BACHELOR

An extension of join-the-shortest-queue(d)
a fluid limit approach

Arts, Jor J.P.E.

Award date:
2019

Technische Universiteit
Eindhoven
University of Technology

5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

An extension of Join-the-Shortest-Queue(d): a fluid limit approach

Bachelor final project

Author

Jor Arts (0910273)

Supervisors

Mark van der Boor
Sem Borst

Date

April 2, 2019

2 - An extension of Join-the-Shortest-Queue(d):
a fluid limit approach

Abstract

In computer communication networks, such as the Internet, data packets are transmitted through large groups of servers. In order to reduce waiting time at the buffer of a server, a dispatcher is used that executes a certain scalable load balancing algorithm (LBA). One such LBA is the JSQ(d) policy. This policy picks d servers from the total at random and forwards the packet to the server that has the shortest queue. In this report we aim at extending this well known policy by adding an extra kind of service to the system, representing the time it takes a data packet or simply task to arrive at the buffer of a server. We will refer to this as the JSQ(d, δ) policy, where $\frac{1}{\delta}$ represents the average time of the extra service. For this JSQ(d, δ) policy we will aim to derive a fluid limit. We then will provide a simulation to compare to our fluid limit and to visualise some properties of our model. Using simulation we will also look at the JSQd(d, δ) policy. This policy differs from the JSQ(d, δ) policy in that the average time of the extra service now also depends on d, which should be interpreted as a communication burden or time penalty when d increases.

Contents

1 Introduction

In this report, we will take a look at a specific type of a scalable load balancing algorithm (often referred to as LBA's). Load balancing algorithms or policies serve to distribute service requests or tasks among servers. Those tasks can for example be file transfers, data base look-ups or computing jobs. There are many well known load balancing algorithms and several of them are mentioned and explained in [3]. The goal that load balancing algorithms try to achieve, is to distribute the tasks evenly over the servers in order to reduce the waiting time of the tasks. Load balancing algorithms have attracted strong attention in recent years. This is mainly spurred by crucial scalability challenges arising in cloud networks and data centers, like Google and YouTube, with massive numbers of servers. One problem that arises here is that the number of messages to receive information about the total system state would grow linearly with the number of servers, causing a communication burden.

The load balancing algorithm of interest in this report is an extension of the well known Join-the-Shortest Queue (JSQ) policy. The JSQ policy keeps track of the number of tasks that all the servers in the system have, and when a new task arrives, the JSQ policy sends the new task to the server with the shortest queue (with ties being broken arbitrarily).

However, as we just mentioned, a communication burden may arise when using this policy, since for every incoming task the number of messages it takes to receive all queue lengths equals twice the number of servers, because we need one message to request and one to receive the number of tasks per server. So this overhead becomes a problem as the number of servers grows large. Therefore the JSQ(d) policy has been introduced. This policy only determines the queue lengths of d servers. This will create a balance between choosing the shortest queue and the communication burden. The benefits of this policy are that now the number of messages needed equals only a fixed value of $2d$. [1] [3]. In Figure 1.1 a schematic representation of the system with the JSQ/JSQ(d) policy is shown.

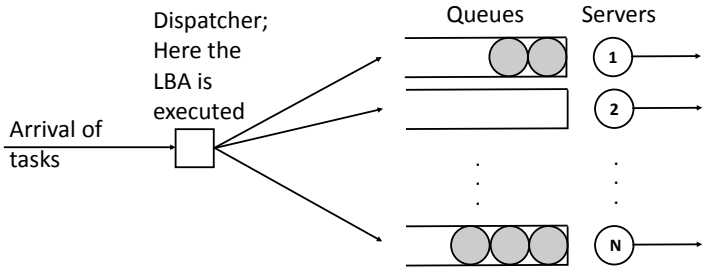

Figure 1.1: A schematic representation of a load balancing algorithm

The policy that we consider is closely related to JSQ(d). We will refer to it as the JSQ(d, δ) policy. The underlying policy is similar to the JSQ(d) policy. However, in the JSQ(d, δ) policy we will take into account that whenever a task is forwarded by the dispatcher to a server, it takes a certain amount of time to arrive at this server. We note that in practice this time may be extremely small, but on the other hand we note that this may also hold for the time it takes a server to process a task. Here $\frac{1}{\delta}$ is the mean time it takes to arrive at the server.

We will analyse this JSQ(d, δ) system and try to derive a fluid limit for it. The fluid limit of such a system can be interpreted as the scaled queue occupancy state as the number of servers grows large. Furthermore we will take a closer look at the behaviour of this system using simulations, which give us insightful plots. We will for example look at the queue length distribution when the system is in equilibrium.

Finally we will also briefly take a look at a variant where the mean time to arrive at the server depends on d and δ instead of only δ. This can be seen as giving a time penalty when we increase d, because this contributes to the communication burden. Here we will take a look at the influence of this change on the mean waiting time of a task.

2 Model description

2.1 Basic model

The models that we want to analyse are very similar to the model that is analysed in [3]. They refer to it as the JSQ(d) policy. Therefore, the notation that we will use will be similar. Since the models that we will be looking at are extensions of the model used in [3], let us first discuss their model. A schematic display is shown in Figure 2.2 on page 11.

We have a system with N parallel single-server infinite-buffer queues and a single dispatcher where tasks arrive as a Poisson process of rate λN, $\lambda < 1$. The dispatcher then sends any incoming task to one of the servers. At all the servers the service time is exponentially distributed with mean 1 and the service discipline is FIFO. The dispatcher has FIFO service discipline as well. However note that the system as a whole is in general not necessarily FIFO. The load balancing algorithm is called Power-of-d or JSQ(d). This means that whenever a task arrives at the dispatcher it will be sent to the server with the shortest queue among d randomly selected servers with ties being broken arbitrarily (note that the choice of d servers is without replacement) [1]. Also the vector $\mathbf{Q}(t) := (Q_1(t), Q_2(t), ...)$ as introduced in [3] will play a major role. Here $Q_i(t)$ with $i = 1, 2, ..$ denotes the number of servers with i or more tasks waiting at time t, including the possible task in service. This is schematically shown in Figure 2.1 below. Next, let us introduce $q_i(t) := Q_i(t)/N$ to be the fraction of servers with i or more tasks waiting at time t and define $\mathbf{q}(t) := (q_1(t), q_2(t), ...)$. It appears to be convenient to look at the fraction, since those values do not tend to infinity if the number of servers, N, tends to infinity.

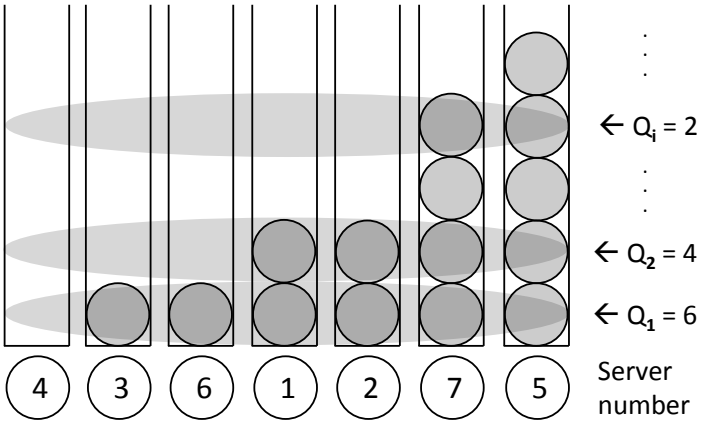

Figure 2.1: A schematic representation of $Q_i(t)$, the number of servers with at least i tasks

2.2 Extended models

Now that it is clear how the model in [3] looks like, we will explain what will be different in our models.

Our first model is schematically displayed in Figure 2.3 at the end of this section. We will refer to it as the JSQ(d, δ) policy. The element of waiting time for receiving the queue lengths has been implemented by introducing N fictional servers that can immediately start serving any incoming task. This can at first sight be seen as N $G|M|\infty$ queues, where the service time is exponentially distributed with rate δ. We will refer to them as Xservers.

In order to give a clear description of the state of the system, we will expand $Q_i(t)$ to $Q_{i,j}(t)$, where i will now describe the number of tasks in the Xservers and j the number of tasks that are in the corresponding real servers. So $Q_{i,j}(t)$ with $i, j = 0, 1, 2, ..$ will denote the number of servers with j or more tasks waiting at time t and with i or more tasks in the corresponding $G|M|\infty$ queue and similarly we will use $q_{i,j}(t)$ to represent the corresponding fraction of servers. Those values $Q_{i,j}(t)$ and $q_{i,j}(t)$ will be stored in matrices, which we will call **Q(**t**)** and **q(**t**)** respectively. Note however that here the case $i = j = 0$ is possible. In this case we will see that for every server and Xserver it holds that they have at least 0 tasks, meaning that $Q_{0,0}(t) = N$ and $q_{0,0}(t) = 1$ at any time t. Due to the term $q_{i,j}(t)$ being used very often when deriving the fluid limit, we will denote it with $q_{i,j}$ to make the expressions we derive more clear. But note that while we omit the notation of dependency of the time, $q_{i,j}$ does depend on the time.

Sometimes we will be looking at certain fractions of servers with for example exactly i tasks in the Xserver. They might come in naturally in our analysis, so we will introduce notation for those cases. The fraction of servers with exactly i tasks in the Xserver and at least j tasks in the server will be denoted by $q_{i=,j}(t)$. Similarly, we also introduce $q_{i,j=}(t)$ and $q_{i=,j=}(t)$.We can however quickly rewrite all three of them in terms of $q_{i,j}(t)$.

