UNIVERSITEIT GENT

MASTER OF SCIENCE IN COMPUTER SCIENCE ENGINEERING

Academic year 2015–2016

PROJECT QUEUEING ANALYSIS AND SIMULATION
DISCRETE EVENT SIMULATOR

Titouan Vervack

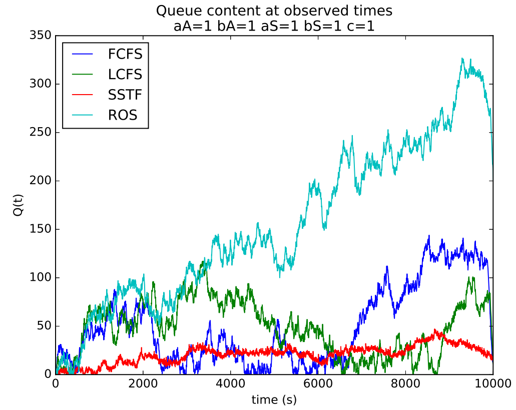
1 Performance measurements

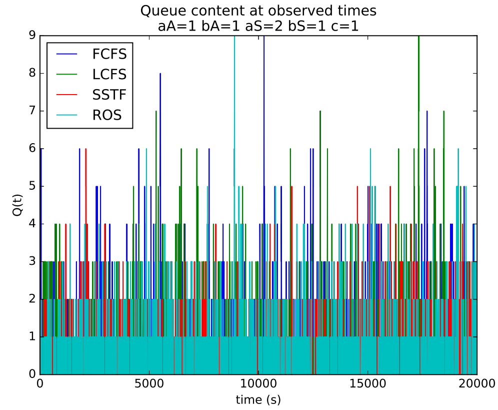
In this section the observed performance measurements using the FCFS, LCFS, SSTF, and ROS scheduling disciplines are discussed. The performance measurements that have been observed are queue content, waiting time and sojourn time. The confidence interval and sample means variance, produced through the batch means method, for the estimated queue content are also shown. All of the code, written for this project, can be found in 2.

It is important to note that for we observe at random times. This has been achieved by scheduling monitor events, which do not change the state of the system, at times sampled from a Poisson distributions, because of the PASTA property we know that these times can be considered random. The same names as in the code of course notes are used. aA is the α of the arrivals, bA the β, aS is the α of the services, bS is the β of the services and c is the number of servers. If the α's and β's are not mentioned, they are presumed to be 1 for the arrivals and 1 for the services. If the number of servers is not mentioned, it is presumed to be 1. Lastly, the λ for the Poisson process monitoring the system is $\frac{aS+bS+aA+bA}{3}$.

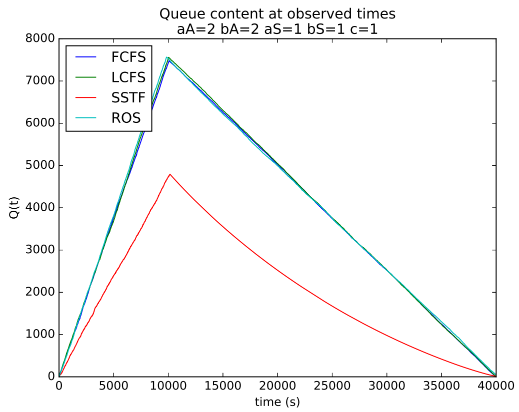
1.1 Queue content

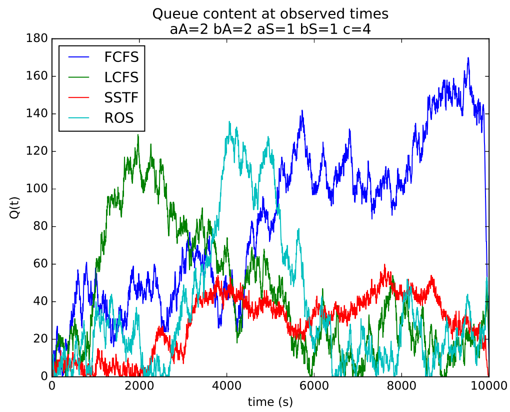
For the queue content with our default values, we see that in every case, but the SSTF, the queue content is quite high most of the time. This is because we are drawing service rates and arrival rates from the same distribution. Sometimes the arrivals will be a little faster than the the services, hence the down drafts, on the other hand they will also be be longer at some times, causing the peaks. The queue content of SSTF is lower overall, because it first gets rid of all the services that can be done quickly. However at some time it will run out of service that can be done quickly and it the content will rise momentarily. If we now increase our β of the services and see that our queue content bounces between 0 and 10, this is because services now take less long than arrivals.





