Performance Evaluation of a Deficit Scheduler

Università di Pisa

Pezzuti Francesca Sanguinetti Marta Vivani Alessio

Academic year 2021/2022

Contents

1 Introduction

Our project consisted in the implementation and performance analysis of a deficit scheduler. There is a source sending jobs to a server. Job inter-arrival times are IID RVs and their service times are IID RV. The server serves jobs in the queue automatically and service occurs within one turn. The expected time that the server should spend on the source is Q. However:

- If the queue is empty, the turn is terminated.
- If the next job cannot be served entirely before the end of the turn, the turn is terminated and the remaining time (called deficit) is saved and summed to the duration of the next turn. If the deficit is equal to D then the next turn will be Q+D units of time.

After a turn, the server then *goes on vacation* for some time. Vacations are IID RVs. If a server comes back from vacation and finds the queue empty, it immediately starts a new vacation.

2 Simulation

We used OMNeT++ to simulate our system, exploiting NED to implement the structure of the system and C++ to define the behavior of each component.

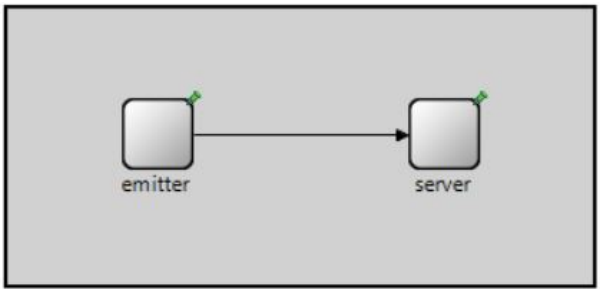

Figure 1: OMNeT++ model

Our network is composed by two simple modules, an emitter that sends jobs and a server that receives them, puts them in a queue and serves them. The emitter has two factors, the inter-arrival time and the service-time, while the server has only one factor, the vacation period. For each one of them we created a different Random Number Generator from the .ini file having different seed that also changes depending on the repetition id.

```
1  package deficitscheduler.simulations;
2
3  import deficitscheduler.Emitter;
4  import deficitscheduler.Server;
5  import deficitscheduler.Ricevitore;
6
7  network simpleEmitter
8  {
9      submodules:
10
11         emitter: Emitter {
12             constant = false;
13         }
14         server: Server {
15             constant = false;
16             turn_time = emitter.requested_service_time;
17         }
18     connections:
19         emitter.out --> server.in;
20 }
```

2.1 Emitter

A simple module that sends jobs to the server.

2.1.1 Emitter.ned

Its parameters are:

- `constant:`

  a boolean that specifies if we are using constant values or exponential distributions;

- `interarrival_time:`

  it is the value of inter-arrival times in case we are using constant values, otherwise it is the mean of the exponential distribution;

- `requested_service_time:`

  it represents either the constant service time of each job or the mean of the exponential distribution of service times.

It has only one output gate.

```
package deficitscheduler;

simple Emitter {

    parameters:
        bool constant;
      double interarrival_time @unit(s) = default(10ms);
      double requested_service_time @unit(s) = default(2ms);
    gates:
        output out;

}
```

2.1.2 Emitter.h

```
#ifndef __DEFICITSCHEDULER_EMITTER_H_
#define __DEFICITSCHEDULER_EMITTER_H_

#include <omnetpp.h>

using namespace omnetpp;

class Emitter : public cSimpleModule
{
    private:
        bool constant;
        simtime_t arrival_time;
        simtime_t service_time;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *job);
};

#endif
```

2.1.3 Emitter.cc

When the emitter is initialized, the function *initialize()* obtains the values of arrival time and service time, then it creates the first job that it wants to send and sets its service time. In the end it schedules a message after a period equal to *arrival_time*.
Then, when a self message is received, it creates a new message, sets the values and schedules a new message after a period equal to *arrival_time*. This is done by the function *handleMessage(cMessage *msg)*.

```
#include "Emitter.h"
#include "Job_m.h"

Define_Module(Emitter);

void Emitter::initialize()
{
    constant = par("constant");
    if (constant) {
        arrival_time = par("interarrival_time");
        service_time = par("requested_service_time");
    }
    else {
        simtime_t mean_arrival_time = par("interarrival_time");
        simtime_t mean_service_time = par("requested_service_time")
    ;
        arrival_time = exponential(mean_arrival_time, 0);
        service_time = exponential(mean_service_time, 1);
    }
    Job *job_to_send = new Job("job_to_send");
    job_to_send->setService_time(service_time.str().c_str());
    scheduleAt(simTime() + arrival_time, job_to_send);
}

void Emitter::handleMessage(cMessage *msg){
    if(msg->isSelfMessage()) {
        send(msg, "out");
        //crea nuovo messaggio
        Job *job_to_send = new Job("job_to_send");

        if(!constant){
            simtime_t mean_arrival_time = par("interarrival_time");
            simtime_t mean_service_time = par("
    requested_service_time");
            arrival_time = exponential(mean_arrival_time, 0);
            service_time = exponential(mean_service_time, 1);
        }
        job_to_send->setService_time(service_time.str().c_str());

        scheduleAt(simTime() + arrival_time, job_to_send);
    }

}
```

2.2 Server

A simple module that receives jobs through its input gate, queues them and serves them as described in paragraph 1.

2.2.1 Server.ned

For what concerns its structure, the server has the following parameters:

- `constant:`

  as before, it specifies if we are using constant values or exponential distributions;

- `vacation_time:`

  in case we are using constant values it is the duration of the vacation, otherwise it corresponds to the mean of the exponential distribution;

- `turn_time:`

  it is a constant value representing the duration of the turn.

We have two more parameters that allow us to define a recorder that will keep track of some statistics of the signal of the response time.
In the end, this module has only one input gate.

```
1  package deficitscheduler;
2
3  simple Server {
4
5      parameters:
6          bool constant;
7          double vacation_time @unit(s) = default(0ms);
8          double turn_time @unit(s) = default(1ms);
9          @signal[delay];
10         @statistic[response_time](source="delay"; record=vector,
       mean;);
11   gates:
12       input in;
13 }
```

2.2.2 Server.h

```
#ifndef __DEFICITSCHEDULER_SERVER_H_
#define __DEFICITSCHEDULER_SERVER_H_

#include <omnetpp.h>

using namespace omnetpp;

struct info_holder {
    simtime_t arrival_instant;
    simtime_t service_time;
};

class Server : public cSimpleModule {
    private:
        bool constant;
        simsignal_t delay_signal;
        std::vector<info_holder> queue;
        simtime_t deficit;
        simtime_t Q;
        simtime_t vacation_period;
        cMessage *end_of_vacation;
        cMessage *end_of_service;
        bool working;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
        virtual void startService();
        virtual void startVacation();
        virtual void getResponseTime();
};

#endif
```

2.2.3 Server.cc

When the *initialize()* is called, the server instantiates his parameters and then he immediately starts a vacation, since when the server begins his service the queue will be empty. After the vacation is ended he starts his standard workflow. When it receives a Job from the emitter, it always adds it to the queue since the Server will be either in vacation or already serving a Job. The queue is composed by a list of structures keeping information about the *arrival time* and the *required service time*. In order to compute the *response time*, every time a Job is served we check the departure time of the Job.

```
#include "Server.h"
#include "Job_m.h"

Define_Module(Server);

void Server::initialize() {

    deficit = 0.0;
    constant = par("constant");
    delay_signal = registerSignal("delay");
    Q = par("turn_time");
    vacation_period = par("vacation_time");
    end_of_vacation = new cMessage("vacation");
    end_of_service = new cMessage("service");
    scheduleAt(simTime(), end_of_vacation);
}

void Server::handleMessage(cMessage *msg) {

    if (msg->isSelfMessage()) {

        //if the vacation is ended i start a service turn
        if (strcmp(msg->getName(), "vacation") == 0) {
            if (queue.size() != 0) {
                //Check the size of the next job to serve
                //If it's small enough we serve it, otherwise we
    start a new vacation
                //and get the deficit for the next turn.
                simtime_t job_duration = queue[0].service_time;
                if (job_duration > Q + deficit) {
                    deficit = deficit + Q;
                    startVacation();
                } else {
                    deficit = deficit + Q - job_duration;
                    startService();
                }
            } else {
                //If true, it take the const value, otherwise gets
    it form exponential
                deficit = 0.0;
                EV << "Empty Queue" <<  endl;
                startVacation();
            }
        } else {
```

```
43              //We calculate the response time and remove the job
    from the queue
44              getResponseTime();
45              //Here the service is done, so we look for other jobs
    or we go in vacation
46              if (queue.size() != 0) {
47                  simtime_t job_duration = queue[0].service_time;
48                  if (job_duration > deficit) {
49                      startVacation();
50                  } else {
51                      deficit = deficit - job_duration;
52                      startService();
53                  }
54              } else {
55                  deficit = 0.0;
56                  EV << "Empty Queue" << endl;
57                  startVacation();
58              }
59          }
60      } else {
61          //At this point the server is on vacation or is serving
    another job, so
62          //we just add the new job to the queue.
63          Job *job = check_and_cast<Job*>(msg);
64          struct info_holder job_info_holder;
65          job_info_holder.arrival_instant = simTime();
66          job_info_holder.service_time = SimTime::parse(job->
    getService_time());
67          queue.push_back(job_info_holder);
68          delete(job);
69          EV << "New entry in queue, queue size: " << queue.size() <<
     endl;
70      }
71  }
72
73  void Server::startService() {
74      simtime_t job_duration = queue[0].service_time;
75      EV << "Begin Service of: " << job_duration << endl;
76      EV << "New Deficit: " << deficit << endl;
77      scheduleAt(simTime() + job_duration, end_of_service);
78  }
79
80  void Server::startVacation() {
81      if (!constant) {
82          simtime_t mean_period_time = par("vacation_time");
83          vacation_period = exponential(mean_period_time, 0);
84      }
85      EV << "Begin Vacation of: " << vacation_period << endl;
86      EV << "New Deficit: " << deficit << endl;
87      scheduleAt(simTime() + vacation_period, end_of_vacation);
88  }
89
90  void Server::getResponseTime() {
91      simtime_t now = simTime();
92      simtime_t response_time = now - queue[0].arrival_instant;
93      emit(delay_signal, response_time);
94      //struct info_holder* job = &queue.begin();
```

```
95      queue.erase(queue.begin());
96      //delete(job);
97  }
```

