SSA Bonus - Group 25

Pascal Anema i6221933, Nicolas Perez Zambrano i6224201, Simona Vychytilova i6223546

May 24, 2021

1 Current Queuing Method

The current method used by the company distributes products into the queues based on their length. Since the servers equipped with GPUs have two queues, the total queue length is the sum of the two separate queues. When a product comes into the system it is added to the shortest queue that can handle the product. The GPU servers prioritise the GPU queue over the regular queue. This layout is visualised below.

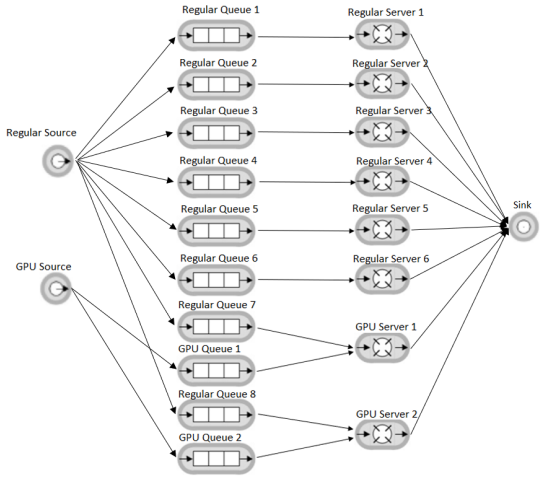

Figure 1: Current Queuing Technique used

2 Optimised Queuing Methods

An idea to optimise the current queuing system the company has in place was proposed. In this system, only 2 queues exist, one for the regular jobs and one for the GPU jobs. Each server then takes jobs from the queue based on which type it is.

The issue with this is that if the GPU queue is empty the GPU machines are idle and therefore there are various moments where a GPU machine could help out, but doesn't.

To find a way of improving on this issue the method decided upon was that when a regular job is created it can enter the GPU queue only if all the regular servers are busy and a GPU server is

idle. Therefore it will not spend any time in the queue and go straight to the GPU server. These two queuing systems are visualised below.

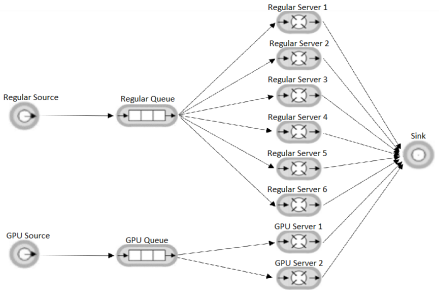

Figure 2: First Optimised Queuing Method

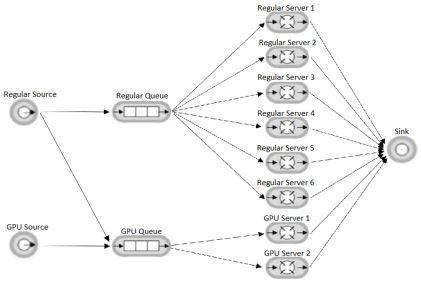

Figure 3: Second Optimised Queuing Method

3 Results

3.1 Long run test

In order to get an idea of how the systems work when running for a longer period of time, the three systems were ran for the equivalent of 10 years. A scatter plot containing all the delays was produced to allow data to be visualised. The average delay for each 15 minute interval was also displayed to see how the delay varies throughout one day. The results of this are seen below:

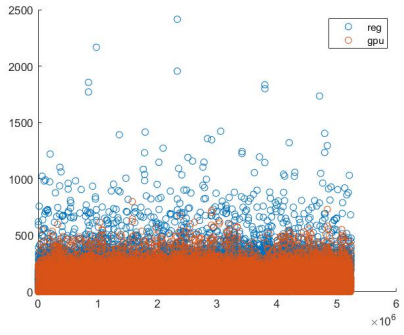

Figure 4: Current Queuing Method Scatter

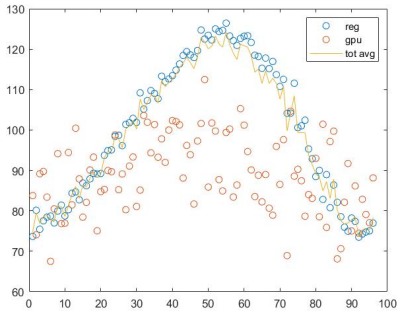

Figure 5: Current Queuing Method Average Delays (in minutes) in a Day (points given for each quarter-hour of the day)

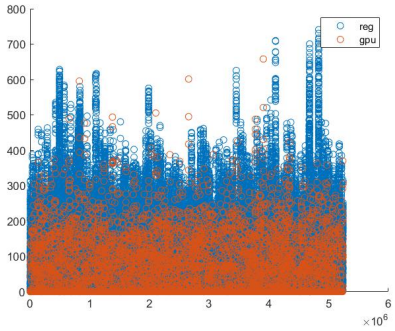

Figure 6: First Optimised Queuing Method Scatter

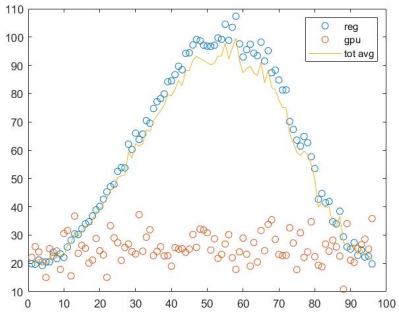

Figure 7: First Optimised Queuing Method Average Delays in a Day

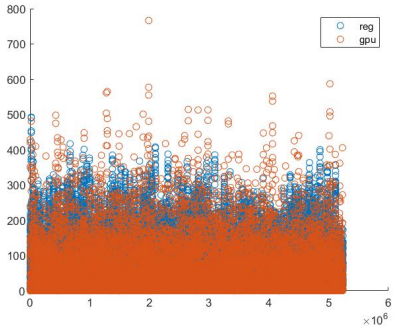

Figure 8: Second Optimised Queuing Method Scatter

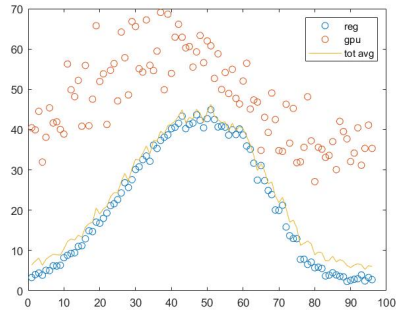

Figure 9: Second Optimised Queuing Method Average Delays in a Day

In all three systems the regular jobs follow a sinusoidal pattern similar to that of the arrival rates. This is as expected because a higher arrival rate will result in a larger queue. The delays of the GPU jobs have a much higher variance and no common pattern can be seen between the different systems.

3.2 Individual System Performance

Each system was also simulated 1000 times for 4 weeks and for each simulation the mean and 90% percentile of delay for the regular, GPU and overall jobs. The bounds were calculated, and the results for each metric are in the following table:

	Total bounds	GPU Bounds	Regular Bounds
Default	(97.2133, 97.6354)	(75.4110, 76.5620)	(98.9647, 99.3886)
First	(65.0163, 66.7034)	(16.6490, 17.4972)	(68.9554, 70.7747)
Second	(31.8504, 32.6724)	(48.8123, 50.1669)	(30.0850, 30.9404)

Table 1: Mean Delay Bounds

	Percentile total	Percentile GPU	Percentile regular
Default	(187.4350, 188.1604)	(212.5251, 214.9117)	(185.7963, 186.5171)
First	(197.6764, 202.7901)	(61.2383, 65.9588)	(203.6429, 209.0236)
Second	(115.1233, 117.9511)	(151.7192, 155.6495)	(110.4658, 113.5097)