When we introduce the JSQ(d, δ) policy, we run into the following question. When we try to find the shortest queue length, do we only look at the tasks in the server or also at the tasks in the corresponding Xserver? Since we do not want to exclude a certain option, we introduce three policies which all fall under the overarching name of JSQ(d, δ).

- **JSQx(**d, δ**)**
  This policy only takes the number of tasks in the Xservers into account and then executes the JSQ(d) policy
- **JSQs(**d, δ**)**
  This policy only takes the number of tasks in the servers into account and then executes the JSQ(d) policy
- **JSQxs(**d, δ**)**
  This policy takes both the number of tasks in the Xservers and in the servers into account and then executes the JSQ(d) policy

The reason why we introduce JSQs(d, δ) and JSQx(d, δ) is that when δ is relatively large, in comparison with the mean service time at the server, there will be almost no tasks in the Xservers and analogously when δ is relatively small, there will be a lot of tasks in the Xservers. So we can make some assumptions to simplify the analysis of the system. Furthermore, it might be beneficial when trying to apply this in practice when it is impossible or undesirable to determine one of those two queue lengths.

For the JSQs(d, δ),JSQx(d, δ) and JSQxs(d, δ) policies we will derive an equation for the fluid limit. This will be done in the next section.

Finally we will briefly look at our second model. This model is only slightly different than our first model. The only difference is that we will take the rate at the Xserver to be $\frac{\delta}{d}$ instead of δ. Here we will use the JSQxs(d, δ) policy. But to avoid confusion, we will call it JSQd(d, δ), so that it is clear that we are looking at another model. Here the idea is to give a time penalty to the task when we increase d. This should reflect the idea of the communication burden that arises when we increase d. We choose to do this in the simple way by multiplying the average time in the Xserver with d. However, this could also be done in any other way, but that will be out of our scope. As we will see the choice for the time penalty also strongly depends on the value of δ, since the rate at the Xserver is $\frac{\delta}{d}$.

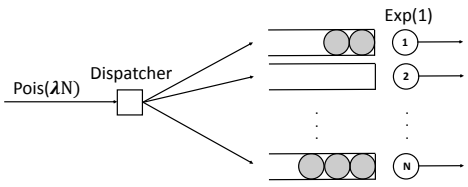

Figure 2.2: The model in [3]

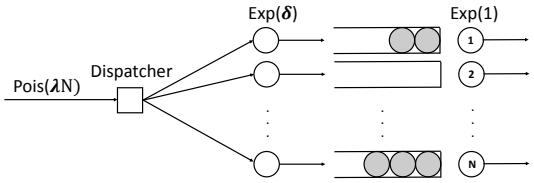

Figure 2.3: The model that we will be looking at

2.3 Assumptions

We introduce a few assumptions to make the model easier to work with.

- There is no loss of tasks at any point in time due to some kind of impatience or full buffers.
- It is possible to determine the number of tasks in the servers as well as to determine the number of tasks that already have been sent to the servers, but have not arrived yet, e.g. the number of tasks in the Xservers.
- In the JSQs(d, δ) policy the number of tasks in the Xservers is approximately 0, meaning that for determining the shortest queue length, we only have to look at the number of tasks in the server.
- In the JSQx(d, δ) policy the number of tasks in the servers is relatively small to the number of tasks in the Xservers, meaning that for determining the shortest queue length, it is sufficient to only look at the number of tasks in the Xservers.
- $\sum_{i=0}^{\infty}\sum_{j=0}^{\infty} q_{i,j}(0) < \infty$, meaning that the system does not contain infinitely many tasks at time $t = 0$. This is also mentioned in [1]. If we did not exclude this, we might have a system where every Xserver and server has infinitely many tasks. In this situation we are in equilibrium, since nothing would change $\frac{\mathrm{d}}{\mathrm{d}t}q_{i,j}(t)$ for i, j finite.

3 Analysis of the fluid limits

When we are analysing the JSQs(d, δ), JSQx(d, δ) and JSQxs(d, δ) policies, we are mostly interested in the fluid limit. As we will see, the fluid limits for the policies mentioned above, are very familiar. We want to give an expression that describes the change in the $q_{i,j}$ values for all three the policies. To find an expression for $\frac{\mathrm{d}}{\mathrm{d}t}q_{i,j}(t)$, we have to look at the events that yield a change in the $q_{i,j}$ values, with what probability this occurs and at what rate. This will be done in this section. This kind of reasoning is typically what is also used in other studies. [1], [2]
The reason why we want to determine the fluid limit is because we get rid of the randomness and hence the behaviour is deterministic, which makes analysing easier.

For all three the policies we can distinguish three different events which might yield a change in the $q_{i,j}$ values, namely an arrival at an Xserver, a departure at an Xserver (or equivalently, an arrival at a server) and a departure at a Server. If we look at a specific policy, we see that only the event of an arrival at an Xserver is affected by the policy and hence will be different. The other events will be equal for all three the policies.

We will first analyse the JSQs(d, δ) policy, where we will derive the expressions for the events of a departure at an Xserver and a departure at a server. Those expressions will then be the same for the JSQx(d, δ) and the JSQxs(d, δ) policies.

However, since the expression for the JSQs(d, δ) policy is already rather complicated, we will already state the fluid limit here, to give some insight in the expression and the factors that we will encounter when trying to obtain the fluid limit. Here the terms between square brackets correspond to the above-mentioned events respectively. Here we will assume that $i, j \neq 0$. The case where i or j equals zero will be treated later on.

$$\frac{\mathrm{d}}{\mathrm{d}t}q_{i,j}(t) = \left[\lambda \cdot \sum_{k=j}^{\infty}(\mathbb{1}_K \cdot \frac{q_{i-1=,k=}}{q_{0,k=}} \cdot (q^d_{0,k} - q^d_{0,k+1}))\right] + \left[\delta \cdot \textstyle\sum_{l=i+1}^{\infty}(l \cdot q_{l=,j-1=}) - i \cdot \delta \cdot q_{i=,j}\right] - \left[q_{i,j=}\right],$$

where K is the event that there are servers with exactly k tasks.

3.1 The JSQs(d, δ) policy

In order to derive the fluid limit, we have to use a limiting process to get from the finite system to a deterministic process given by the appropriate differential equations. The reasoning that our approach is alright is also explained in [1]. Here they use a theorem that is called Kurtz's theorem. This will allow us to take the limit for N to infinity. However we will not go in further detail on how to exactly implement this theorem.

3.1.1 Arrival at an Xserver

On average the arrival rate at each Xserver is λ. This is comparable to the JSQ(d) policy [3].

We see that $q_{i,j}$ increases if the task arrives at an Xserver with exactly $i-1$ tasks in the Xserver and at least j tasks in the queue for the normal server. Hence we want to determine the probability that the policy sends a task to an Xserver with exactly $i-1$ tasks and a server with at least j tasks. Since the probability is different for every j, we want to look at the probability for every j separately and take the sum of those probabilities. Hence

$$
\begin{aligned}
&\mathbb{P}(\text{policy sends task to an Xserver with exactly } i-1 \text{ tasks and at least } j \text{ tasks in server})\\
&= \sum_{k=j}^{\infty} \mathbb{P}(\text{policy sends task to an Xserver with exactly } i-1 \text{ tasks and exactly } k \text{ tasks in server})\\
&= \sum_{k=j}^{\infty} (\mathbb{P}(\text{policy send task to an Xserver with exactly } i-1 \text{ tasks in Xserver}\\
&\mid \text{ policy sends task to a server with exactly } k \text{ tasks in server}) \cdot\\
&\mathbb{P}(\text{policy sends task to a server with exactly } k \text{ tasks}))\\
&= \sum_{k=j}^{\infty} (\mathbb{P}(\text{exactly } i-1 \text{ tasks in Xserver} \mid \text{exactly } k \text{ tasks in server}) \cdot\\
&\mathbb{P}(\text{policy sends task to a server with exactly } k \text{ tasks}))\\
&= \sum_{k=j}^{\infty} (\tfrac{\mathbb{P}(\text{exactly } i-1 \text{ tasks in Xserver and exactly } k \text{ tasks in server})}{\mathbb{P}(\text{exactly } k \text{ tasks in server})} \cdot (q_{0,k}^d - q_{0,k+1}^d))\\
&= \sum_{k=j}^{\infty} (\tfrac{q_{i-1=,k=}}{q_{0,k=}} \cdot (q_{0,k}^d - q_{0,k+1}^d))\\
&= \sum_{k=j}^{\infty} (\tfrac{q_{i-1,k} - q_{i,k} - q_{i-1,k+1} + q_{i,k+1}}{q_{0,k} - q_{0,k+1}} \cdot (q_{0,k}^d - q_{0,k+1}^d))
\end{aligned}
$$

Here we use at the fourth equality the definition of a conditional probability and that the probability that the policy sends a task to a server with exactly k tasks equals $q_{0,k}^d - q_{0,k+1}^d$. This is the probability that all servers have at least k tasks in the server,

but not all of them have $k+1$ tasks in the server.
In the last step we express $\frac{q_{=i-1,=k}}{q_{0,=k}}$ in terms of $q_{i,j}$. To get the fraction of servers with exactly $i-1$ tasks at the Xserver and exactly k tasks at the server, we first have to subtract the fraction of servers with at least i tasks at the Xserver and at least k tasks at the servers from the fraction of servers with at least $i-1$ tasks at the Xserver and at least k tasks at the servers. This gives us the fraction of servers with exactly $i-1$ tasks at the Xservers and at least k tasks at the servers. So we also want to subtract the fraction of servers with at least $i-1$ tasks at the Xserver and at least $k+1$ tasks at the server to find the fraction of servers with exactly k tasks at the server. However now we subtracted the fraction of servers with at least i tasks at the Xserver and at least $k+1$ tasks at the server twice, hence we have to add this term to obtain our result. However we see that the terms this way seem to get pretty long. Therefore, we will use the shorter notation from now on. But keep in mind that the expression $q_{i=,j=}$ can easily be rewritten as explained here.