2.3 Job

We extended the cMessage by adding a string value which contains the requested service time for that specific Job. We chose the string so that we can convert the requested service time, that is a simtime_t, to a string and back again to it's original type.

2.3.1 Job.msg

```
1  //package deficitscheduler;
2
3  message Job {
4      string service_time;
5  }
```

3 Queueing theory model

We studied a simplified version of the system using queueing theory in order to check if the OMNeT++ model we designed was working appropriately.

To perform queueing theory modeling we considered the scenario with **exponential arrival rates**, **exponential service rates** and **no server's vacations**.

3.1 Simplifications

To model the system with queueing theory we have neglected the vacations of the server. With this simplification, turns does not make sense anymore because even if we have to calculate a deficit because we have a turn that can't serve a specific job, since we have no vacation, the time required to get a turn time big enough to serve that job is null, so it's exactly like an M/M/1 system.
With this simplification the system can be modeled as an **M/M/1 system** with:

- $E[t_s] = \frac{1}{\mu}$ = mean service time

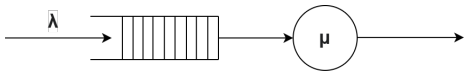

Figure 2: Simple model of the system for queueing theory

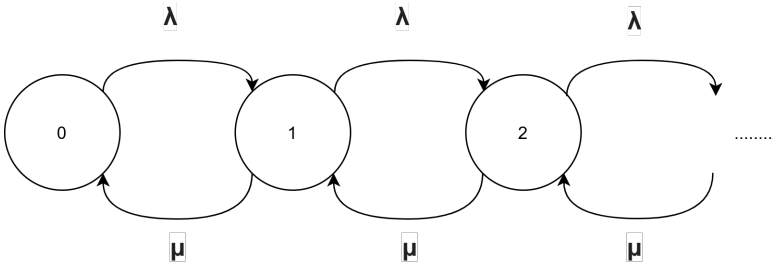

Figure 3: CTMC considering no server's vacations

In state denoted as i there are exactly i jobs in the system.

3.2 Computing the SS probabilities

3.2.1 Local equilibrium equations

$$\begin{cases} p_0 \cdot \lambda = p_1 \cdot \mu & n = 0 \\ p_j \cdot \lambda = p_{j+1} \cdot \mu & n \geq 1 \end{cases}$$

From the *local equilibrium equations* can be obtained p_n as a function of p_0 as it follows:

$$p_n = \left(\frac{\lambda}{\mu}\right)^n \cdot p_0$$

3.2.2 Normalization condition

We can compute the *normalization condition* as:

$$p_0 \cdot \left[\sum_{j=0}^{+\infty} (\frac{\lambda}{\mu})^n\right] = 1$$

3.2.3 Solving the SS system

The system composed by the local equilibrium equations and the normalization condition can be solved and the unique solution for the steady-state probabilities is:

$$\begin{cases} p_0 = 1 - \frac{\lambda}{\mu} \\ p_n = \frac{\lambda^n}{\mu^n} \cdot p_0 \end{cases}$$

The **stability condition** for the system is $\lambda < \mu$
The **Utilization** of the system is $\rho = \frac{\lambda}{\mu}$

3.3 Checking the PASTA condition

The system is PASTA because $\lambda_n = \lambda \ \forall n$.

3.4 Little's Law and Mean Performance Indexes

We can apply Little's Law in order to compute mean performance indexes and we can apply M/M/1's formulas.

- $E[R] = \frac{E[N]}{\bar{\lambda}} = \frac{1}{\mu - \lambda}$
- $E[N] = \frac{\rho}{1-\rho} = \frac{\frac{\lambda}{\mu}}{1-\frac{\lambda}{\mu}}$

4 Validation

In order to validate our OMNeT++ model, we used a simplified version of the system where the server doesn't go on vacation. In this way we compared the results for the mean response time obtained with simulation against those obtained using the model designed with queuing theory. In particular we noticed that the mean response time $E[R]$ of the simulations, ran with **vacations = 0ms** and **exponential arrival rates** and **exponential service times**, was almost equal to $\frac{1}{\mu-\lambda}$. Making the comparisons of the results we noticed that the system was working as we expected to be working so we can state that our simulation model is valid.

To perform the validation of the OMNeT++ model we changed in the code the Vacation duration to $0ms$ so the Server will still go on vacation and it will still handle turns. We had to modify some parts of Server.cc because there were some problems related to putting $Vacations = 0ms$: if the Server goes on vacation when there are no jobs in the queue, it will wake up after $0ms$ and will find the queue empty again and will schedule another vacation of $0ms$, in this way the Server will fall in an infinite loop because the simulated time cannot advance and an arriving job has no time to queue up since its related event won't be handled by the Server. To solve this issue we used a boolean variable *working*: *working = true* if the server is handling a job, otherwise *working = false*. When the server receives a job, if *working = true* then the arriving jobs is putted in the queue, otherwise the server starts its service and sets *working = true*.

When the server finishes to serve a job and the turn isn't finished yet, it checks whether there are any another jobs in the queue or not. If there aren't any other jobs, it sets *working=false* and goes idle instead of going on vacation, in this way we avoided the infinite loop. In the end it will re-start its service whenever a new job arrives. Otherwise, if there are jobs in the queue, it evaluates the requested service time of the new job and if it is larger than the remaining time goes on vacation in order to build a turn large enough. Since vacations are 0ms long, it immediately builds a turn large enough and starts the service. This means that the system works as an M/M/1.

The Server is initialized with **working=false** and it starts on an idle state.

```
#include "Server.h"
#include "Job_m.h"

Define_Module(Server);

void Server::initialize() {

    deficit = 0.0;
    constant = par("constant");
    delay_signal = registerSignal("delay");
    Q = par("turn_time");
    vacation_period = par("vacation_time");
    end_of_vacation = new cMessage("vacation");
    end_of_service = new cMessage("service");
    //scheduleAt(simTime(), end_of_vacation);
}

void Server::handleMessage(cMessage *msg) {

    if (msg->isSelfMessage()) {

        //if the vacation is ended i start a service turn
        if (strcmp(msg->getName(), "vacation") == 0) {
            if (queue.size() != 0) {
                //Check the size of the next job to serve
                //If it's small enough we serve it, otherwise we
    start a new vacation
                //and get the deficit for the next turn.
                simtime_t job_duration = queue[0].service_time;
                if (job_duration > Q + deficit) {
                    deficit = deficit + Q;
                    startVacation();
                } else {
                    deficit = deficit + Q - job_duration;
                    startService();
                }
            } else {
                //If true, it take the const value, otherwise gets
    it form exponential
                deficit = 0.0;
                EV << "Empty Queue" <<  endl;
                startVacation();
            }
        } else {
            working = false;
            //We calculate the response time and remove the job
    from the queue
            getResponseTime();
            //Here the service is done, so we look for other jobs
    or we go in vacation
            if (queue.size() != 0) {
                simtime_t job_duration = queue[0].service_time;
                if (job_duration > deficit) {
                    startVacation();
                } else {
                    deficit = deficit - job_duration;
                    startService();
```

```
                }
            } else {
                deficit = 0.0;
                EV << "Empty Queue" << endl;
                //startVacation();
            }

        }
    } else {
        //At this point the server is on vacation or is serving
    another job, so
        //we just add the new job to the queue.
        Job *job = check_and_cast<Job*>(msg);
        struct info_holder job_info_holder;
        job_info_holder.arrival_instant = simTime();
        job_info_holder.service_time = SimTime::parse(job->
    getService_time());
        queue.push_back(job_info_holder);
        EV << "New entry in queue, queue size: " << queue.size() <<
     endl;
        if(working==false){
            working = true;
            startService();
        }
    }
}

void Server::startService() {
    working = true;
    simtime_t job_duration = queue[0].service_time;
    EV << "Begin Service of: " << job_duration << endl;
    EV << "New Deficit: " << deficit << endl;
    scheduleAt(simTime() + job_duration, end_of_service);
}

void Server::startVacation() {
    if (!constant) {
        simtime_t mean_period_time = par("vacation_time");
        vacation_period = exponential(mean_period_time, 0);
    }

    EV << "Begin Vacation of: " << vacation_period << endl;
    EV << "New Deficit: " << deficit << endl;
    scheduleAt(simTime() + vacation_period, end_of_vacation);
}

void Server::getResponseTime() {
    simtime_t now = simTime();
    simtime_t response_time = now - queue[0].arrival_instant;
    emit(delay_signal, response_time);
    queue.erase(queue.begin());
}
```