Table 2: 90th Percentile Delay Bounds

The team interpreted the constraint that obtaining the minimum value with 99% performance guarantee was equal to getting the upper bounds on how high the delay could be. This was done with a similar method to getting the regular bounds, but in this case it was calculated with a t-value of 98%. The decision for 98% was because we were calculating only the upper bounds, and by having the alpha value be equal to 2% in this case it made guaranteed the one-tailed test to have a 99% guarantee. This was done by using the mean of each of the statistics, as well as the 90th percentile. Results are as follows:

	Total	GPU	Regular	Percentile total	Percentile GPU	Percentile regular
Default	97.6879	76.7054	99.5089	188.3663	215.5889	186.6068
First	67.1821	17.7379	71.2909	204.2412	67.2983	210.5505
Second	32.9056	50.5513	31.1831	118.7535	156.7648	114.3735

Table 3: Upper Bounds

For the 95% the confidence intervals and the minimum value, that can be assured with a confidence of 99% are listed below

3.3 Comparison between systems

For this section, a paired t-test was applied to every combination of systems. These tests were done for the total mean, mean for GPU jobs, as well as mean for the normal jobs. Each individual test was done with a confidence interval of 0.5% probability, in order to deal with the Bonferroni inequality. No tests were done for the 90th percentile nor for the 99th percentile guarantees. Results are as follows:

	Total values	GPU Bounds	CPU Bounds
default - first	(30.2047, 32.9243)	(57.8049, 60.0220)	(27.8476, 30.7756)
default - second	(64.4348, 65.8911)	(25.1105, 27.8834)	(67.9108, 69.4172)
first - second	(32.1071, 35.0898)	(-33.6683, -31.1648)	(37.7538, 40.9509)

Table 4: Total Delay

4 Discussion

The most important point of discussion comes when comparing the different systems with each other, because that's what provides us the most concrete reference on whether on system is truly better than the other. In this case, because we're assuming that delay is bad, that means that if in the comparison the bounds are positive then the second system mentioned is better, and if it's the negative then the first system is better.

Based off this, it is easy to see that the default system is the worst one in the three metrics measured, at least compared to the two other methods. The interesting comparison is in the first method vs the second method, as it's the only one where one model doesn't dominate. What you can see there is that the first method is better for GPU jobs, but for normal jobs the second one is better. Additionally, because there are more normal jobs than GPU jobs, this also means that the second method performs better overall.

This occurs because if a GPU job arrives in the first system it will be process independently to the regular jobs. While in the second method the regular jobs can sometimes take advantage of idle GPU machines. In the second system as the arrival rate of the regular jobs increases it becomes more likely that all the regular servers are busy and they are put into the GPU queue. This therefore increases the waiting time of the GPU jobs which is why the delay for GPU also approximates a sinusoid.

The decision on which version to use depends then on what the company has to focus on. If the company only cares about having the best performance overall, using the second method would be better. Actually, considering how many more regular jobs there are, it might be worthwhile to devote more resources to them, for example by making one of the GPU machines always take in regular jobs. Now, if it turns out that GPU jobs are in some way more urgent, then in that case the first method would be better.

While the variance wasn't calculated for each run, we can get some intuition on how it worked on each system by comparing the mean vs the 90th percentile. Based off this, it seems that the one advantage the default server seems to have is that it has comparably less variance. This isn't worth that much when you take into account that even this "smaller" variance still leads to worse times on the 90th percentile compared to the second version, and only slightly better compared to the first version. This happens because the improvements to the core system are that big, that the extra blow in variance is of little effect.

A Appendix

This section serves as extra explanation for how we generate our non-stationary exponential random variables. As we used a different method from the thinning algorithm described in the lectures.