Note here the special case where $q_{0,k=}=0$ with $k \in \{j, j+1, ...\}$, and hence also $q_{i-1=,k=}=0$. In this case we would be dividing zero by zero. Hence we have to take a closer look at this case. When we do so, we are able to conclude that, since the fraction of servers with at least j tasks equals zero, it follows that the probability that the policy sends a task to an Xserver with exactly $i-1$ tasks and exactly k tasks in server equals zero as well. Hence this term will not be included in the fluid limit in this case. So in the equation we can use an indicator function to solve this problem. Let K be the event that $q_{0,k=} \neq 0$. This gives us the following term:
$\sum_{k=j}^{\infty}(\mathbb{1}_K \cdot \frac{q_{i-1=,k=}}{q_{0,k=}} \cdot (q_{0,k}^d - q_{0,k+1}^d))$

3.1.2 Departure at an Xserver

Here we see that $q_{i,j}$ can increase and can decrease.

1. $q_{i,j}$ decreases if the departure at an Xserver occurs when we have at least j tasks in the server and exactly i tasks in the Xserver. This happens at a rate of $i \cdot \delta$

   The fraction of servers that are in this state equals $q_{i=,j}$.

2. $q_{i,j}$ increases if the departure at an Xserver occurs when we have exactly $j-1$ tasks in the server and at least $i+1$ tasks in the Xserver.

   Like we just mentioned, if we had exactly $i+1$ tasks in the Xserver, then the rate would be $(i+1) \cdot \delta$. The corresponding fraction would in this case be $q_{i+1=,j-1=}$. However, we also have to take the Xservers into account with exactly $i+2$ tasks, and the Xservers with exactly $i+3$ tasks, and so on. If we would sum over them, we get the term for at least $i+1$ tasks in the Xserver.

Doing this gives us the following sum:
$\delta \cdot \sum_{l=i+1}^{\infty} l \cdot q_{l=,j-1=}$

3.1.3 Departure at a server

The departure rate at each server is 1.

We see that $q_{i,j}$ decreases if the departure occurs at a server with at least i tasks in the Xserver and exactly j tasks in the server. The fraction of servers that are in this state equals $q_{i,j=}$.

3.1.4 The fluid limit for JSQs(d, δ)

If we add all the terms derived in the previous subsections, we get the following equation for the fluid limit as already seen, assuming that $i, j \neq 0$:

$$\frac{\mathrm{d}}{\mathrm{d}t} q_{i,j}(t) = \left[\lambda \cdot \sum_{k=j}^{\infty} (\mathbb{1}_K \cdot \tfrac{q_{i-1=,k=}}{q_{0,k=}} \cdot (q_{0,k}^d - q_{0,k+1}^d))\right] + \left[\delta \cdot \textstyle\sum_{l=i+1}^{\infty} (l \cdot q_{l=,j-1=}) - i \cdot \delta \cdot q_{i=,j}\right] - \left[q_{i,j=}\right],$$

where K is the event that $q_{0,k=} \neq 0$.

Furthermore, we will have to look at the situations for $\frac{\mathrm{d}}{\mathrm{d}t} q_{0,j}(t)$ and $\frac{\mathrm{d}}{\mathrm{d}t} q_{i,0}(t)$, since we will note that some of the events can not occur in those situations.

Let us look at $\frac{\mathrm{d}}{\mathrm{d}t} q_{0,j}(t)$. Here we see that $q_{0,j}$ can not increase due to an arrival at an Xserver, since an Xserver can not have -1 tasks. Also we look at the term that corresponds to a departure at an Xserver. Here we see that when $i = 0$, this can not occur. However, this is already included in the equation, since we multiply by zero in that case.
This leads to the following equation:

$$\frac{\mathrm{d}}{\mathrm{d}t} q_{0,j}(t) = \left[\delta \cdot \textstyle\sum_{l=1}^{\infty} (l \cdot q_{l=,j-1=})\right] - \left[q_{0,j=}\right]$$

Let us look at $\frac{\mathrm{d}}{\mathrm{d}t} q_{i,0}(t)$. Here we see that $q_{i,0}$ can not decrease due to a departure at a server. Furthermore, $q_{i,0}$ can not increase by a departure at an Xserver, since this would require exactly -1 tasks at the server. This leads to the following equation:

$$\frac{\mathrm{d}}{\mathrm{d}t} q_{i,0}(t) = \left[\lambda \sum_{k=0}^{\infty} (\mathbb{1}_K \cdot \tfrac{q_{i-1=,k=}}{q_{0,k=}} \cdot (q_{0,k}^d - q_{0,k+1}^d))\right] - \left[i \cdot \delta \cdot q_{i=,0}\right],$$

where K is the event that $q_{0,k=} \neq 0$.

3.2 The JSQx(d, δ) policy

As mentioned at the beginning of section 3, the derivation of the fluid limit for the JSQx(d, δ) policy would be very similar to the derivation of the fluid limit for the JSQs(d, δ). The policy only changes the term that corresponds to the arrival at an Xserver.

So in this section we will only derive the term for the arrival at an Xserver. After that we will state the fluid limit for the JSQx(d, δ) policy.

3.2.1 Arrival at an Xserver

On average the arrival rate at each Xserver is λ, just as before.

Now we again want to determine the probability that the policy sends a task to an Xserver with exactly $i-1$ taks and a server with at least j tasks. But due to the different policy, we will find another expression for this. So with similar reasoning as before, we get

$$\begin{aligned}
&\mathbb{P}(\text{policy sends task to an Xserver with exactly } i-1 \text{ tasks and at least } j \text{ tasks in server}) \\
&= \mathbb{P}(\text{policy sends task to a server with at least } j \text{ tasks in server} \\
&\mid \text{policy sends task to a Xserver with exactly } i-1 \text{ tasks}) \cdot \\
&\mathbb{P}(\text{policy sends task to a Xserver with exactly } i-1 \text{ tasks})) \\
&= \frac{\mathbb{P}(\text{at least } j \text{ tasks in server and exactly } i-1 \text{ tasks in Xserver})}{\mathbb{P}(\text{exactly } i-1 \text{ tasks in Xserver})} \cdot (q^d_{i-1,0} - q^d_{i,0}) \\
&= \frac{q_{i-1=,j}}{q_{i-1=,0}} \cdot (q^d_{i-1,0} - q^d_{i,0})
\end{aligned}$$

Similar to what we saw in subsection 3.1.1, note the special case where $q_{=i-1,0} = 0$, and hence also $q_{=i-1,j} = 0$. In this case we would be dividing zero by zero. However in this case we are able to conclude that the the probability that the policy sends a task to a server with at least j tasks in the server given that the policy sends a task to a Xserver with exactly $i-1$ tasks equals zero. So similar to subsection 3.1.1, we can use an indicator function in the equation to solve this problem. Let K be the event that $q_{i-1=,0} \neq 0$. This gives us the following term:

$$\mathbb{1}_K \cdot \frac{q_{i-1=,j}}{q_{i-1=,0}} \cdot (q^d_{i-1,0} - q^d_{i,0})$$

3.2.2 The fluid limit for JSQx(d, δ)

Using sections 3.1 and 3.2.1, we get the following fluid limit for the JSQx(d, δ) policy for the case where $i, j \neq 0$:

$$\frac{\mathrm{d}}{\mathrm{d}t}q_{i,j}(t) = \left[\lambda \cdot \mathbb{1}_K \cdot \frac{q_{i-1=,j}}{q_{i-1=,0}} \cdot (q^d_{i-1,0} - q^d_{i,0})\right] + \left[\delta \cdot \sum_{l=i+1}^{\infty}(l \cdot q_{l=,j-1=}) - i \cdot \delta \cdot q_{i=,j}\right] - \left[q_{i,j=}\right],$$

where K is the event that $q_{i-1=,0} \neq 0$.

Using section 3.1.4 we get the following expressions for the fluid limit for $\frac{\mathrm{d}}{\mathrm{d}t}q_{0,j}(t)$ and $\frac{\mathrm{d}}{\mathrm{d}t}q_{i,0}(t)$

$$\frac{\mathrm{d}}{\mathrm{d}t}q_{0,j}(t) = \left[\delta \cdot \sum_{l=1}^{\infty} l \cdot q_{l=,j-1=})\right] - \left[q_{0,j=}\right]$$

$$\frac{\mathrm{d}}{\mathrm{d}t}q_{i,0}(t) = \left[\lambda \cdot (q^d_{i-1,0} - q^d_{i,0})\right] - \left[i \cdot \delta \cdot q_{i=,0}\right]$$

3.3 The JSQxs(d, δ) policy

As mentioned at the beginning of section 3.2 in this section we will only have to derive the term for the arrival at an Xserver. After that we will state the fluid limit for the JSQxs(d, δ) policy.

3.3.1 Arrival at an Xserver

On average the arrival rate at each Xserver is λ, just as before.