5 Constant case

We started analyzing the system with constant values for inter-arrival times, service times and vacation period. We tested it in limit cases and observed that, for different values of Q the system behaves in a different way. In particular, **the smaller the value of the turn, the higher needs to be the inter-arrival time, maintaining all the other values equal**. For instance, let's analyze the case for $Q = \frac{S_t}{2}$: since the turn time is half the time required for a job to be served, the system will have to go on vacation two times to handle the request of the job, one in order to build a turn big enough, one after the job is served. Knowing that, we can easily state that every time the system will have to take two vacations and one service time for each job that arrives.
In the end we wrote a stability condition for our system:

$$Interarrival\_time \geq Vacation \cdot \frac{Service\_time}{Q} + Service\_time$$

To prove our statement we simply changed values here and there and plotted the results. Note that, if $Q \to \infty$, then the first addendum of the above formula goes to 0, meaning that our system behaves like an M/M/1 system, which is the same as stating that our server doesn't go on vacation.

5.1 Testing the hypothesis

Here we report some plots where we have considered these two cases for different values of Q(1/10, 1 and 10 service times):

1. $Interarrival\_time = Vacation \cdot \frac{Service\_time}{Q} + Service\_time$ ■
2. $Interarrival\_time < Vacation \cdot \frac{Service\_time}{Q} + Service\_time$ ■

Q=3ms

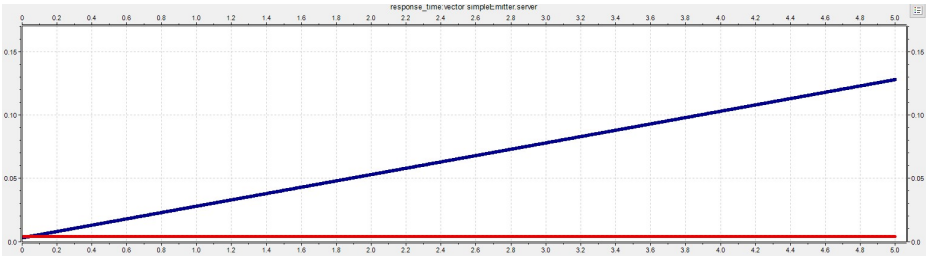

Figure 4: Interarrival= 3,9ms e 4ms; Service time=3ms; Vacation=1ms.

Q=30ms

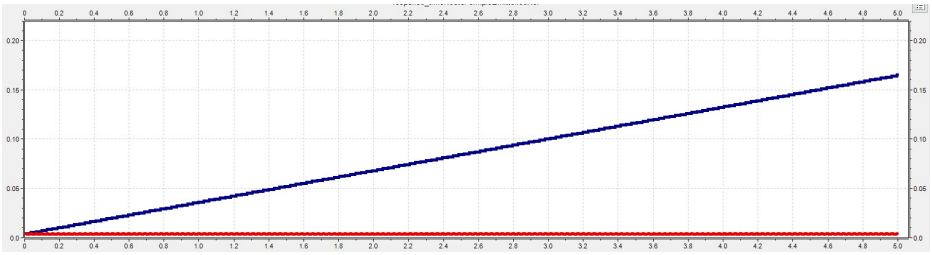

Figure 5: Interarrival= 3ms e 3,1ms; Service time=3ms; Vacation=1ms.

Q=0,3ms

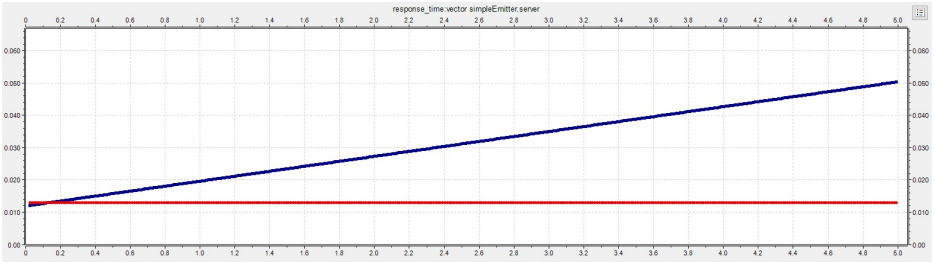

Figure 6: Interarrival= 12,9ms e 13ms; Service time=3ms; Vacation=1ms.

6 Exponential case

From now on we will use, for simplicity, μ for the Mean Service time; δ for the Mean Vacation period; λ for the Mean Inter-arrival time.

6.1 Tuning of the parameters for the simulations

We performed the tuning of the parameters for the simulations for 5 different scenarios because we noticed that it was necessary to have different values for the parameters depending on the ratio between Q and S_t ($\frac{1}{10}, \frac{1}{2}, 1, 5, 10$) in order to make the system stable.

The min value and max value for **inter-arrival time** have been estimated using some information found on data-sheets.

The min and max values for the service time and the vacations have been estimated taking in consideration that should be:

$$\frac{service\_time}{Q} * vacations + service\_time < inter\_arrivals$$

Then, for each case we have made different trials and we have selected a range of values that made the system stable in both the best and worst scenarios, but still had some interesting values, meaning that it does involve queuing and may have peaks of response time depending on the variability of the factors. We also decided to variate only the values of the inter-arrival time, maintaining the same two couple of values for vacations and service time because in this way we can easily state that with big values of Q we can use lower inter-arrival times making the system faster and for small values of Q we can make it extremely slower and sensible to variability.

6.1.1 Scenario $Q = S_t$

	Min	Max
inter-arrival time	$16ms$	$20ms$
service time	$3ms$	$7ms$
vacation	$1ms$	$5ms$

Table 1: min and max for the mean value of the parameters in the case $Q = S_t$

6.1.2 Scenario $Q = 10S_t$

	Min	Max
inter-arrival time	$8ms$	$12ms$
service time	$3ms$	$7ms$
vacation	$1ms$	$5ms$

Table 2: min and max for the mean value of the parameters in the case $Q = 10S_t$

6.1.3 Scenario $Q = \frac{1}{10}S_t$

	Min	Max
inter-arrival time	$77ms$	$81ms$
service time	$3ms$	$7ms$
vacation	$1ms$	$5ms$

Table 3: min and max for the mean value of the parameters in the case $Q = \frac{1}{10}S_t$

6.1.4 Scenario $Q = \frac{1}{2}S_t$

	Min	Max
inter-arrival time	$22ms$	$26ms$
service time	$3ms$	$7ms$
vacation	$1ms$	$5ms$

Table 4: min and max for the mean value of the parameters in the case $Q = \frac{1}{2}S_t$

6.1.5 Scenario $Q = 5S_t$

	Min	Max
inter-arrival time	$10ms$	$14ms$
service time	$3ms$	$7ms$
vacation	$1ms$	$5ms$

Table 5: min and max for the mean value of the parameters in the case $Q = 5S_t$

6.2 Estimation of the warm-up period

To estimate the duration of the warm-up period, we did 40 *simulations* (5 independent repetitions of 8 different scenarios).

The warm-up period was estimated taking into consideration all the different cases shown above. Here we report two plots of the sliding window average in two of the cases in which $Q = S_t$ and $Q = 10S_t$.