A.1 Non-stationary Exponential Distribution

For the non-stationary exponential distribution, we can compute the average rate of arrival over a given time interval $[a, b)$ for our non-stationary exponential distribution T as

$$\lambda_{[a,b)} = \frac{1}{b-a}\int_a^b \lambda(t)dt$$

If we now take our interval to be $[\tau, \tau+\alpha)$, we can use the current time τ and inter-arrival time α as parameters into the PDF and CDF of our distribution:

$$\lambda_{\tau,\alpha} := \frac{1}{\alpha}\int_\tau^{\tau+\alpha} \lambda(t)dt \tag{1}$$

$$P\left(T \leq \alpha\right) = 1 - \exp\left(-\lambda_{\tau,\alpha} \cdot \alpha\right) \tag{2}$$

$$P\left(T \leq \alpha\right) = 1 - \exp\left(\frac{1}{\alpha}\int_\tau^{\tau+\alpha} \lambda(t)dt \cdot \alpha\right) \tag{3}$$

$$P\left(T \leq \alpha\right) = 1 - \exp\left(\int_\tau^{\tau+\alpha} \lambda(t)dt\right) \tag{4}$$

$$\tag{5}$$

For further equations, we will use $\Lambda\left(\tau, \alpha\right)$ to denote the expected number of arrivals over the interval $[\tau, \tau+\alpha)$:

$$\Lambda\left(\tau, \alpha\right) := \int_\tau^{\tau+\alpha} \lambda(t)dt$$

Thus we get for the definition of the CDF:

$$F_T\left(\tau, \alpha\right) = 1 - \exp\left(\Lambda\left(\tau, \alpha\right)\right)$$

We can, using the definition of the PDF of the exponential distribution, also define the PDF as follows:

$$f_T\left(\tau, t\right) = \frac{d}{d\alpha}F_T\left(\tau, t\right) \tag{6}$$

$$f_T\left(\tau, \alpha\right) = \lambda_{\tau,\alpha}\exp\left(-\lambda_{\tau,\alpha} \cdot \alpha\right) \tag{7}$$

$$f_T\left(\tau, \alpha\right) = \frac{\Lambda\left(\tau, \alpha\right)}{\alpha}\exp\left(-\Lambda\left(\tau, \alpha\right)\right) \tag{8}$$

In order to generate random variables according to the non-stationary distribution with our given λ function, we can use several approaches. Since the CDF and PDF resulting from solving the integral $\Lambda\left(\tau, \alpha\right)$ for $\tau, \alpha \in [0, \infty)$ are both continuous and neatly defined, we decided to use the inverse transformation method. Our given λ function is defined with the mean- and amplitude rate of arrival M and A and the period P as follows:

$$\lambda(t) = M + A \cdot \sin\left(\frac{2\pi}{P}t\right)$$

We need to compute the integral $\Lambda(\tau, \alpha) = \int_{\tau}^{\tau+\alpha} \lambda(t)dt$ to get the exact CDF and PDF of our distribution. This is done below:

$$\Lambda(\tau, \alpha) = \int_{\tau}^{\tau+\alpha} \lambda(t)dt \tag{9}$$

$$= \int_{\tau}^{\tau+\alpha} M + A \cdot \sin\left(\frac{2\pi}{P}t\right) dt \tag{10}$$

$$= M \cdot t\Big|_{\tau}^{\tau+\alpha} + A \cdot \left(-\frac{P}{2\pi}\cos(\frac{2\pi}{P}t)\Big|_{\tau}^{\tau+\alpha}\right) \tag{11}$$

$$= M\alpha + A\frac{P}{2\pi} \cdot \left(\cos\left(\frac{2\pi}{P}\tau\right) - \cos\left(\frac{2\pi}{P}(\tau+\alpha)\right)\right) \tag{12}$$