Now we once more want to determine the probability that the policy sends a task to an Xserver with exactly $i-1$ taks and a server with at least j tasks. But due to the different policy, we will find another expression for this. So with similar reasoning as before, we get

$$\begin{aligned}&\mathbb{P}(\text{policy sends task to an Xserver with exactly } i-1 \text{ tasks and at least } j \text{ tasks in server})\\ &= \sum_{l=j}^{\infty} \mathbb{P}(\text{policy sends task to an Xserver with exactly } i-1 \text{ tasks and exactly } l \text{ tasks in server})\end{aligned}$$

$$= \sum_{l=j}^{\infty} \left[\frac{q_{i-1=,l=}}{\sum_{k=1}^{i+l} q_{i+l-k=,k-1=}} \cdot \right.$$
$$\Big(\mathbb{P}(\text{policy picks a server and corresponding Xserver with a total of at least } l+i-1 \text{ tasks})^d -$$
$$\left. \mathbb{P}(\text{policy picks a server and corresponding Xserver with a total of at least } l+i \text{ tasks})^d \Big) \right]$$
$$= \sum_{l=j}^{\infty} \left[\frac{q_{i-1=,l=}}{\sum_{k=1}^{i+l} q_{i+l-k=,k-1=}} \cdot \right.$$
$$\Big((1-\mathbb{P}(\text{policy picks a server and corresponding Xserver with a total of less than } l+i-1 \text{ tasks}))^d -$$
$$\left. (1-\mathbb{P}(\text{policy picks a server and corresponding Xserver with a total of less than } l+i \text{ tasks}))^d \Big) \right]$$

Before continuing, let us explain what we wrote down in the last two steps.
In the second-last step the policy could only send a task to an Xserver with exactly $i-1$ and a server with exactly l tasks if the policy picks d servers for which the total number of tasks in the Xserver and the server equals at least $l+i-1$, but not all of the d servers have a total number of tasks of at least $l+i$. In this case the total number of tasks equals $l+i-1$. But if this occurs, there might be several servers for which it holds that the total number of tasks equals exactly $l+i-1$, for example if there are i tasks in the Xserver and $l-1$ tasks in the server (assuming that $l-1 \geq 0$). In this case the probability that the policy picks a Xserver with exactly $i-1$ tasks and a server with exactly l tasks is the fraction of the amount of those servers over the amount of servers for which it holds that the total number of tasks equals exactly $l+i-1$.
In the last step we simply rewrite the probability. This is done since we should keep in mind that the matrix $\mathbf{q}(t)$ is in theory infinite, but this will not be the case in the simulation. So when creating our simulation it turns out that we here might prefer to use this last form.

But if we look closely, we see that we do get in trouble if $\sum_{k=1}^{i+l} q_{i+l-k=,k-1=} = 0$, since we then seem to be dividing by zero. However if this is the case we will see that this means that also $q_{=i-1,=l} = 0$, meaning that the probability that the policy sends a task to an Xserver with exactly $i-1$ tasks and a server with exactly l tasks already equals zero, hence in this case we simply get a term that equals zero

Now we will come up with an expression for the probability that the policy picks a server and corresponding Xserver with a total of less than $l+i$ tasks. Here it will implicitly become clear why we did that last step. Since the policy picks a server at random, we just have to calculate which fraction of servers has a total number of tasks less than $l+i$. This leads to the following result:

$$\mathbb{P}(\text{policy picks a server and corresponding Xserver with a total of less than } l+i \text{ tasks})$$
$$= (q_{i+l-1=,0} - q_{i+l-1=,1}) + (q_{i+l-2=,0} - q_{i+l-2=,2}) + (q_{i+l-3=,0} - q_{i+l-3=,3}) + \cdots + (q_{0=,0} -$$

$q_{0=,i+l})$
$= \sum_{k=1}^{i+l} q_{i+l-k=,0} - \sum_{k=1}^{i+l} q_{i+l-k=,k}$
$= \sum_{k=1}^{i+l} (q_{i+l-k=,0} - q_{i+l-k=,k})$

Analogously we find:

$\mathbb{P}$(policy picks a server and corresponding Xserver with a total of less than $l+i-1$ tasks)
$= \sum_{k=1}^{i+l-1} (q_{i+l-1-k=,0} - q_{i+l-1-k=,k})$

Combining all this we get the following equation:

$\mathbb{P}$(policy sends task to an Xserver with exactly $i-1$ tasks and at least j tasks in server)
$= \sum_{l=j}^{\infty} \mathbb{1}_K \cdot \left[\frac{q_{i-1=,l=}}{\sum_{k=1}^{i+l} q_{i+l-k=,k-1=}} \cdot \left((1 - \sum_{k=1}^{i+l-1} q_{i+l-1-k=,0} - q_{i+l-1-k=,k})^d - (1 - \sum_{k=1}^{i+l} q_{i+l-k=,0} - q_{i+l-k=,k})^d \right) \right]$

Where K is the event that $q_{i-1=,l=} \neq 0$.

3.3.2 The fluid limit for JSQxs(d, δ)

Using the reasoning from sections 3.1 and 3.2.1, we get the following fluid limit for the JSQxs(d, δ) policy for the case where $i, j \neq 0$:

$\frac{\mathrm{d}}{\mathrm{d}t} q_{i,j}(t) = \left[\lambda \cdot \sum_{l=j}^{\infty} \mathbb{1}_K \cdot \left[\frac{q_{i-1=,l=}}{\sum_{k=1}^{i+l} q_{i+l-k=,k-1=}} \cdot \left((1 - \sum_{k=1}^{i+l-1} q_{i+l-1-k=,0} - q_{i+l-1-k=,k})^d - (1 - \sum_{k=1}^{i+l} q_{i+l-k=,0} - q_{i+l-k=,k})^d \right) \right] \right] + \left[\delta \cdot \sum_{l=i+1}^{\infty} (l \cdot q_{l=,j-1=}) - i \cdot \delta \cdot q_{i=,j} \right] - \left[q_{i,j=} \right],$

Where K is the event that $q_{i-1=,l=} \neq 0$.

Using the reasoning from section 3.1.4 we get the following expressions for the fluid limit for $\frac{\mathrm{d}}{\mathrm{d}t} q_{0,j}(t)$ and $\frac{\mathrm{d}}{\mathrm{d}t} q_{i,0}(t)$

$\frac{\mathrm{d}}{\mathrm{d}t} q_{0,j}(t) = \left[\delta \cdot \sum_{l=1}^{\infty} (l \cdot q_{l=,j-1=}) \right] - \left[q_{0,=j} \right]$

$$\frac{\mathrm{d}}{\mathrm{d}t}q_{i,0}(t) = \Big[\lambda \cdot \sum_{l=0}^{\infty} \mathbb{1}_K \cdot \Big[\frac{q_{=i-1,=l}}{\sum_{k=1}^{i+l} q_{i+l-k=,k-1=}} \cdot \Big((1 - \sum_{k=1}^{i+l-1} q_{i+l-1-k=,0} - q_{i+l-1-k=,k})^d - (1 - \sum_{k=1}^{i+l} q_{i+l-k=,0} - q_{i+l-k=,k})^d\Big)\Big]\Big] - \Big[i \cdot \delta \cdot q_{i=,0}\Big]$$

Where K is the event that $q_{i-1=,l=} \neq 0$.

But in the derived formula for $\frac{\mathrm{d}}{\mathrm{d}t}q_{i,0}(t)$, there is still one thing that we want to mention. In the case that $i = 1$ and $l = 0$, we see that we have to compute a sum from $k = 1$ to 0. We then conclude that this empty sum by definition equals zero. This is the case where we want to calculate the probability that the policy picks a server with in total less than zero tasks. Intuitively, we can see that indeed this probability should equal zero. Hence this is well included in the formula.

4 Simulation

We will mainly focus on the JSQs(d, δ) policy in the case where we only demonstrate a certain kind of behaviour. The results for JSQx(d, δ) are highly similar and are therefore partly omitted.

Further we want to note that we also tried to verify the fluid limit of the JSQxs(d, δ) policy, but it turned out that the simulation and the fluid limit do not match at all. We here saw in the fluid limit that $q_{0,1}$ was not converging to λ, the fraction of time that the server is busy. This gives us the idea that the simulation of the fluid limit itself might be incorrect. Due to a lack of time, we will not spent more time on this and therefore we will not verify whether this fluid limit seems to be correct.

4.1 Validation of our simulation

The results and graphs in this section are based on a simulation of a system with $N = 100$, an arrival rate of $\lambda = \frac{1}{2}$, a service rate of $\mu = 1$ and the number of choices $d = 2$.

We will show some graphs based on our simulation. To visualise the results of our simulation, we will first simulate JSQs(d, δ) and take $\delta = \frac{1}{100}$. This means that there will be a lot of tasks in the Xserver and that looking at the queue length at the server gives a bad impression of the sum of those two queues. Since it now feels like we do not use any useful information about the state of the system, it intuitively might be considered as the model in [3] with random arrivals at the servers.

In the next two graphs we will see the behaviour of $q_{0,j}$ and $q_{0,j=}$ respectively with $j \in 0, 1, \ldots, 9$. The graphs shown here are based on a single run of the simulation. Also the moments at which the values of $q_{0,j}$ are being determined is after a certain number of arrivals/departures. So we will see a lot of jumps in here.