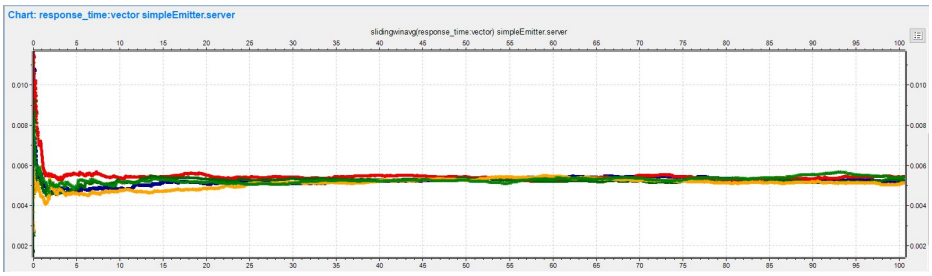

Figure 7: Sliding window average of the response time (case $Q = S_t$)

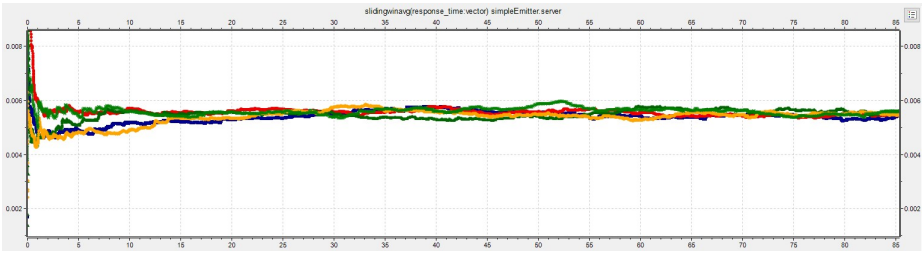

Figure 8: Sliding window average of the response time (case $Q = 10S_t$)

As we can notice from figure 7 and 8, the duration of the warm-up period can be estimated at $3s$.

6.3 $2^k r$ factorial analysis

In order to understand which factor affects the most the mean response time, we performed the $2^k r$ factorial analysis with 3 factors (inter-arrivals λ, services μ and vacations δ), 5 repetitions and $Q = \left\{\frac{1}{10}S_t, \frac{1}{2}S_t, S_t, 5S_t, 10S_t\right\}$, for Q we chose to perform the $2^k r$ factorial analysis both in the case with $Q = S_t$ (Variable Q case) and $Q = \frac{S_{t_{Min}} + S_{t_{Max}}}{2}$ (Constant Q case). For the cases $Q = \frac{1}{2}S_t$ and $Q = 5S_t$ we only analyzed the variable Q case.

With the factorial analysis we noticed that the *Constant Q case* brings out some issues. In fact, let's assume we have Q = 5ms, and the service time has a min value of 3ms and a max of 7ms. It's easy to see that for the case in which the service time is lower, the system will be faster and less sensible to variations, since even if we end up having a big job, it will probably be served within the turn. That does the exact opposite for the case in which we have the highest service time, this time the job will almost every time make the system go on vacation once in order to be served. For bigger values of Q this issue is negligible, since we will have Q = 50ms, and the worst case for the service time is 7ms, so we will end up serving, on average, 7 jobs before the vacation, instead of 10, while for a service time of 3ms we will have one vacation every 16 jobs, so for both cases we will add, or remove, just a few more vacation, so we won't increment, or decrease, a lot the mean response time of the system. If we check this for the system with Q = 0,5ms then for the service time at 3ms we will need 6 vacations in order to serve one job, if instead the service time is at 7ms, we'll end up taking 14 vacations for each job, meaning that we will change a lot the value of the mean response time. This leads to an interesting consideration about the system itself: **the smaller the value of $\frac{Q}{S_t}$ is, the more the system will be sensible to small variations of Q.**

For the simulations used to collect the data needed to perform the $2^k r$ factorial analysis we used the values found with the tuning for the factors.

For what concerns testing the hypothesis, we tested that residuals were normally distributed using a normal QQ plot, while we created a scatterplot of the residuals vs. the predicted response to test constant std. We also wanted to test IIDness using a scatterplot of the residuals vs the observation number, but we didn't know the sequence of observations so we assumed there was no bias.

6.3.1 Case $Q = S_t$

Results

As we can see from the table of the variations, in this case the factors that matter the most are *vacations*, *service times* and the *interplay of vacations and service times*. This makes sense since the system, on average, will serve a job and then start a vacation.

Percentage of variation of the factors and the error							
λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$	ϵ
4.48%	33.83%	48.39%	2.37%	2.59%	6.87%	1.31%	0.16%

Confidence interval with 98% confidence level							
	λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$
CI-	-0.00338	0.00838	0.01008	-0.00252	-0.00263	0.00364	-0.00194
CI+	-0.00289	0.00887	0.01057	-0.00203	-0.00214	0.00413	-0.00145

Testing hypothesis

In this case the normality assumption is verified since the QQ plot shown in Figure 9 looks reasonably linear. We can also assume the standard deviation as constant, despite the plot shows a certain trend (Figure 10). In fact, we can ignore it since the errors are one order of magnitude below the predicted response.

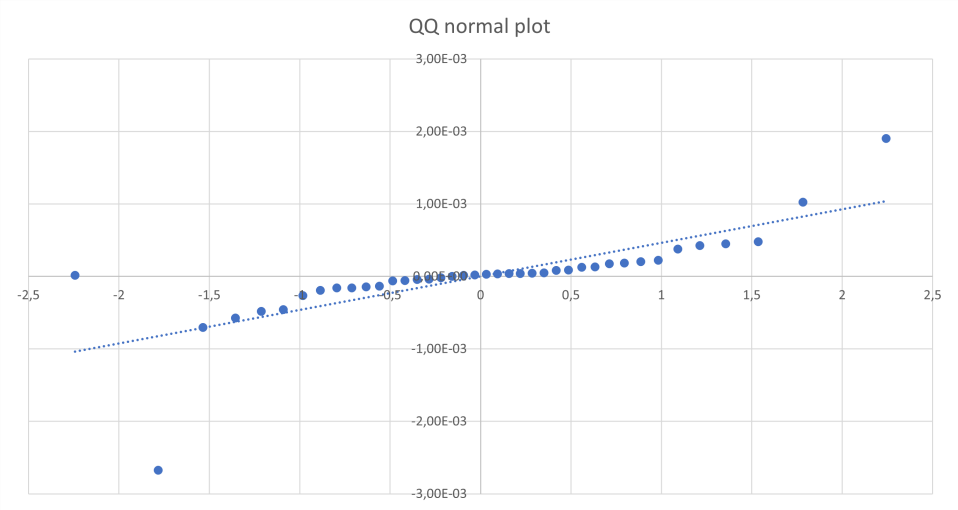

Figure 9: QQ plot for $Q = S_t$

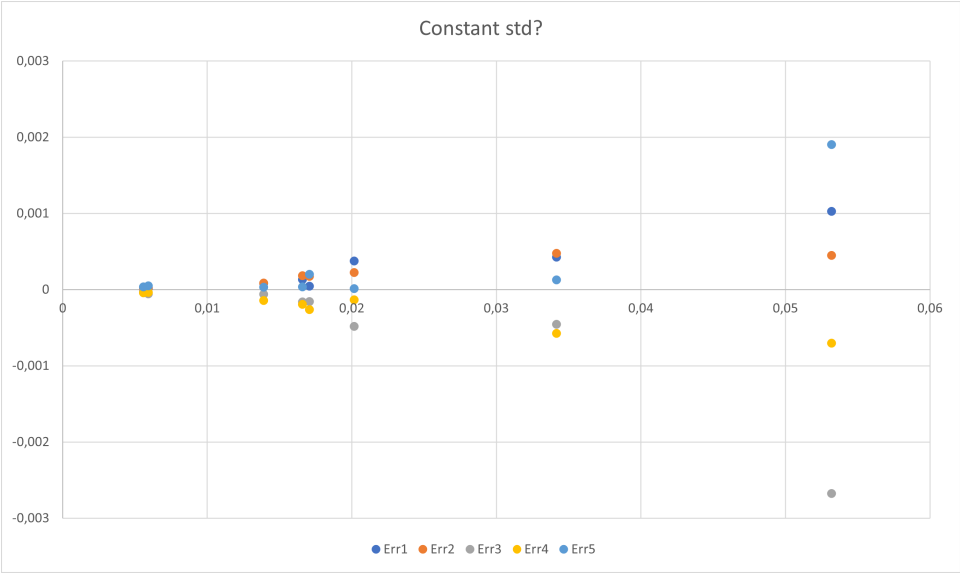

Figure 10: Scatterplot for $Q = S_t$

6.3.2 Case $Q = 10S_t$

Results

In the case $Q = 10S_t$, the factors *service times*, *inter-arrivals* and the *interplay of inter-arrivals and services* are the ones that affect the most the mean response time. This is explainable by the fact that turn time is big enough to serve many jobs during a single turn, so the server rarely goes on vacation, that's why it does not really affect the performances of the system. +

Percentage of variation of the factors and the error							
λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$	ϵ
15.96%	62.25%	5.67%	13.96%	0.61%	0.87%	0.56%	0.11%

Confidence interval with 98% confidence level							
	λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$
CI-	-0.00596	0.01119	0.00324	-0.00558	-0.00132	0.00115	-0.00127
CI+	-0.00557	0.01158	0.00363	-0.00519	-0.00093	0.00154	-0.00088

Testing hypothesis

Since the QQ plot in Figure 11, except for a few points, looks approximately linear, we can state that the assumption of normality is verified.