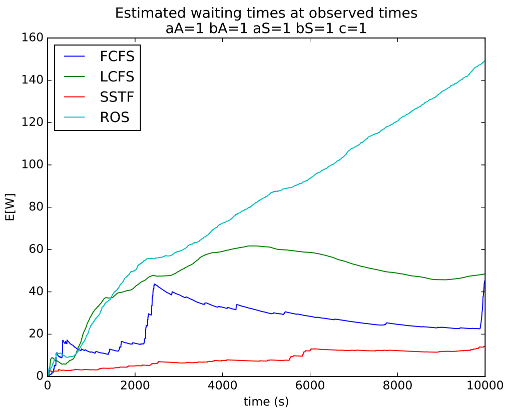
If we increase $\alpha and \beta$ of our arrival process, the arrivals will be faster than the services and thus we see the queue content rising until the maximum amount of arrivals has happened after which we see a steady decrease of the customers. SSTF performs better in this case for the same reasons as mentioned before. Adding more servers to allows everything but FCFS to cope with the number of arrivals. Checking out LCFS, we notice that the peak of the distribution is around 2000, but since FCFS first has to has process all of the others, it will only notice this maximum later on, at about 4000s.

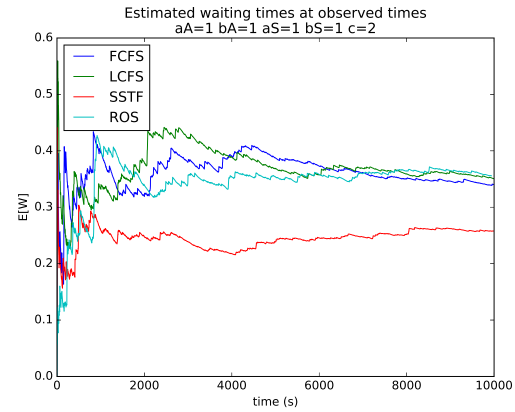


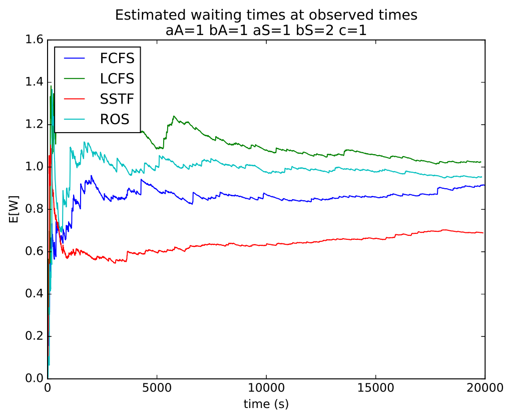


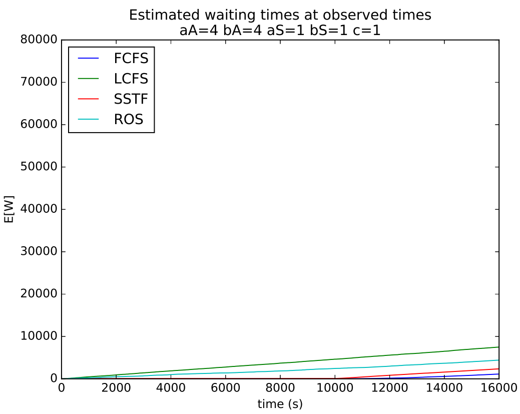
1.2 Waiting time

With our default values, waiting times converge after some time, however for ROS they go to infinity. When we increase the server count the waiting time converges to zero because we now have two servers while the arrival and service rate are nearly the same. Increasing the β of the services causes the services to happen faster than the arrivals, which causes the waiting time to converge to 0. Again we see, when we increase α and β of the arrivals, our waiting times goes to infinity because arrivals are now faster than services and we can not cope with it.