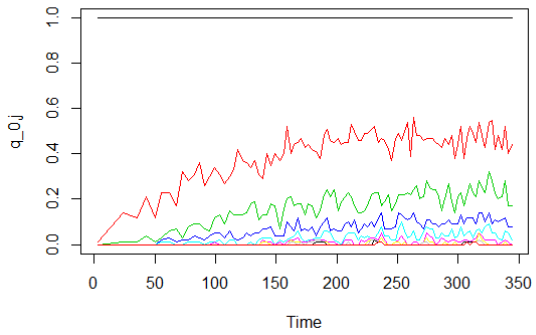

Figure 4.1: A single simulation run of $q_{0,j}$

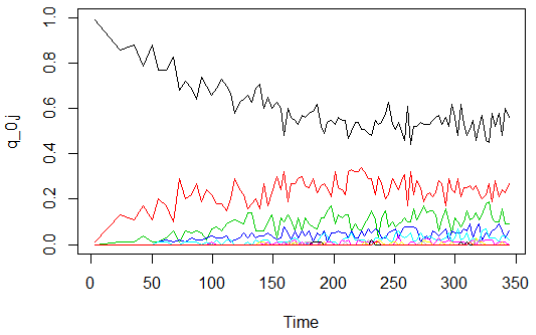

Figure 4.2: A single simulation run of $q_{0,j_=}$

We see that the lines are far from a smooth curve. However we can already recognise a certain trend in the lines.

Now we will look at the simulation results when we run it 100 times and take the average of those runs. This plot will be shown below.

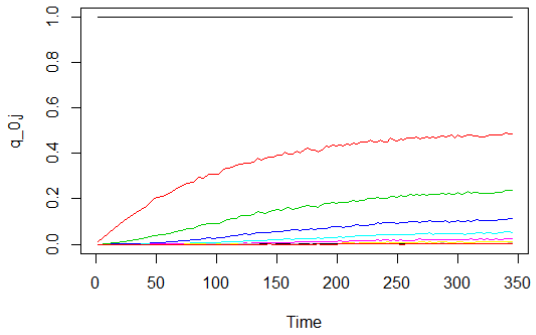

Figure 4.3: Average of $q_{0,j}$ over 100 simulation runs

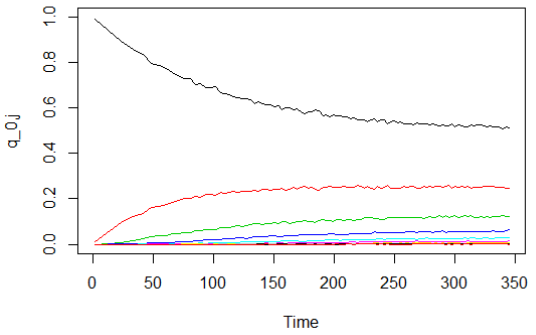

Figure 4.4: Average of $q_{0,j_=}$ over 100 simulation runs

Here we see that the lines are already fairly smooth compared to the previous plots.

Furthermore, we are interested in the behaviour for $t \to \infty$. So we made another plot where we increased the running time, so we are better able to show this behaviour. This plot will be shown below.

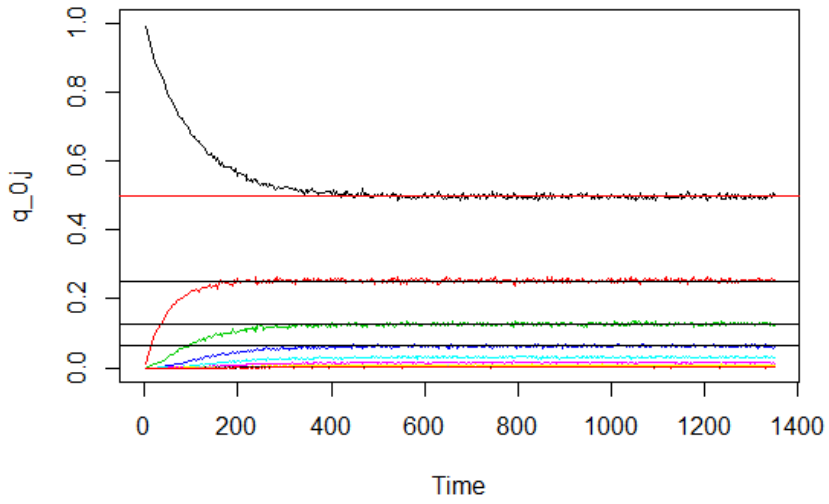

Figure 4.5: Average of $q_{0,j=}$ over 100 simulation runs over a large time

To test the correctness of our simulation, we can derive certain properties from our simulation and compare them with several basic scenarios and rules. One such rule is to take the occupancy rate into account. From queueing theory we know that the occupancy rate equals the fraction of time that the server is occupied. In our simulation the occupancy rate equals λ. We see that $q_{0,0=}$ tends to $\frac{1}{2}$, which should equal 1 minus the occupancy rate, λ, which is correct.

Take again a look at Figure 4.5. Note that we added some horizontal lines to illustrate the limit behaviour. The horizontal lines correspond to $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$ and $\frac{1}{16}$. Furthermore note that, as stated in [3], the queue length distribution for random assignment has λ^i as fixed point. In this simulation we took $\lambda = \frac{1}{2}$. This hints at the idea that for small δ the equilibrium values might be the same as those for the random assignment policy. We will come back to this later on in our report.

4.2 Simulation of our own model

Let us first see how the policies JSQs(d, δ), JSQx(d, δ) and JSQxs(d, δ) behave if we compare them with each other. Here it will of course be interesting to let the parameter δ vary, since this is where the essential difference lies between those policies. So we will fix all the other parameters and then compare the output of the simulation. We will simulate the mean time that a task spends in the queue at the server, the mean number of tasks that are in a server (so in the server itself and the corresponding

queue, but not in the corresponding Xserver), the mean time in the system (so from the moment of arrival in the Xserver until the moment it finishes its service in a server) and the mean number of tasks that are in the system as a whole.

The results are based on 10 simulation runs of a system with $N = 100$ servers, an arrival rate of $\lambda = 0.5$ per server, a service rate of $\mu = 1$ and the number of choices $d = 2$. To obtain the results, we let our simulation generate 100.000 tasks in advance which we do not keep track of just to give the system time to reach the equilibrium. Once this has been reached we generate another 100.000 tasks that the simulation does keep track of. The obtained results are shown in the table below:

policy	δ	mean time at queue	mean number of tasks in a server	mean time in system	mean number of tasks in the system
JSQs(d, δ)	$\frac{1}{100}$	0.9682	1.0059	101.9685	5087.9992
	$\frac{1}{10}$	0.8102	0.8867	11.8019	587.9922
	1	0.4941	0.7257	2.4947	122.1441
	10	0.3095	0.6661	1.4098	71.8512
	100	0.2737	0.6333	1.2831	63.8787
JSQxs(d, δ)	$\frac{1}{100}$	0.9272	0.9628	101.8507	5085.357
	$\frac{1}{10}$	0.6855	0.8459	11.6992	586.5648
	1	0.3943	0.6935	2.3979	118.5244
	10	0.2876	0.6662	1.3888	71.3669
	100	0.2741	0.6389	1.2841	64.4401
JSQx(d, δ)	$\frac{1}{100}$	0.9575	0.9596	101.9825	5102.5641
	$\frac{1}{10}$	0.8037	0.8751	11.8077	591.1482
	1	0.7073	0.8351	2.7096	135.4705
	10	0.9146	0.973	2.0136	102.4988
	100	0.993	1.0074	2.0025	101.2347

Table 4.1: The influence of δ on the performance of the three different policies.

One more sanity check to see whether our simulation works properly, is to check whether Little's law holds. For convenience we recall Little's law.

$$\mathbb{E}(L) = \bar{\lambda}\mathbb{E}(S)$$

Where $\mathbb{E}(L)$ denotes the mean number of tasks in the system, $\mathbb{E}(S)$ denotes the mean sojourn time and $\bar{\lambda}$ denotes the average number of tasks entering the system per unit time. So we can compute with this law the mean number of tasks in the system and compare this with the value we obtained from the simulation. This is shown in the table below. In the last column of the table we will also note the fraction of the two, which should be close to 1. Note that in our system we have $\bar{\lambda} = \lambda N$.

policy	δ	$\bar{\lambda}\mathbb{E}(S)$	$\mathbb{E}(L)$	$\frac{\bar{\lambda}\mathbb{E}(S)}{\mathbb{E}(L)}$
JSQs(d,δ)	$\frac{1}{100}$	5098.425	5087.9992	1.002
	$\frac{1}{10}$	590.095	587.9922	1.0036
	1	124.735	122.1441	1.0212
	10	70.49	71.8512	0.9811
	100	64.155	63.8787	1.0043
JSQxs(d,δ)	$\frac{1}{100}$	5092.535	5085.357	1.0014
	$\frac{1}{10}$	584.96	586.5648	0.9973
	1	119.895	118.5244	1.0116
	10	69.44	71.3669	0.973
	100	64.205	64.4401	0.9964
JSQx(d,δ)	$\frac{1}{100}$	5099.125	5102.5641	0.9993
	$\frac{1}{10}$	590.385	591.1482	0.9987
	1	135.48	135.4705	1.0001
	10	100.68	102.4988	0.9823
	100	100.125	101.2347	0.989

Table 4.2: Validation of Little's law for the obtained results.

So we see that the fractions are all close to the value 1 as they should be. Due to the simulation, there might be of course a small deviation from this value, so it is fine if those values are not exactly equal to 1.

Now let us look at the same kind of graphs for the JSQs(d,δ) policy as in the previous subsection, however this time we take $\delta = 100$, so the Xservers are relatively empty. We will show two graphs to see the behaviour of $q_{0,j}$ and $q_{0,j=}$ with $j \in 0,1,2,3,4$. The graphs are based on 100 runs.