From Figure 12 we can notice that the standard deviation doesn't look constant but if we pay attention to the axis' scales, we can see that the ys are

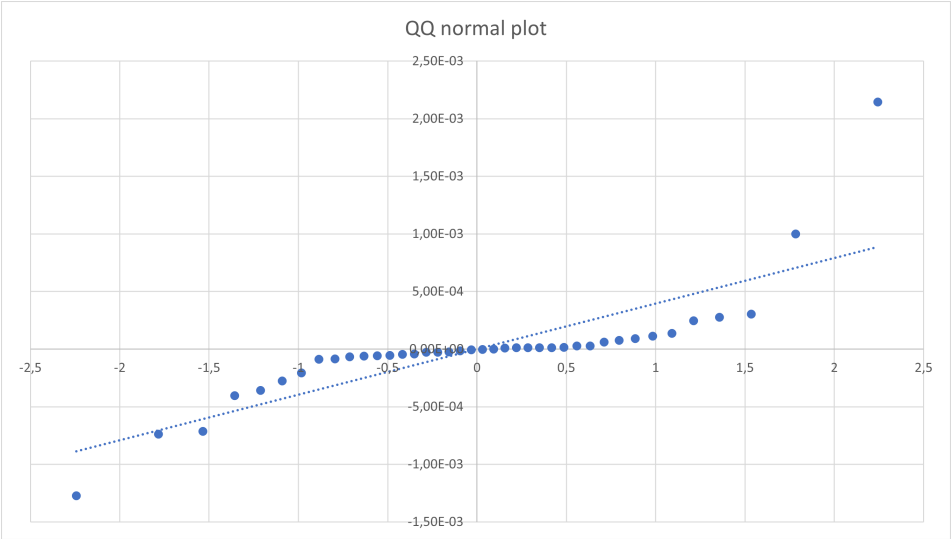

Figure 11: QQ plot for $Q = 10S_t$

significantly smaller than the xs, as a consequence we ignore the trend and we consider the standard deviation to be approximately constant.

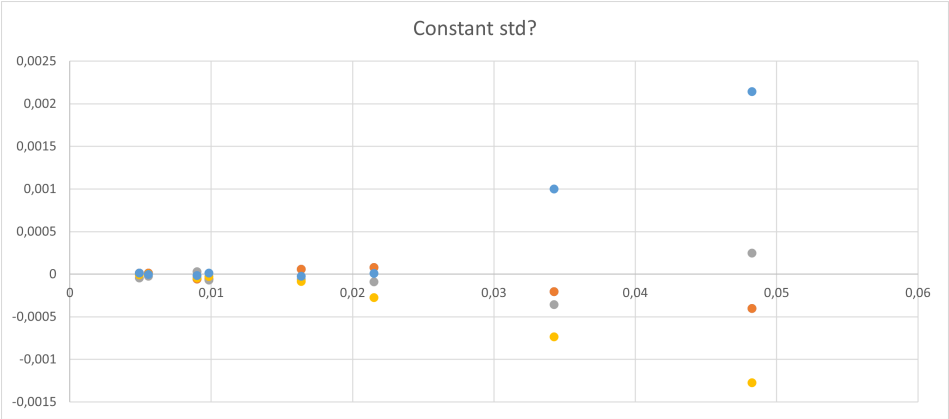

Figure 12: Scatterplot for $Q = 10S_t$

6.3.3 Case $Q = \frac{1}{10}S_t$

Results

Vacations is the most affecting factor on the performance of the system, this is due to the fact that in this case the server needs more vacations to build up a turn big enough to serve an entire job, i.e. the bigger the vacation is, the higher will be the response time. It's also important to note that this system is very likely to be unstable since it's extremely sensible to small variations.

Percentage of variation of the factors and the error							
λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$	ϵ
0.38%	2.04%	95.29%	0.00%	0.36%	1.17%	0.00%	0.75%

Confidence interval with 98% confidence level							
	λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$
CI-	-0.00847	0.00931	0.08180	-0.00347	-0.00835	0.00628	-0.00343
CI+	-0.00226	0.01553	0.08801	0.00274	-0.00213	0.01249	0.00277

Testing hypothesis

The QQ plot can be assumed linear, meaning that the residuals are normal. About the standard deviation, even though it might show a trend, since residuals are one order of magnitude below the predicted response, we can consider it constant.

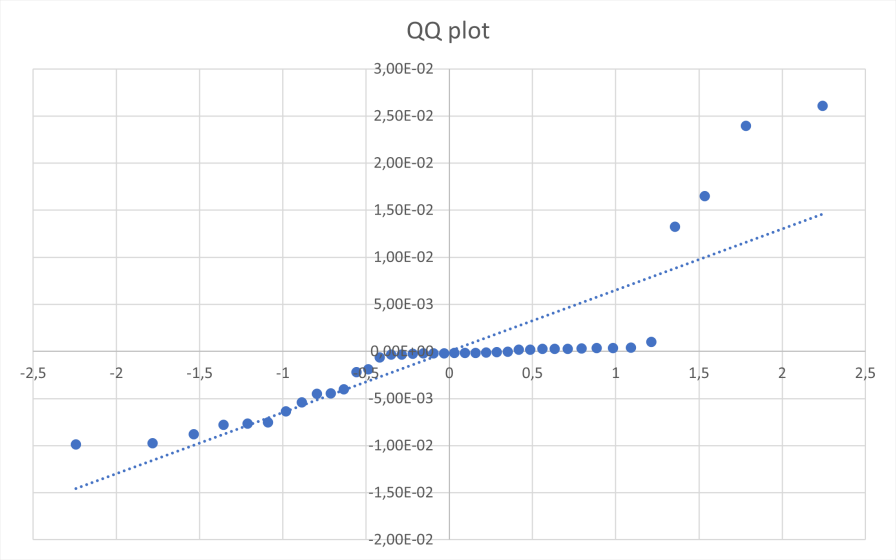

Figure 13: QQ plot for $Q = \frac{S_t}{10}$

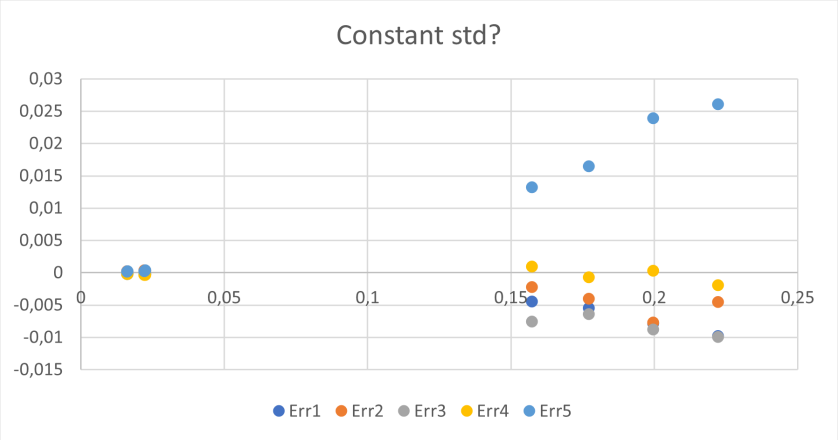

Figure 14: Scatterplot for $Q = \frac{S_t}{10}$

6.3.4 Case $Q = \frac{1}{2}S_t$

Results

In this case we can easily see that *Vacation* is the factor that most affects the system while the service time has got a lower impact. This leads to the conclusion that **the smaller the value of Q gets, the more the system is affected by the value of the vacation, while the service time becomes less relevant**.

Percentage of variation of the factors and the error							
λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$	ϵ
3.34%	18.65%	65.74%	1.28%	2.70%	7.12%	1.02%	0.14%

Confidence interval with 98% confidence level							
	λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$
CI-	-0.00492	0.01033	0.01974	-0.00319	-0.00447	0.00623	-0.00289
CI+	-0.00415	0.01111	0.02052	-0.00242	-0.00369	0.00701	-0.00211

Testing hypothesis

Also in this case we can consider the assumptions as verified since the QQ plot in Figure 15 can be assumed as linear. The same goes for the standard deviation that can be considered as constant since we can ignore the trend thanks to the fact that the errors are one order of magnitude below the predicted response.