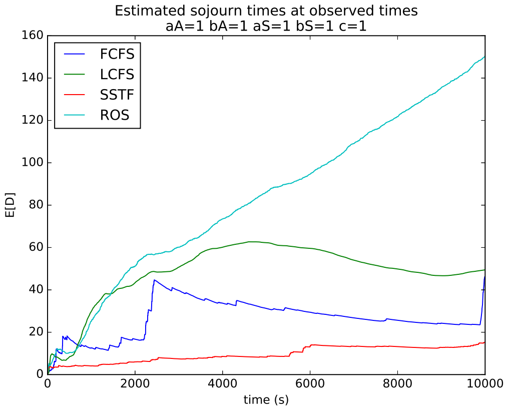


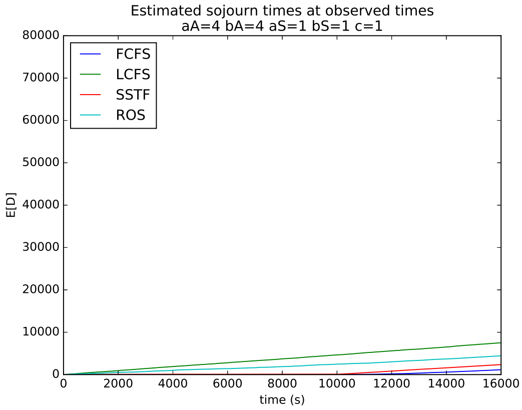




1.3 Sojourn time

The sojourn time remains almost identical to the waiting time as it is the waiting time + the service time. The sojourn time can thus only be significantly different if the service time takes up the majority of the sojourn time. Having a longer service time however, causes a longer waiting time, thus it can never make a significant difference. We just show two graphs from waiting time again to show that there is indeed almost no difference.

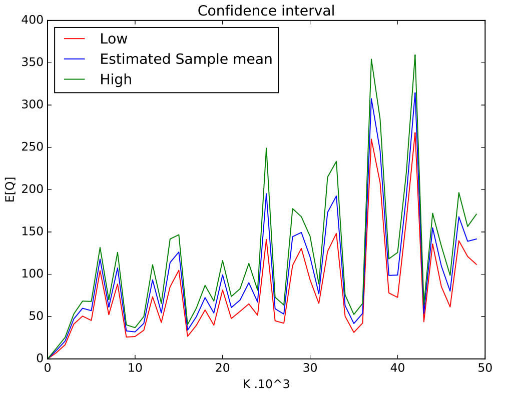


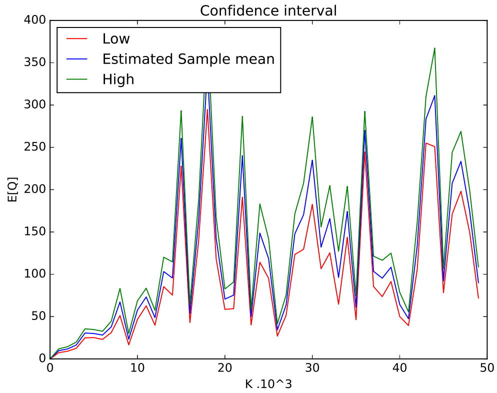


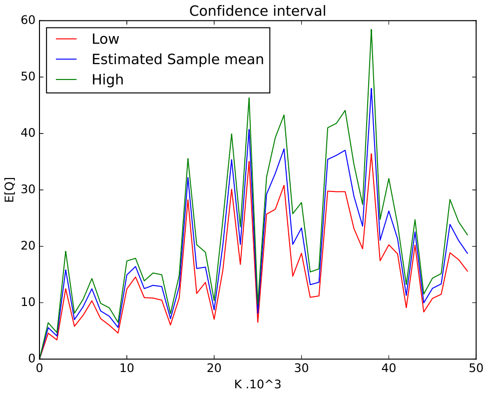
1.4 Batch means method

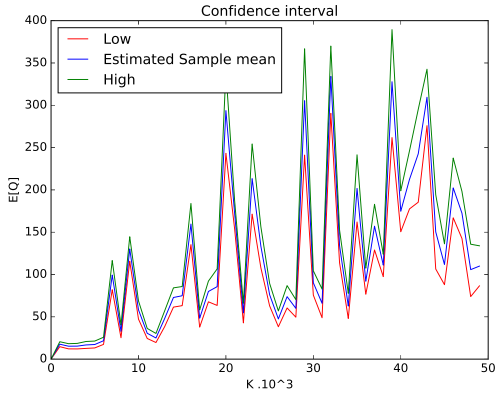
For the batch means method a k_p of 2.58 was chosen, this value can be found in the standard normal distribution tables (Z-tables). It is the factor needed to calculate the 99% confidence interval.