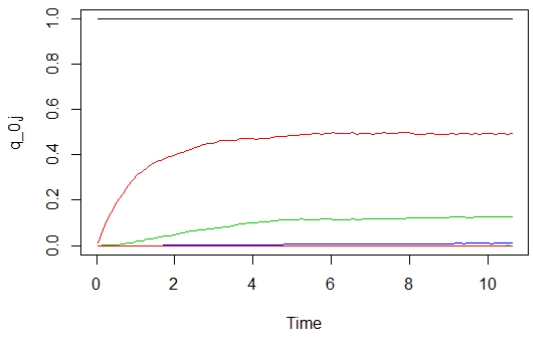

Figure 4.6: Average of $q_{0,j}$ over 100 simulation runs

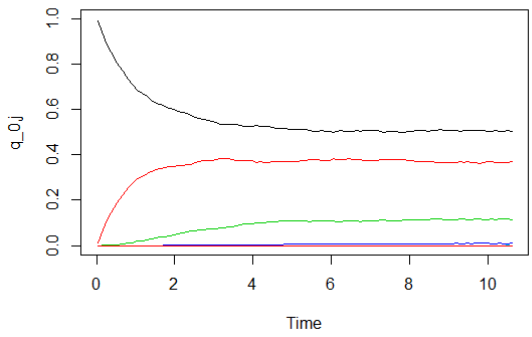

Figure 4.7: Average of $q_{0,j=}$ over 100 simulation runs

A sharp-eyed reader might notice at this point that the time on the x-axis in figures 4.6 and 4.7 is much smaller than in figures 4.3, 4.4 and 4.5. To understand this behaviour, one should keep in mind that we changed the value for δ. Now we have $\delta = 100$, meaning that the mean time in the Xserver is only $\frac{1}{100}$, whereas before we had $\delta = \frac{1}{100}$, meaning that the mean time in the Xserver is 100. This causes the difference for the state of the systems as function of the time.

Until this point, we used in every simulation run a system that started completely empty, but one might wonder what would happen if we took a different initial situation. Therefore, we also run a simulation where we changed the initial state such that every queue at a server contained 5 tasks. The influence of the different initial state is visible in the graphs below.

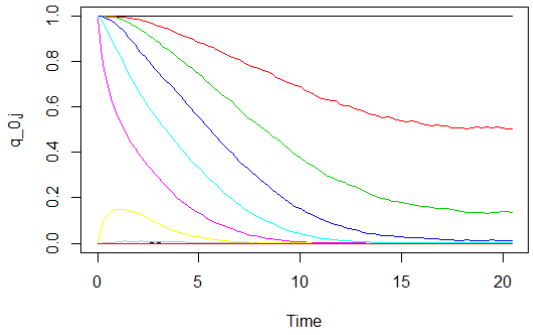

Figure 4.8: Average of $q_{0,j}$ over 100 simulation runs where $q_{0,5_=}(0) = 1$

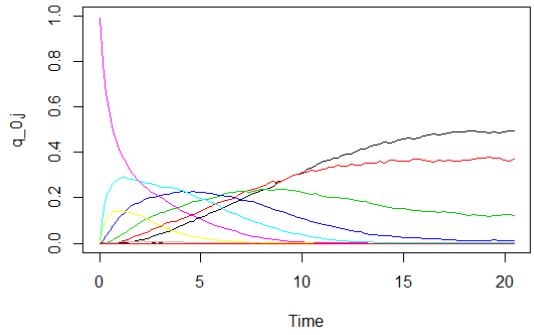

Figure 4.9: Average of $q_{0,j_=}$ over 100 simulation runs where $q_{0,5_=}(0) = 1$

We see that the lines seem to converge to the same values as in Figures 4.6 and 4.7. We also see the behaviour that over time the maximum attained values of $q_{0,4_=}$, $q_{0,3_=}$ and $q_{0,2_=}$ appear in time in this order, which we can interpret as the queue sizes decreasing. Something else that we notice is the behaviour of the yellow line, representing $q_{0,6}$. This line starts increasing shortly in the beginning, due to the JSQs(d, δ) policy choosing $2\ (= d)$ servers with both queue length 5.

Remark that our system is an extension of the JSQ(d) policy. The extension was to add the Xservers. However we usually take δ to be large, which causes the mean time in the Xserver to be small. So if we set the time in the Xserver to be zero, we would expect to recover the JSQ(d) policy. This would mean that we have to take the limit of δ to infinity. We will investigate this scenario later on in our report using simulation.

4.3 Fluid limit

Until so far, we have been looking at the results of the simulation of the system. But in this subsection we will look at the result if we try to plot $q_{0,j}$, while using the fluid limit that we derived.

Since the fluid limit is a rather complicated differential equation, we will try to do this numerically. We will use the idea of forward Euler to achieve this. This means that if we look at the fluid limit, we consider that $\frac{\mathrm{d}}{\mathrm{d}t}q_{i,j}(t)$ can be interpreted as the limit of Δt to zero of $\frac{\Delta q_{i,j}(t)}{\Delta t}$, where Δt denotes a small time interval and $\Delta q_{i,j}(t)$ denotes the change of $q_{i,j}$ during this small time interval. Then by multiplying both sides of the fluid limit with Δt, we get an expression for $\Delta q_{i,j}(t)$.
Hence if we know $q_{i,j}$ at time t, then we can estimate the increase $\Delta q_{i,j}(t)$. So we can get the approximation of $q_{i,j}$ at time $t + \Delta t$, which is given by $q_{i,j}(t + \Delta t) \approx q_{i,j}(t) + \Delta q_{i,j}(t)$.
If we now assume that we start with a certain initial state, for example all Xservers and servers empty, then we can calculate the matrices $\mathbf{q}(n \cdot \Delta t)$ one by one, where $n \in \mathbb{N}$. Those matrices represent the system state at time $n \cdot \Delta t$. From all those matrices, we can derive the values of $q_{0,j}$ over time.

We will use the approach above for the JSQs(d, δ) and the JSQx(d, δ) policies. We take an arrival rate of $\lambda = \frac{1}{2}$ per server, a service rate of $\mu = 1$, the number of choices $d = 2$ and a time step size $\Delta t = \frac{1}{100}$.

First we will look at the JSQs(d, δ) policy. Here we will take a service rate of $\delta = 10$ at the Xservers. This gives us the following plot:

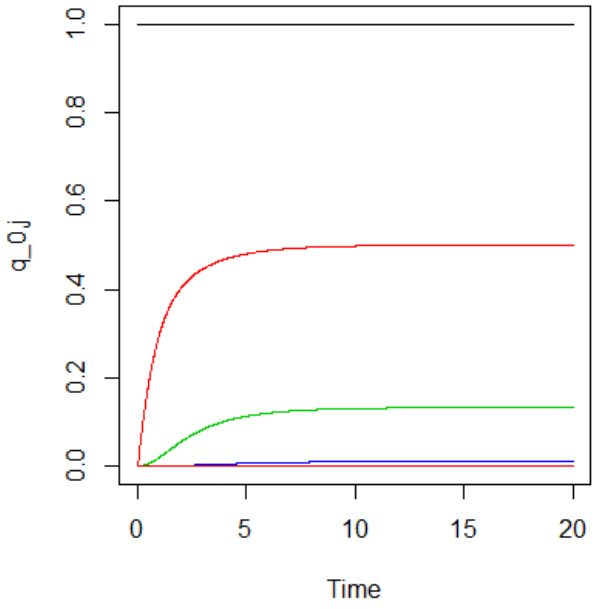

Figure 4.10: Simulation of the fluid limit of JSQs(d, δ), with $\lambda = \frac{1}{2}$

Here we see that all lines are monotone increasing in contrast to the plots for the simulation. Since there is no randomness in the fluid limit, this is also to be expected for the case where the initial state is empty.
Furthermore, we also see the behaviour that the convergence values of $q_{0,j}$ decrease extremely fast towards 0 as j decreases. This is comparable to the behaviour of the JSQ(d) policy.

Next we will look at the JSQx(d, δ) policy. Here we will take a service rate of $\delta = \frac{1}{10}$ at the Xservers. This gives us the following plot:

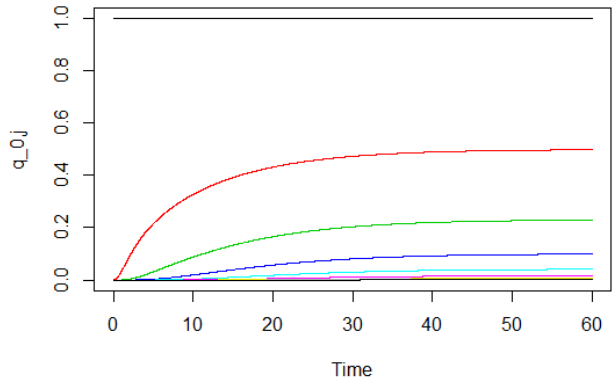

Figure 4.11: Simulation of the fluid limit of JSQx(d, δ), with $\lambda = \frac{1}{2}$

Here we see that the lines for $q_{0,j}$ with $j \in 2, 3, 4, ..$ converge to a higher value than in figure 4.10.

4.4 Comparison of simulation and fluid limit

Now that we have illustrated both the results from our simulation and the results from our fluid limit, we are of course interested whether both results match, since this would provide support for our derived fluid limit. Let us first look at the JSQs(d, δ) policy.
We will show three plots for different values of the arrival rate λ where we show in each plot both the simulation and the fluid limit. We will do this for $\lambda = \frac{1}{5}, \lambda = \frac{1}{2}$ and $\lambda = \frac{9}{10}$. Furhter we will take $\delta = 10$ and $d = 2$.