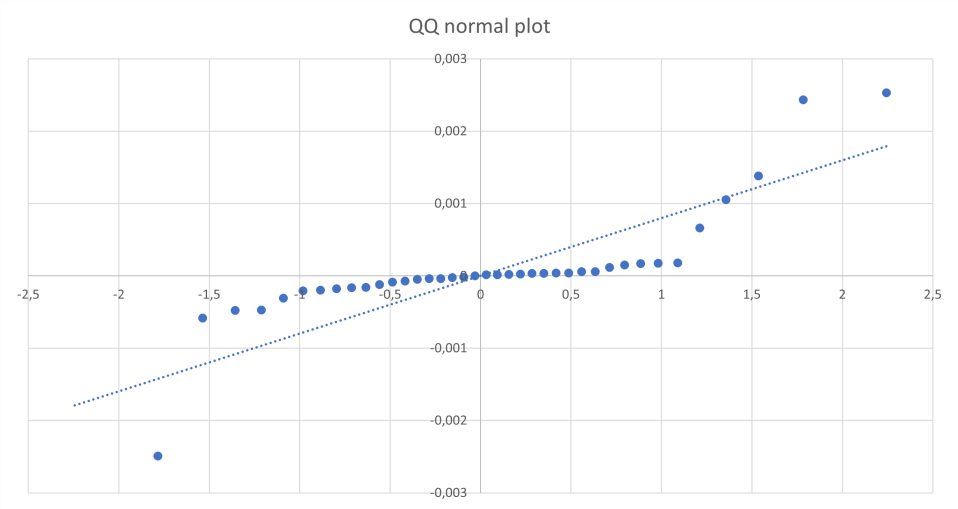

Figure 15: QQ plot for $Q = \frac{S_t}{2}$

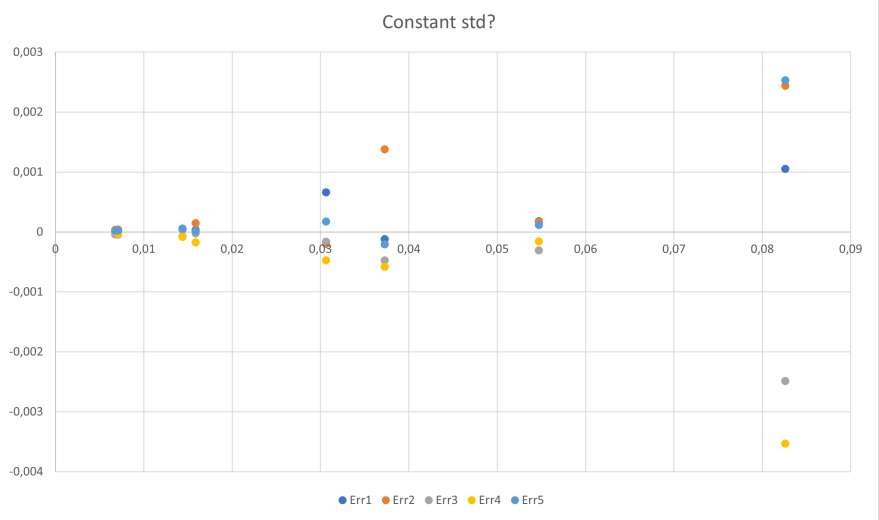

Figure 16: Scatterplot for $Q = \frac{S_t}{2}$

6.3.5 Case $Q = 5S_t$

Results

In the case $Q = 5S_t$, the *Service time* is the most affecting factor on the performance of the system. We can see that while Q grows, the impact of the service time becomes more important, and the vacation becomes less meaningful. This makes perfect sense, in fact for an increasing value of Q, the system becomes more and more close to an M/M/1, meaning that the vacation has a smaller impact on the response time.

Percentage of variation of the factors and the error							
λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$	ϵ
12.11%	61.26%	11.72%	10.00%	1.38%	2.30%	1.15%	0.07%

Confidence interval with 98% confidence level							
	λ	μ	δ	$\lambda\mu$	$\lambda\delta$	$\mu\delta$	$\lambda\mu\delta$
CI-	-0.00450	0.00969	0.00416	-0.00410	-0.00160	0.00177	-0,00147
CI+	-0.00423	0.00995	0.00443	-0.00383	-0.00134	0.00203	-0.00121

Testing hypothesis

Once again the QQ plot looks approximately linear and the constant standard deviation assumption can be held by the fact that residuals are orders of magnitude smaller that the predicted response.

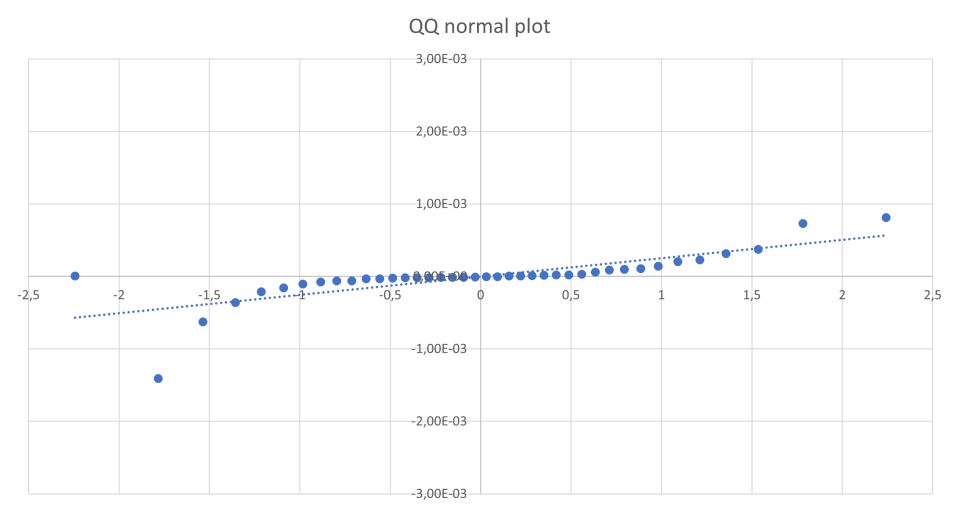

Figure 17: QQ plot for $Q = 5S_t$

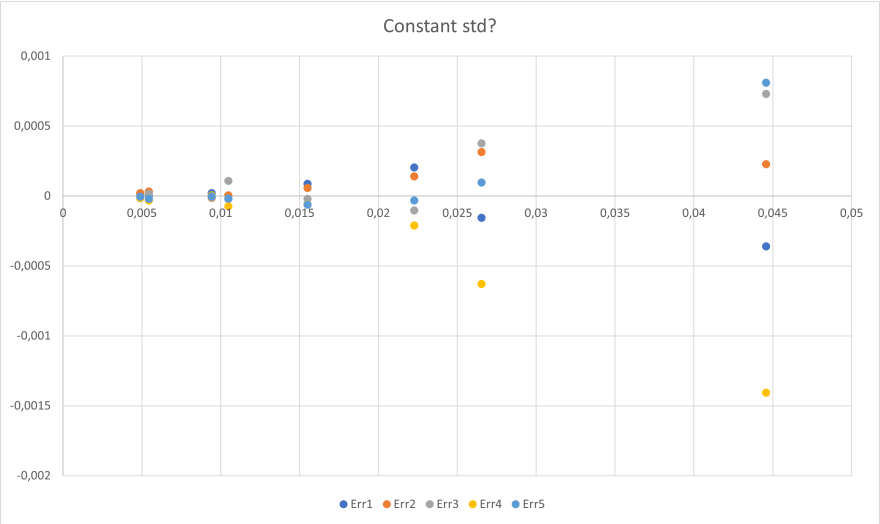

Figure 18: Scatterplot for $Q = 5S_t$

7 Data Analysis for the Exponential case

Once we obtained the results from the $2^k r$ analysis, we exploited them in the data analysis, with the aim of getting some insights on the system and doing different model fittings.

In order to perform the fitting we ran 15 replicas for each scenario of interest, meaning that we only changed the values of the factors that had most impact on the system. Out of the 15 replicas we took the average response time, and then from those we computed the sample mean, variance and CoV. We used QQ plots in order to test the assumption of normality and computed the confidence intervals for every sample mean. Then we took the sample mean for each scenario and tried to find a model that fitted our results. None of them looked linear at all, but after a simple transformation we managed to fit them into a linear model that allowed us to predict, with a specific confidence range, the average behavior of our system when varying one or two factors inside the range of interest. For each regression we computed the confidence interval for the slope, the offset and for the predicted response, with a confidence level of 95%.