The plots for the different scheduling disciplines have been not been shown in one figure as before, as it would make the figure very unclear. The below figures depict the 99% confidence interval for the estimated waiting time obtained through the batch means method.

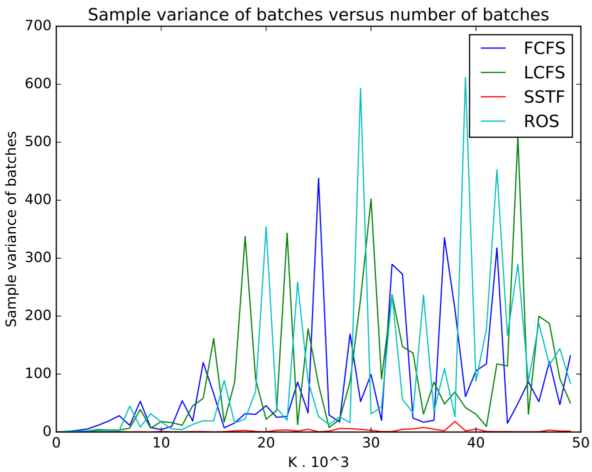








Below you can find the sample variances obtained through the batch means method.



2 Code

```python
import matplotlib.pyplot as plt

from FCFS import FCFS
from LCFS import LCFS
from ROS import ROS
from SSTF import SSTF

base = "../figures/"
batches = 50


def plot_batch_means(batch_means, strat):
    plt.figure()
    plt.plot(range(0, batches), [b[0] for b in batch_means], 'r', label="Low")
    plt.plot(range(0, batches), [b[1] for b in batch_means], 'b', label="Estimated Sample mean")
    plt.plot(range(0, batches), [b[2] for b in batch_means], 'g', label="High")
    plt.xlabel("K .10^3")
```

```
    plt.ylabel("E[Q]")
    plt.legend(loc=2)
    plt.title("Confidence interval")
    plt.savefig(base + strat.name() + "_batch_mean_ci.eps")


def plot_batch_vars(batch_means, strats):
    plt.figure()
    for k in range(0, len(strats)):
        plt.plot(range(0, batches), [b[3] for b in batch_means[k]],
                 label=strats[k].name())

    plt.xlabel("K . 10^3")
    plt.ylabel("Sample variance of batches")
    plt.legend(loc=1)
    plt.title("Sample variance of batches versus number of batches")
    plt.savefig(base + "batch_mean_var.eps")


def plot_queue_content(strats, aarr=1, barr=1, aser=1, bser=1, c=1):
    plt.figure()
    for st in strats:
        plt.plot(st.mon_events[:-10], st.queue_contents[:-10], label=st.name())
    plt.xlabel("time (s)")
    plt.ylabel("Q(t)")
    plt.legend(loc=2)
    plt.gca().set_xlim([0, 16000])
    plt.title("Queue content at observed times\naA={0} bA={1} aS={2} bS={3} c={4}".format(aarr, barr, aser, bser, c))
    plt.savefig(base + "queue_content" + str(aarr) + str(barr) + str(aser) + str(bser) + str(c) + ".eps")


def plot_waiting_times(strats, aarr=1, barr=1, aser=1, bser=1, c=1):
    plt.figure()
    for st in strats:
        plt.plot(st.mon_events[:-10], st.avg_waiting_time[:-10], label=st.name())
    plt.xlabel("time (s)")
    plt.ylabel("E[W]")
    plt.legend(loc=2)
    plt.gca().set_xlim([0, 16000])
    plt.title(
        "Estimated waiting times at observed times\naA={0} bA={1} aS={2} bS={3} c={4}".format(aarr, barr, aser, bser,
                                                                                              c))
    plt.savefig(base + "waiting_times" + str(aarr) + str(barr) + str(aser) + str(bser) + str(c) + ".eps")


def plot_sojourn_times(strats, aarr=1, barr=1, aser=1, bser=1, c=1):
    plt.figure()
    for st in strats:
        plt.plot(st.mon_events[:-10], st.avg_sojourn_time[:-10], label=st.name())
    plt.xlabel("time (s)")
    plt.ylabel("E[D]")
    plt.legend(loc=2)
    plt.gca().set_xlim([0, 16000])
    plt.title(
        "Estimated sojourn times at observed times\naA={0} bA={1} aS={2} bS={3} c={4}".format(aarr, barr, aser, bser,
                                                                                              c))
    plt.savefig(base + "sojourn_times" + str(aarr) + str(barr) + str(aser) + str(bser) + str(c) + ".eps")


if __name__ == '__main__':
    strategies = [FCFS, LCFS, SSTF, ROS]

    done = []
    for s in strategies:
        done.append(s(10000, 4, 4, 1, 1, 1))
        done[-1].run()