To simplify the expression a bit, we use the following symbols:

$$\omega := \frac{P}{2\pi} \tag{13}$$

$$z(\tau) := \omega \cdot \cos\left(\frac{\tau}{\omega}\right) \tag{14}$$

This gives for the solution to the integral $\Lambda(\tau, \alpha)$:

$$\Lambda(\tau, \alpha) = M\alpha + A \cdot z(\tau) - A\omega \cdot \cos\left(\frac{\tau+\alpha}{\omega}\right)$$

With this solution, we can compute the PDF and CDF exactly if the current time τ and inter-arrival time α are known. But in order to generate inter-arrival times α instead, we need the inverse CDF of this distribution. The inverse CDF is, even though it is a non-decreasing function over the interval $[0, 1)$, not properly defined. To circumvent this issue, we can use a numerical method to approach the value of the inverse CDF for use in the inverse transformation method.

A.2 Newton's method

Let ${F_T}^{-1}$ be the inverse CDF such that ${F_T}^{-1}[F_T(t)] = t$ for $t \in [0, \infty)$.
We use Newton's method for N iterations to compute the Nth estimate of t such that $F_T(t) = p$. We compute this with $t_0 = \frac{1}{M}$ and the following procedure:

$$t_{i+1} = t_i - \frac{F_T(\tau, t_i) - p}{f_T(\tau, t_i)}$$

This gives a prediction t_N such that $F_T(\tau, t_N) \approx p$ and ${F_T}^{-1}(\tau, p) \approx t_N$. The error of this prediction can be computed by $E = |F_T(\tau, t_N) - p|$. To balance simulation performance and accuracy, we decided to use 12 iterations of Newton's method for our inverse CDF. This results on average in an error of $2.3 \cdot 10^{-10}$ and gives a maximum error of $3.8 \cdot 10^{-6}$ over 100000 calculations.

A.3 Generating random variates

Using Newton's method as a predictor for ${F_T}^{-1}(\tau, p)$, we generate random variables according to the non-stationary exponential distribution using the inverse transformation method:

1. retrieve the current time τ
2. Generate $p \sim U[0, 1)$
3. Approximate ${F_T}^{-1}(\tau, p) \approx t_N$ using 12 iterations of Newton's method
4. return t_{12} as inter-arrival time

This is the procedure we use to generate the non-stationary exponential random variables. Graphs showing the generated random variables over time do indeed seem to follow the sinusoidal relationship that was requested. This can be seen below in the experimentally retrieved approximate arrival rates for regular and GPU jobs. Here, the regular jobs are distributed according to the non-stationary exponential distribution with $\lambda(t) = 2.0 + 0.8 \cdot \sin(\frac{2\pi}{24}t)$, where t is given in hours.

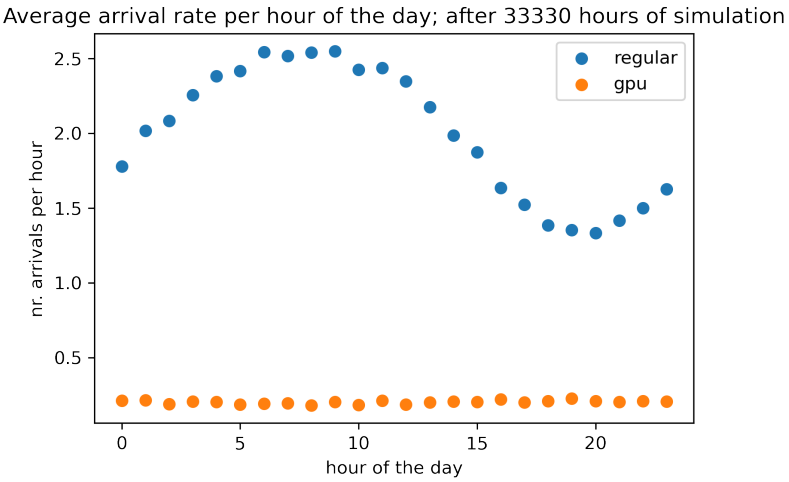

Figure 10: Arrival Rate of regular and GPU jobs