7.1 Case $Q = \frac{1}{10}S_t$

As a result of the $2^k r$ analysis we obtained that the factor with the higher variation was δ ($95,29\%$ of variation) so we used fixed values for inter-arrival time and service time while we made the vacation varying between $1ms$ and $5ms$ with steps of $0.25ms$.

7.1.1 Linear regression for δ

Settings for the simulations

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${77ms}
**.emitter.requested_service_time = ${7ms}
**.server.vacation_time = ${1ms, 1.25ms, 1.5ms, 1.75ms, 2ms, 2.25ms
    , 2.5ms, 2.75ms, 3ms, 3.25ms, 3.5ms, 3.75ms, 4ms, 4.25ms, 4.5ms
    , 4.75ms, 5ms}
**.server.turn_time = ${0.7ms}
```

Fitting
Plotting the mean of the response time of each scenario against the values of δ, we noticed that the distribution was perfectly fitting an exponential (Figure 19), then we applied a transformation in order to fit the distribution with a linear model (Figure 20) and in the end we tested the assumptions and we verified that the assumptions were met.

The result obtained can be explained by the fact that when you increase the value of the vacation the mean response time will suffer a huge increase, since for smaller values of Q the system will need a larger number of vacations in order to build a turn big enough to serve one job.

Moreover, the model is also valid for values outside our case study range, for big δs the system is unstable and the model approaches ∞, while **for small values of δ, the system behaves like an M/M/1**, and exactly like one in which the vacation is null, meaning that the mean response time will be like the theoretical one. Our regression model tends to that value for $\delta \to 0$.

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	558.8994146	-4.275803281
Lower Limit	558.8605854	-4.300396719

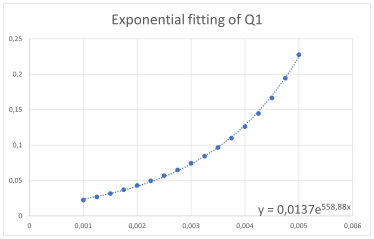

Figure 19: Fitting with the exponential

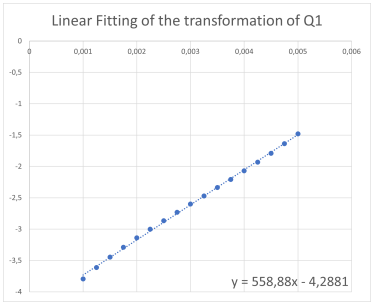

Figure 20: Transformation of the exponential for linear regression

7.2 Case $Q = \frac{1}{2}S_t$

In this case what we got was that δ accounted for the $65,74\%$ of the variation, so we fixed the inter-arrival time and service time and changed the vacation between $1ms$ and $5ms$ with steps of $0.5ms$.

7.2.1 Linear regression for δ

Settings for the simulation

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${22ms}
**.emitter.requested_service_time = ${7ms}
**.server.vacation_time = ${1ms, 1.5m, 2ms, 2.5ms, 3ms, 3.5ms, 4ms,
    4.5ms, 5ms}
**.server.turn_time = ${3.5ms}
```

Fitting

From Figure 21 we can see that the distribution of the mean of the means of the response time obtained by varying δ, is fitting well the exponential, for the same reasons of the previous case.

In Figure 22 it is shown the transformation of the exponential applied in order to perform the linear fitting.

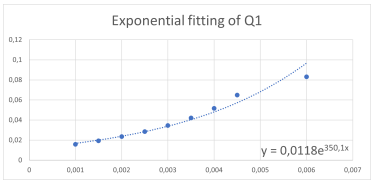

Figure 21: Fitting with the exponential

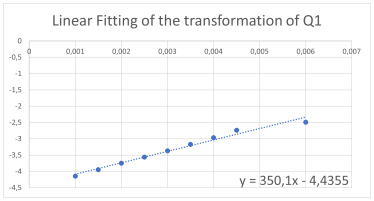

Figure 22: Transformation of the exponential for linear regression

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	350.2274218	-4.293371211
Lower Limit	349.9725782	-4.577628789

7.3 Case $Q = S_t$

Here what we achieved was that δ accounted for the $48,39\%$ of the variation and μ for the $33,3\%$. For this reason we decided to study two different cases, one in which we varied only the vacation and another one in which we varied only the service time.

Moreover we noticed that the interplay between δ and μ accounted for the $6,87\%$, so we decided to study that case as well.

7.3.1 Linear regression for δ

Settings for the simulations

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${16ms}
**.emitter.requested_service_time = ${7ms}
**.server.vacation_time = ${1ms, 1.5ms, 2ms, 2.5ms, 3ms, 3.5ms, 4ms
    , 4.5ms, 5ms}
**.server.turn_time = ${7ms}
```

Fitting

In Figure 23 is shown the scatterplot of the distribution obtained by varying the vacations and maintaining fixed the other factors, from the graph we can easily see that the distribution fits very well an exponential. This result means that in this scenario, **increasing the vacations really decreases the**

performances of the system in terms of response time and it is explainable by the fact that since Q is of the same order of magnitude of μ and μ is exponentially distributed, a big job could arrive (with a requested service time bigger than the value of Q), the server will have to go on vacation to build a bigger turn to serve it.

The linear transformation used for linear regression is shown in Figure 24.

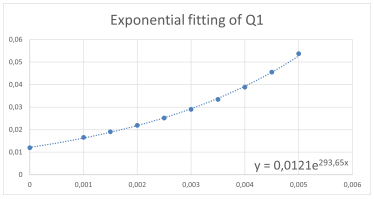

Figure 23: Fitting with the exponential

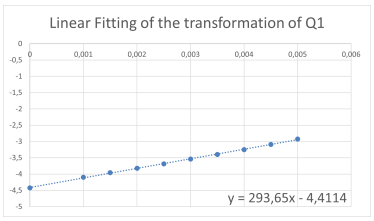

Figure 24: Transformation of the exponential for linear regression

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	293.6720203	-4.386202448
Lower Limit	293.6279797	-4.436597552

7.3.2 Linear regression for μ

Settings for the simulations

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${16ms}
**.emitter.requested_service_time = ${3ms, 3.5ms, 4ms, 4.5ms, 5ms,
    5.5ms, 6ms, 6.5ms, 7ms}
**.server.vacation_time = ${5}
```

Fitting

Also in this case the model which fits the data in the best way is the exponential one (Figure 25). This means that increasing the mean service time has a huge impact on the response time, which is reasonable if we think about how the system works.

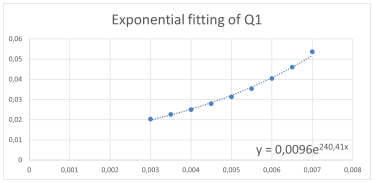

Figure 25: Fitting with the exponential

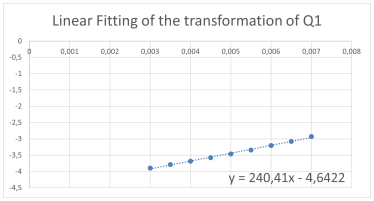

Figure 26: Transformation of the exponential for linear regression

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	240.4648084	-4.579350362
Lower Limit	240.3551916	-4.705049638

7.3.3 Linear regression for $\delta\mu$

Settings for the simulations

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${16ms}
**.emitter.requested_service_time = ${3ms, 4ms, 6ms, 7ms}
**.server.vacation_time = ${1ms, 2ms, 4ms, 5ms}
```

Fitting

Varying δ and μ, the distribution of the mean response time for the scenario $Q = S_t$ fits very well an exponential (Figure 27), this result makes sense if we think that the mean response time has an exponential behavior depending on both δ and μ. We decided to use as a variable for the linear regression the sum of δ and μ, this because since the ration between the turn time and the service time is equal to 1, the time that the server actually needs to completely serve a job is, on average, the sum of the two.

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	266.0997927	-6.077275135
Lower Limit	265.9802073	-6.249309157

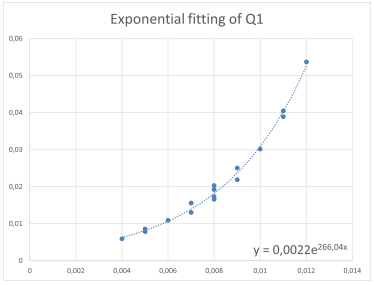

Figure 27: Fitting with the exponential

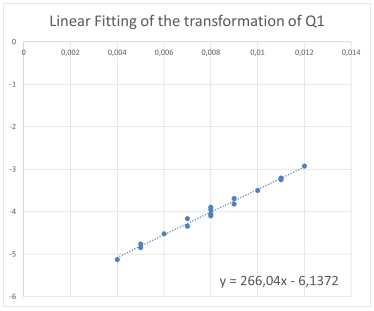

Figure 28: Transformation of the exponential for linear regression

7.4 Case $Q = 5S_t$

When we performed the data analysis for $Q = 5S_t$, we decided to study two cases in which we varied the most affecting factors discovered with the $2^k r$ factorial analysis that are the service time μ (61.26%) and the interplay of the service time and the inter-arrivals (10%).

7.4.1 Linear regression for μ

Settings for the simulations

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${10ms}
**.emitter.requested_service_time = ${3ms, 3.5ms, 4ms, 4.5ms, 5ms,
    5.5ms, 6ms, 6.5ms, 7ms}
**.server.vacation_time = ${5ms}
```

Fitting

In this case, the exponential is the model that best fits the distribution of the mean response time (Figure 29), it is reasonable if we think that Q is approximately equal to 5 times μ so the server can serve approximately 5 jobs before it goes on vacation (the server completes approximately 5 jobs in a single turn) so the vacation doesn't affect too much the response time, as a consequence keeping fixed the inter-arrivals of the jobs and increasing the service times, will cause a big increase of the response time.