```

```
    plot_queue_content(done, 4, 4, 1, 1, 1)
    plot_waiting_times(done, 4, 4, 1, 1, 1)
    plot_sojourn_times(done, 4, 4, 1, 1, 1)

    means2 = []
    for s in strategies:
        means = [(0, 0, 0, 0)]
        for i in range(1, batches):
            strategy = s(i * 1000)
            strategy.run()
            means.append(strategy.batch_means(2.58))

        means2.append(means)
        plot_batch_means(means, s)
    plot_batch_vars(means2, done)
```

Listing 1: main.py

```
from abc import abstractmethod
from math import sqrt, ceil, floor

from numpy.random import poisson, gamma

from agenda import Agenda
from customer import Customer


class Strategy:
    def __init__(self, max_arrivals, a_ser=1, b_ser=1, a_arr=1, b_arr=1, c=1):
        self.max_arrivals = max_arrivals
        self.aS = a_ser
        self.bS = b_ser
        self.aA = a_arr
        self.bA = b_arr
        self.c = c
        self.lam = (self.aS + self.bS + self.aA + self.bA) / 3
        # M
        self.batch_size = floor(self.max_arrivals / 100)

        # Keeps track of customers
        self.arrivals = 0
        self.departures = 0
        self.done = []

        # Performance measures
        self.mon_events = []
        self.queue_contents = []
        self.avg_waiting_time = []
        self.avg_sojourn_time = []

        # Utils for calculating performance measures
        self.prev_waiting_time = 0
        self.prev_sojourn_time = 0
        self.prev_done_index = 0

    def run(self):
        agenda = Agenda(self.c)
        agenda.schedule(0, self.arrival, [])
        agenda.schedule(0, self.monitor, [])
        while agenda.has_events():
            agenda.next_event()

    def batch_means(self, kp=1):
        # K
        tot_event = len(self.mon_events)
        # L
        batches = ceil(tot_event / self.batch_size)
```

```
        sample_means = []
        for i in range(0, batches):
            batch = self.queue_contents[i * self.batch_size:(i + 1) * self.batch_size]
            # j_l
            sample_means.append(sum(batch) / len(batch))

        # J_K
        overall_sample_mean = sum(sample_means) / batches
        temp_sum = sum([(l - overall_sample_mean) ** 2 for l in sample_means])
        variance = temp_sum / (batches ** 2)
        sd = sqrt(temp_sum) / batches
        # J
        estimated_mean = sum(self.queue_contents) / len(self.mon_events)
        low = overall_sample_mean - (kp * sd)
        high = overall_sample_mean + (kp * sd)
        return low, estimated_mean, high, variance

    def arrival(self, agenda):
        # Servers are available, service the customer
        service_time = gamma(self.aS, self.bS)
        if agenda.has_servers():
            # Initialise the customer
            customer = Customer(agenda.current_time, service_time)
            customer.set_service_start(agenda.current_time)

            # Schedule customer for departure
            agenda.get_server()
            agenda.schedule(agenda.current_time + service_time, self.departure, [customer])

        # No servers available, add customer to the queue
        else:
            agenda.add_customer(Customer(agenda.current_time, service_time))

        self.arrivals += 1
        if self.arrivals < self.max_arrivals:
            # New arrival
            new_arr_time = gamma(self.aA, self.bA)
            agenda.schedule(agenda.current_time + new_arr_time, self.arrival, [])

    def departure(self, agenda, customer):
        self.departures += 1
        customer.set_service_done(agenda.current_time)
        agenda.free_server()
        self.done.append(customer)

        # If customers in queue
        if agenda.queue:
            # Get the customer according to the current strategy and further initialise it
            new_customer = self.get_customer(agenda)
            new_customer.set_service_start(agenda.current_time)

            # Schedule customer for departure
            agenda.get_server()
            dep_time = agenda.current_time
            agenda.schedule(dep_time + new_customer.service_time, self.departure, [new_customer])

    def monitor(self, agenda):
        self.mon_events.append(agenda.current_time)
        # Sum the performance measures since the previous monitor and save them
        self.queue_contents.append(len(agenda.queue))
        self.prev_waiting_time += sum([getattr(x, "waiting_time")() for x in self.done[self.prev_done_index:]])
        self.prev_sojourn_time += sum([getattr(x, "sojourn_time")() for x in self.done[self.prev_done_index:]])
        denominator = 1 if len(self.done) == 0 else len(self.done)
        self.avg_waiting_time.append(self.prev_waiting_time / denominator)
        self.avg_sojourn_time.append(self.prev_sojourn_time / denominator)
        self.prev_done_index = len(self.done)

