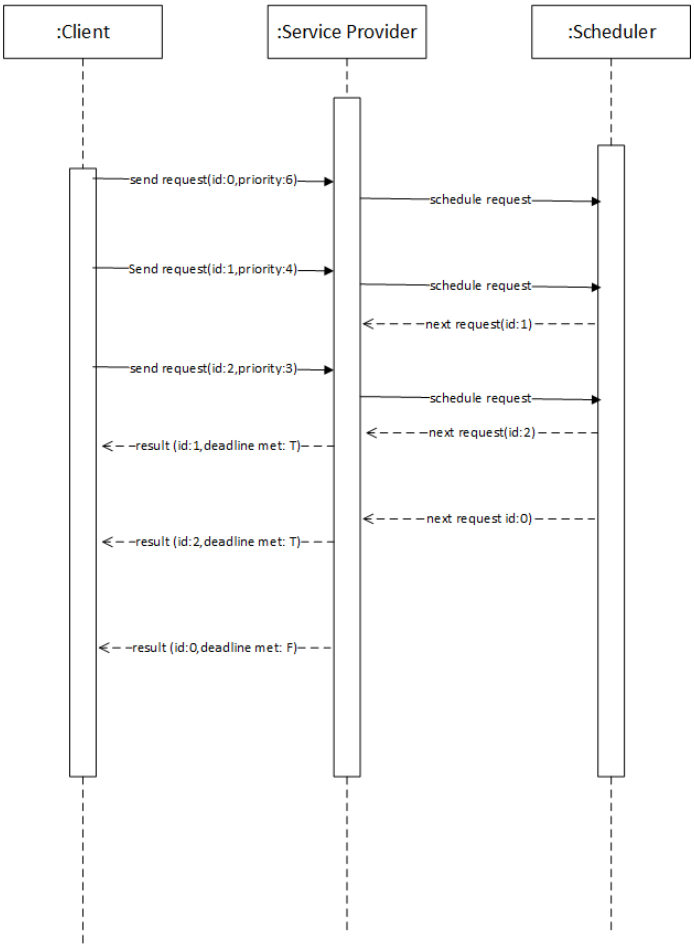

Fig. 14. UML Sequence Diagram of Priority Queue Simulation