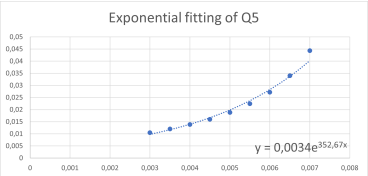

Figure 29: Fitting with the exponential

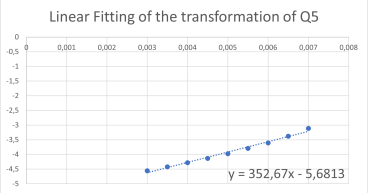

Figure 30: Transformation of the exponential for linear regression

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	352.7702998	-5.550308268
Lower Limit	352.5697002	-5.812291732

7.4.2 Linear regression for $\lambda\mu$

Settings for the simulations

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${10ms, 11ms, 13ms, 14ms}
**.emitter.requested_service_time = ${3ms, 4ms, 6ms, 7ms}
**.server.vacation_time = ${5ms}
```

Fitting

We decided to study the interplay of the two factors as the ratio between the Inter-arrival *rate* and the Service time *rate*, because it resembles the utilization for an M/M/1 system. In fact, for big values of the ratio between Q and the service time our server behaves in such way. The response time increases exponentially with the utilization of the M/M/1, and the system does the same thing, as expected.

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	3140.177707	-5.340380178
Lower Limit	3140.022293	-5.568380882

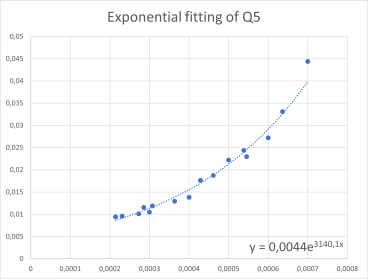

Figure 31: Fitting with the exponential

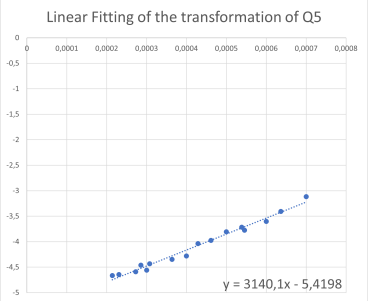

Figure 32: Transformation of the exponential for linear regression

7.5 Case $Q = 10S_t$

For $Q = 10S_t$, the most affecting factors on the performance of the systems are the service time μ (62.25% of variation), the inter-arrivals of the jobs (15.96% of variation) λ and the interplay of the two (13.96%), so during the data analysis of this case, we decided to study just the effects of those factors in the three cases documented below.

7.5.1 Linear regression for μ

Settings for the simulations

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${9ms}
**.emitter.requested_service_time = ${3ms, 3.5ms, 4ms, 4.5ms, 5ms,
    5.5ms, 6ms, 6.5ms, 7ms}
**.server.vacation_time = ${5ms}
```

Fitting

Also in this case we used an exponential model to fit our data (Figure 33). The response time grows exponentially with the value of service time for the same reasons explained in the case $Q = 5S_t$ where we varied the factor μ and we maintained fixed λ and δ.

7.5.2 Linear regression for λ

Settings for the simulations

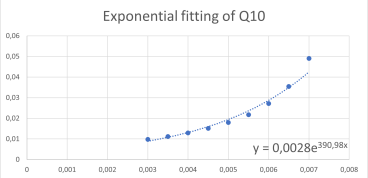

Figure 33: Fitting with the exponential.

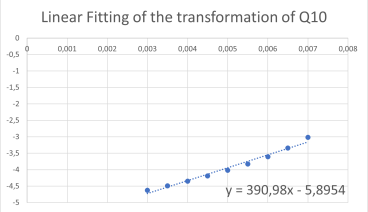

Figure 34: Transformation of the exponential for linear regression

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	391.1127699	-5.721008807
Lower Limit	390.8472301	-6.069791193

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${9ms, 9.5ms, 10ms, 10.5ms, 11ms,
    11.5ms, 12ms, 12.5ms, 13ms}
**.emitter.requested_service_time = ${7ms}
**.server.vacation_time = ${5}
**.server.turn_time = ${70ms}
```

Fitting

As we can see from Figure 35, we used an exponential to model the distribution of the response time even in this case. The practical explanation is that even if Q is very big compared to the service time, if the inter-arrivals of the jobs are very frequent then the jobs may queue up and the response time grows, if instead the arrivals are not so close between each other, then the server can easily serve the jobs with a small response time. Note that this model works properly only inside the range of our interest. In fact, assuming that we take a very large value for λ, in this model the response time approaches 0, but that doesn't make sense in the real system where the mean response time is lower bounded by a value depending on the actual service time and vacation period, also λ can't be too small because of of the limit that we found for the constant case: $\lambda \geq \delta \frac{\mu}{Q} + \mu$.

7.5.3 Linear regression for $\lambda\mu$

Settings for the simulations

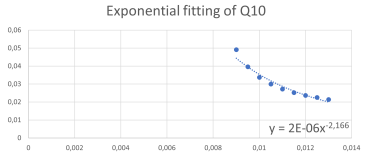

Figure 35: Fitting with the exponential

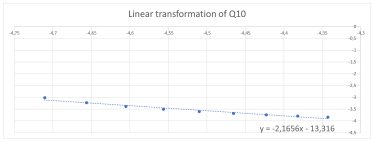

Figure 36: Transformation of the exponential for linear regression

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	-1.983521893	-13.10149438
Lower Limit	-2.347678107	-13.53050562

```
warmup-period = 3s
repeat = 15
seed-set = ${repetition}
sim-time-limit = 1000s
**.emitter.interarrival_time = ${9ms, 10ms, 12ms, 13ms}
**.emitter.requested_service_time = ${3ms, 4ms, 6ms, 7ms}
**.server.vacation_time = ${5ms}
```

Fitting

Considering the interplay between λ and μ we obtained once again an exponential fitting. The reasons behind this behavior are the same illustrated for the case $Q = 5S_t$.

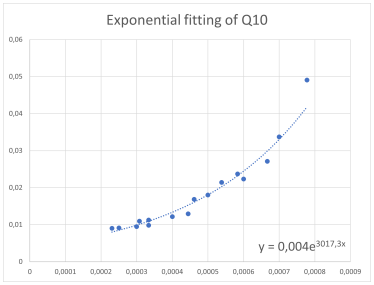

Figure 37: Fitting with the exponential

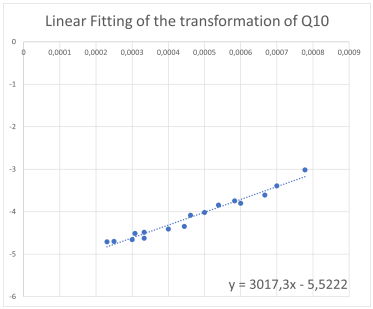

Figure 38: Transformation of the exponential for linear regression

Confidence interval with 95% confidence level		
	Slope	Offset
Upper Limit	3017.401167	-5.417960402
Lower Limit	3017.198833	-5.717214431

8 Conclusions

As a last consideration, we observed how our system behaves in different ways depending on the ratio between service time and turn time. When the ratio is close to 1, the response time will depend mostly on *Service time* and *Vacation*, while as the ratio starts to go below one, δ will get more and more influential and μ will lose importance. This makes sense since when the ratio is really small, the server has to go on vacations many times in order to build a turn big enough to serve a job. This also explains why when we are dealing with small values of Q we use very large values for the inter-arrival times in order to maintain the system stable. On the other hand, as the ratio between the turn time and service time grows bigger, the vacation becomes insignificant for what concerns the response time since the turn is big enough to serve many jobs before going on vacation while inter-arrival time and service time will impact more on the system.