```

```
        # Schedule the next monitoring event if not all customers have been served yet
        if self.departures < self.max_arrivals:
            poiss_time = poisson(self.lam)
            agenda.schedule(agenda.current_time + poiss_time, self.monitor, [])

    @abstractmethod
    def get_customer(self):
        """Returns a customer according to the current strategy"""
        return
```

Listing 2: strategy.py

```
class Agenda:
    def __init__(self, c):
        self.c = c
        self.events = {}
        self.current_time = 0
        self.queue = []

    def schedule(self, time, event, arguments):
        if time in self.events:
            self.events[time].append([event, arguments])
        else:
            self.events[time] = [[event, arguments]]

    def next_event(self):
        self.current_time = min(self.events)
        for event, args in self.events.pop(self.current_time):
            args = [self] + args
            event(*args)

    def has_events(self):
        return len(self.events) > 0

    def has_servers(self):
        return self.c > 0

    def get_server(self):
        self.c -= 1

    def free_server(self):
        self.c += 1

    def add_customer(self, customer):
        self.queue.append(customer)
```

Listing 3: agenda.py

```
class Customer:
    def __init__(self, arrival_time, service_time):
        self.service_time = service_time
        self.arrival_time = arrival_time
        self.service_start = 0
        self.service_done = 0

    def set_service_start(self, service_start):
        self.service_start = service_start

    def set_service_done(self, service_done):
        self.service_done = service_done

    def waiting_time(self):
        return self.service_start - self.arrival_time

    def sojourn_time(self):
        return self.service_done - self.arrival_time
```

Listing 4: customer.py

```
from strategy import Strategy


class FCFS(Strategy):
    @staticmethod
    def get_customer(agenda):
        return agenda.queue.pop()

    @staticmethod
    def name():
        return "FCFS"
```

Listing 5: FCFS.py

```
from strategy import Strategy


class LCFS(Strategy):
    @staticmethod
    def get_customer(agenda):
        return agenda.queue.pop(0)

    @staticmethod
    def name():
        return "LCFS"
```

Listing 6: LCFS.py

```
from operator import attrgetter

from strategy import Strategy


class SSTF(Strategy):
    @staticmethod
    def get_customer(agenda):
        result = min(agenda.queue, key=attrgetter('service_time'))
        agenda.queue.remove(result)

        return result

    @staticmethod
    def name():
        return "SSTF"
```

Listing 7: SSTF.py

```
from random import randrange

from strategy import Strategy


class ROS(Strategy):
    @staticmethod
    def get_customer(agenda):
        return agenda.queue.pop(randrange(len(agenda.queue)))

    @staticmethod
    def name():
        return "ROS"
```

Listing 8: ROS.py