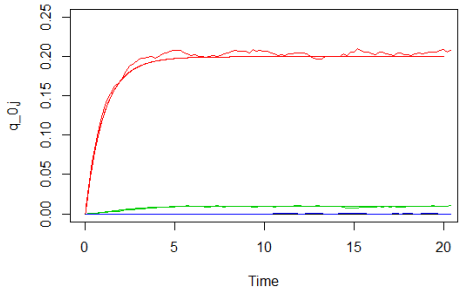

Figure 4.12: The results of the simulation and of the fluid limit of JSQs(d, δ) for arrival rate $\lambda = \frac{1}{5}$

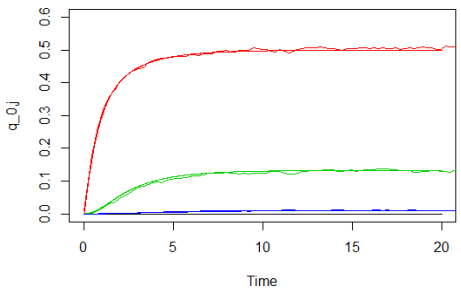

Figure 4.13: The results of the simulation and of the fluid limit of JSQs(d, δ) for arrival rate $\lambda = \frac{1}{2}$

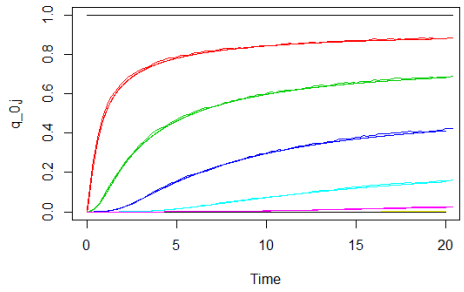

Figure 4.14: The results of the simulation and of the fluid limit of JSQs(d, δ) for arrival rate $\lambda = \frac{9}{10}$

We see that the simulation results seem to be very close to the values that we obtained from the fluid limit.

Next, we will also look at the JSQx(d, δ) policy and include three figures. Again one with $\lambda = \frac{1}{5}$, one with $\lambda = \frac{1}{2}$ and one with $\lambda = \frac{9}{10}$.

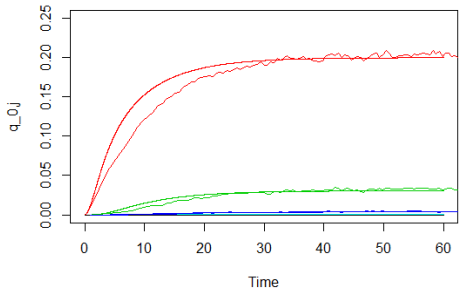

Figure 4.15: The results of the simulation and of the fluid limit of JSQx(d, δ) for arrival rate $\lambda = \frac{1}{5}$

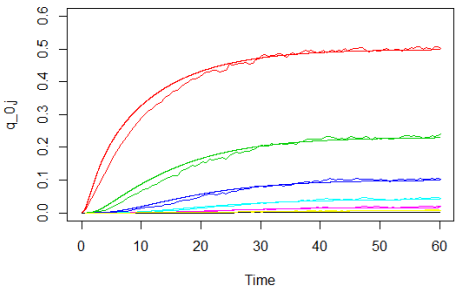

Figure 4.16: The results of the simulation and of the fluid limit of JSQx(d, δ) for arrival rate $\lambda = \frac{1}{2}$

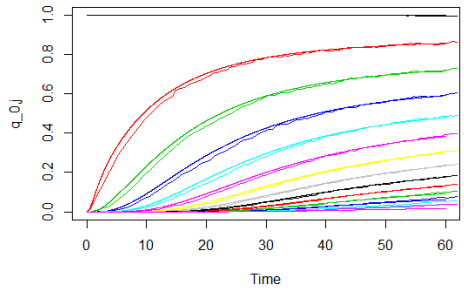

Figure 4.17: The results of the simulation and of the fluid limit of JSQx(d, δ) for arrival rate $\lambda = \frac{9}{10}$

Here we see that our simulation results look to be somehow delayed a bit before they obtain the same values as the fluid limit. This behaviour seems to be less significant when λ increases, but is still suspicious and undesirable. We would like to understand what causes this behaviour. A possible explanation might be that this is somehow due to the waiting time in the Xserver. The delay caused by this waiting time might not be well included in the behaviour of the fluid limit. However we see that the convergence values seem to correspond nevertheless.

4.5 Behaviour of JSQs(δ, d) for large δ

Something else that we wonder about, is the behaviour of the system when δ becomes large. In this case the time that a task is in the Xserver will become relatively small to the time that a task is in the server and its corresponding queue. Therefore if we would take $\delta \to \infty$, we expect to see that the Xservers will be empty almost all the time and that the tasks are almost immediately forwarded to the servers. So in this case, we expect that the system starts to look like the system under the JSQ(d) policy.

We would be interested to look at this case in an analytical way. But since we have no analytical expression for the stationary points of the JSQs(d, δ) policy, we could only take a look at the fluid limit. However, this will cause the mathematical problem of $\infty - \infty$ to arise, due to the δ being present in two different terms. Since we are not able to find a way to deal with this, we will take a look at this behaviour using simulation. In the two figures below we plotted the $q_{0,j}$ values over time, where we used the black lines for the JSQ(d) policy and the coloured lines for the JSQs(d, δ) policy. In the first figure we did this for $\lambda = \frac{1}{2}$ and in the second figure for $\lambda = \frac{9}{10}$. Furthermore, we used $\delta = 100$ in this simulation.

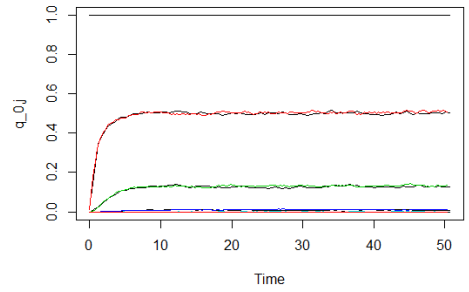

Figure 4.18: Comparison between the JSQ(d) and JSQs(d, δ) policy for arrival rate $\lambda = \frac{1}{2}$

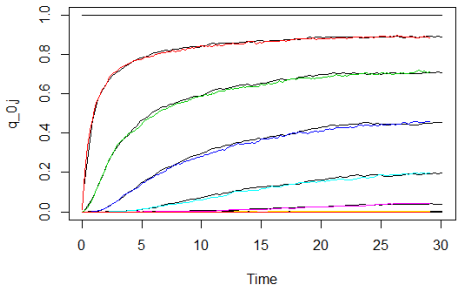

Figure 4.19: Comparison between the JSQ(d) and JSQs(d, δ) policy for arrival rate $\lambda = \frac{9}{10}$

We see that in both the figures the coloured lines match pretty closely with the corresponding black lines.

4.6 Behaviour of JSQs(δ, d) for small δ

In the previous subsection, we considered what would happen for large δ. In this subsection we will consider what will happen in the opposite case, where delta is small. In this case we expect to see that the Xservers will contain many tasks. Therefore taking into account how many tasks the servers contain, becomes a bad indication which server will have the shortest waiting time. Therefore one might get the idea that the system starts to act more as a system where the policy at the dispatcher acts randomly, meaning it just assigns an incoming task to a server at random. To see whether this idea is correct, we used our simulation again to compare the random assignment to the JSQs(d, δ) policy. Like in subsection 4.5, in the figure below we plotted the $q_{0,j}$ values over time, where we used the black lines for the random assignment and the coloured lines for the JSQs(d, δ) policy. Furthermore, we used $\lambda = \frac{1}{2}$ and $\delta = \frac{1}{100}$ in this simulation.

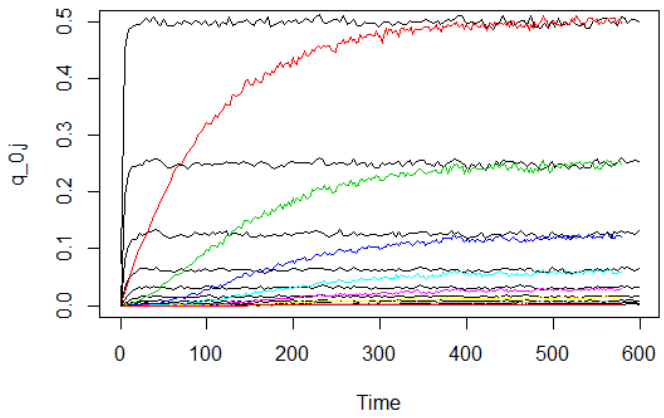

Figure 4.20: Comparison between the random assignment and JSQs(d, δ) policy for arrival rate $\lambda = \frac{1}{2}$

Due to the behaviour of the random assignment policy for large λ, we decided to not include the figure for $\lambda = \frac{9}{10}$, because this turned out to look pretty messy. As we can see in figure 4.20, the lines are clearly acting differently when the time is smaller than 400. We here have to take into account that in the random assignment policy, there are no Xservers included, but in the JSQs(d, δ) policy there are. Note furthermore that we took $\delta = \frac{1}{100}$, meaning that the time in the Xservers is exponentially distributed with mean 100. So this will cause a certain delay in the arrival of tasks at the servers. However, the value of highest interest here is the fixed point of the fluid limit. As we can see, both the policies seem to converge to the same value. However it should also be noted that, compared to other simulations, the JSQs(d, δ) policy with $\delta = \frac{1}{100}$ converge rather slowly

4.7 JSQd(d, δ) policy

As already mentioned in subsection 2.2, the difference between JSQd(d, δ) and JSQxs(d, δ) is only that now the time in the Xserver is exponentially distributed with rate $\frac{\delta}{d}$ instead of rate δ. Hence, whereas in the JSQxs(d, δ) it would only be beneficial to increase the value of d (in the case where we ignore the communication burden), it now might be in our disadvantage to just keep increasing the value of d.

Now let us intuitively explain what is going on. In the scenario we created now, the value of d influences both the average waiting time at the Xserver and the average waiting time at the server. The average waiting time at the server as a function of d is a function that will be monotone decreasing and converging to some value. As mentioned in [1], the decrease for small d values is really significant. On the other hand,

the average waiting time at the Xserver as a function of d is simply a linear function, since this is just the value $\frac{d}{\delta}$. Therefore if we would look at the average total time that a task spends in the system, which is the sum of the waiting time at the Xserver, the waiting time at the server and the service time, than we expect this to be some convex function. In this case we would be interested in the d value that would minimise the average total time.

However before we can do this, we should note that a function of the average total time also depends on the values of δ, λ and even N. We will not take the value of N into account, since we expect this only to stretch out the function in the horizontal direction. But the values of λ and δ do affect this function a lot. Since λ is a value between 0 and 1, we can try out what the effect of λ is. But for δ, this is somewhat different. In the previous two subsections we looked at small and large values for δ. If we take $\delta = \frac{1}{10}$, then we see that the average time at an Xserver is $10 * d$. Meaning that if $d = 1$, we already see that the average time at an Xserver is 10. This would not make sense if we remark that the average service time at a server is only 1. On the other hand we see that if $\delta = 1000$, then increasing d by 1 would give a time penalty of only $\frac{1}{1000}$, which also does not make a lot of sense. So here we see that the choice for the time penalty strongly depends on the value on δ. However the real values of the time penalty and δ are somewhat more to be determined by a real life situation.

As mentioned in the previous paragraph, we will have to take several values for λ and δ into account. Therefore we will use our simulation to see the behaviour. We will first take $\delta = 100$ and compare what the influence of λ is by taking $\lambda = \frac{1}{5}$, $\lambda = \frac{2}{5}$, $\lambda = \frac{3}{5}$ and $\lambda = \frac{4}{5}$.

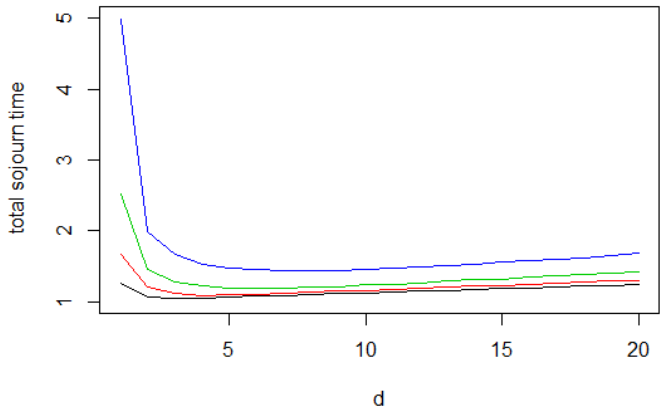

Figure 4.21: The total time in the system when varying d and λ

Here the blue line corresponds to $\lambda = \frac{4}{5}$, the green line to $\lambda = \frac{3}{5}$, the red line to $\lambda = \frac{2}{5}$ and the black line to $\lambda = \frac{1}{5}$. We obviously see that the total time spent in the system decreases as λ decreases. But more interestingly we also can see that the d values that minimise the function change. We will list them also clearly in the table below.

λ	d
0.2	3
0.4	4
0.6	6
0.8	8

Table 4.3: Some values of λ with the corresponding values of d that minimise the total time in the system

Now we will also illustrate the effect of δ as we already described before. We will take $\lambda = \frac{4}{5}$. The values of δ that we will take are $\delta = 1$, $\delta = 10$, $\delta = 100$ and $\delta = 1000$. The reason why we take $\lambda = \frac{4}{5}$ is because it turns out that this creates a rather full system and therefore it will be interesting to consider to increase d even when receiving a time penalty

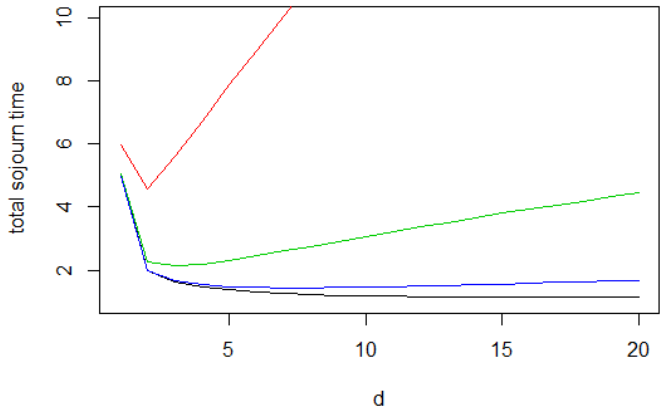

Figure 4.22: The total time in the system when varying d and δ

Here the red line corresponds to $\delta = 1$, the green line to $\delta = 10$, the blue line to $\delta = 100$ and the black line to $\delta = 1000$. We again obviously see that the total time spent in the system decreases as δ increases. But here we can clearly see the link between d and δ. For small values of δ, we see that the influence of the waiting time in the Xserver is dominant an for large values of δ we see that the influence of the waiting time in the server is dominant. For clarity we will again list the d values that minimise the function.

δ	d
1	2
10	3
100	8
1000	18

Table 4.4: Some values of δ with the corresponding values of d that minimise the total time in the system

5 Conclusion

In this report we introduced the JSQ(d, δ) policy and analysed it. The JSQ(d, δ) policy is an extension of the JSQ(d) policy. In our system tasks arrive in the system at a dispatcher according to a Poisson process. There the JSQ(d, δ) policy is executed and tasks are send to one of the N servers. However before they arrive at the queue, an exponentially distributed time passes. We say that those tasks are in the corresponding Xserver of the server. Since there are now two kinds of servers, we introduced the JSQx(d, δ), JSQs(d, δ) and JSQxs(d, δ) policies, respectively taking into account the amount of tasks in the Xserver, the amount of tasks in the server and the amount in both. For the JSQs(d, δ), JSQx(d, δ) and JSQxs(d, δ) policies we analytically derived an expression for the fluid limit.

In order to verify correctness of the fluid limits, we also run a simulation. Comparing the fluid limit of JSQs(d, δ) to our simulation seems to support the correctness of our fluid limit pretty well as can be seen in figures 4.15, 4.16 and 4.17. However for the fluid limit of JSQx(d, δ) we see that the fluid limit in the beginning is a little bit off of the simulation. However the converging value seems to be correct. Our expectation is that this is due to the element of the introduced Xservers and the extra waiting time that they induce. However figuring out this slight mismatch is further left open as future work. We also tried to verify the correctness of the fluid limit for the JSQxs(d, δ) policy, but the results here were that the simulation and the fluid limit did not match. However the fluid limit behaved as if it might be the case that the simulation of the fluid limit itself might contain some error. Due to a lack of time we did not find the solution to this and therefore we can not get a conclusion about the correctness of the derived fluid limit of the JSQxsd, δ) policy.

Besides this we also compared the JSQx(d, δ), JSQs(d, δ) and JSQxs(d, δ) policies in table 4.1. From this table we conclude that JSQs(d, δ) and JSQxs(d, δ) behave very similarly when δ is sufficient larger than 1. Already for $\delta = 10$ we see that the results for JSQs(d, δ) are close to those of JSQxs(d, δ). If δ becomes smaller, then JSQs(d, δ) seems to be slightly less efficient than JSQss(d, δ). For JSQx(d, δ) and JSQxs(d, δ) we see that they behave similarly for $\delta = \frac{1}{100}$ or for δ even smaller. But when δ becomes larger, we see that JSQx(d, δ) performs way worse than JSQxs(d, δ).

Therefore, we conclude that JSQxs(d, δ) is the best policy of the three which we treated. However JSQs(d, δ) is the second best and is only slightly worse then JSQxs(d, δ). And last JSQx(d, δ) performs the worst. So if it is possible to use JSQxs(d, δ), we recommend this policy. However due to practical issues it might be difficult to determine the number of tasks in the Xserver. In this case we recommend the JSQs(d, δ) policy.

Furthermore we analysed some of the properties of the JSQs(d, δ) policy using our simulation. We saw that if we start with a system that is not empty (but neither contains infinitely many tasks), that it would still behave fine and converge to the same value as when the system is empty. Also we looked at the behaviour of the system for small and large values of δ. Here we saw that if δ became small, the equilibrium values converge to the equilibrium values of the random assignment policy. If δ became large, we saw that the equilibrium values converge to the equilibrium values of the JSQ(d) policy.

Finally we also took a brief look at the JSQd(d, δ) policy. Here we saw that giving a time penalty at the Xserver for increasing d leads to the situation where there will be an optimal d value which leads to the minimum average time that a tasks spends in a system. However this value strongly depends on the type of time penalty that we give, the value of δ and the value of λ.

Bibliography

[1] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[2] Michael Mitzenmacher. Analyzing distributed Join-Idle-Queue: A fluid limit approach. *54th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2016*, pages 312–318, 2017.

[3] Mark van der Boor, Sem C. Borst, Johan S. H. van Leeuwaarden, and Debankur Mukherjee. Scalable Load Balancing in Networked Systems: Universality Properties and Stochastic Coupling Methods. 2017.