Server Load Balancing Modeled with Differential Equations

Krish Renjen, Aadith Muthukumar

April 2026

1 Introduction

Load Balancing is the process of distributing incoming tasks across a number of systems in order to avoid overload and maximize throughput. Tasks can range from different types of requests, jobs, and traffic. In this report, we look into the idea of using differential equations in order to model how loads evolve over time, and how useful it can be to model a dynamic system.

1.1 Overview

In this report we will look at two servers that have incoming jobs that they need to process.

For the sake of simplicity, we will assume that jobs are modeled continuously, rather than discretely.

In addition, we will denote the load (the amount of jobs) of a server i as $x_i(t)$. While not explicitly stated, assume that when $x_i(t) = f(x_1, x_2, t)$ it is meant as $x_i(t) = \max(0, f(x_1, x_2, t))$ as a negative load is impossible. It will be omitted from the models below for simplicity.

We will look into the following models:

1. Decoupled Load Balancing Systems + Analysis
2. Simple Coupled Load Balancing Systems + Analysis (2 servers)
3. Complex Load Balancing Systems + Analysis (2+ servers)
4. Deeper dive into Complex Load Balancing System (general equilibrium + stability)
5. Current Load Balancing Methods (Round Robin, Queue Based, Hash Based)
6. How do these balancing methods compare with each other

1.2 Simulation

You can follow along with the simulation at `https://loadbalancing.streamlit.app/`. More information is located in the appendix. Each model will also contain the appropriate settings for the simulation.

2 Decoupled Load Balancing Systems

Before we can analyze load balancing systems, we must first establish a basic model we can build off of. This basic model involves two servers whose load capacity evolves **independently**. It is important to note that this is unrealistic for real balancing, since there is no coordination involved and overloaded servers don't offload work. Nevertheless, the qualitative analysis of such differential systems can inform us on what improvement we can make in order to make a more optimal load balancing system.

2.1 Basic Model

2.1.1 Model

Suppose that we have two servers. We will denote their loads as $x_1(t)$ and $x_2(t)$.

Let:

- p_i = the processing rate for server i (how many jobs per unit time the server can complete)
- λ_i = the number of incoming jobs per unit time for server i

Then,

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \lambda_1 - p_1 \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \lambda_2 - p_2 \end{cases}$$

2.1.2 Simulation Settings

To view this in a simulation, select:

- **Arrival Algorithm:** Basic Model (Constant Rate)
- **Number of Servers:** 2
- **Coupling Gain** (k)**:** 0

2.1.3 Algebraic Analysis

This is a very simple differential equation, that is we can solve it as

$$\begin{cases} x_1 = (\lambda_1 - p_1)t + J_1 \\ x_2 = (\lambda_2 - p_2)t + J_2 \end{cases}$$

where J_i is the initial amount of jobs at time $t = 0$ for server i.

This gives us 3 cases,

$$\lim_{t\to\infty} x_i(t) = \begin{cases} \infty & \lambda_i > p_i \\ J_i & \lambda_i = p_i \\ 0 & \lambda_i < p_i \end{cases}$$

Note: the only time we are able to reach a stable state is if $\lambda_i \leq p_i$.

2.1.4 Graph Analysis

Below are the respective graphs when $\lambda_i > p_i$ and $\lambda_i > p_i$. As you can see, they both respective approach 0 and ∞.

When $\lambda_i = p_i$, the system does not change and just stays at the original amount of jobs. This graph is just a horizontal straight line, so we didn't include it in the report.

2.2 Proportional Processing Rates

As a server begins to have a higher load, they allocate more resources (threads on a CPU) to process tasks. That is, if you only have one task, you will not

need to "use" your entire compute power.

2.2.1 Model

Let μ_i = represent the proportion of processing rate.

That is if $\mu_i = 0.5$ and $x_i(t) = 10$ then at that moment in time, it will process 5 jobs.

Our model is as follows:

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \lambda_1 - \mu_1 x_1 \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \lambda_2 - \mu_2 x_2 \end{cases}$$

2.2.2 Simulation Settings

To view this in a simulation, select:

- **Arrival Algorithm:** Decoupled Static
- **Processing Mode:** Proportional
- **Number of Servers:** 2
- **Coupling Gain** (k)**:** 0

2.2.3 Analysis

We solve a typical equation of the form:

$$\frac{dx}{dt} = \lambda - \mu x.$$

Rewriting:

$$\frac{dx}{dt} + \mu x = \lambda.$$

The integrating factor is:

$$e^{\mu t}.$$

Multiplying both sides:

$$\frac{d}{dt}\left(xe^{\mu t}\right) = \lambda e^{\mu t}.$$

Integrating:

$$xe^{\mu t} = \frac{\lambda}{\mu} e^{\mu t} + C.$$

Thus,

$$x(t) = \frac{\lambda}{\mu} + Ce^{-\mu t}.$$

Applying this to each variable:

$$x_1(t) = \frac{\lambda_1}{\mu_1} + C_1 e^{-\mu_1 t},$$
$$x_2(t) = \frac{\lambda_2}{\mu_2} + C_2 e^{-\mu_2 t}.$$

Set derivatives equal to zero:

$$0 = \lambda_1 - \mu_1 x_1 \quad \Rightarrow \quad x_1^* = \frac{\lambda_1}{\mu_1},$$
$$0 = \lambda_2 - \mu_2 x_2 \quad \Rightarrow \quad x_2^* = \frac{\lambda_2}{\mu_2}.$$

Thus, the equilibrium point is:

$$(x_1^*, x_2^*) = \left(\frac{\lambda_1}{\mu_1}, \frac{\lambda_2}{\mu_2} \right).$$

The solutions contain exponential terms:

$$e^{-\mu_1 t}, \quad e^{-\mu_2 t}.$$

If $\mu_1 > 0$ and $\mu_2 > 0$, then:

$$\lim_{t \to \infty} e^{-\mu_i t} = 0,$$

so the solutions approach the equilibrium point. Therefore, the system is asymptotically stable.

Taking limits:

$$\lim_{t \to \infty} x_1(t) = \frac{\lambda_1}{\mu_1},$$
$$\lim_{t \to \infty} x_2(t) = \frac{\lambda_2}{\mu_2}.$$

The system models a processing system where:

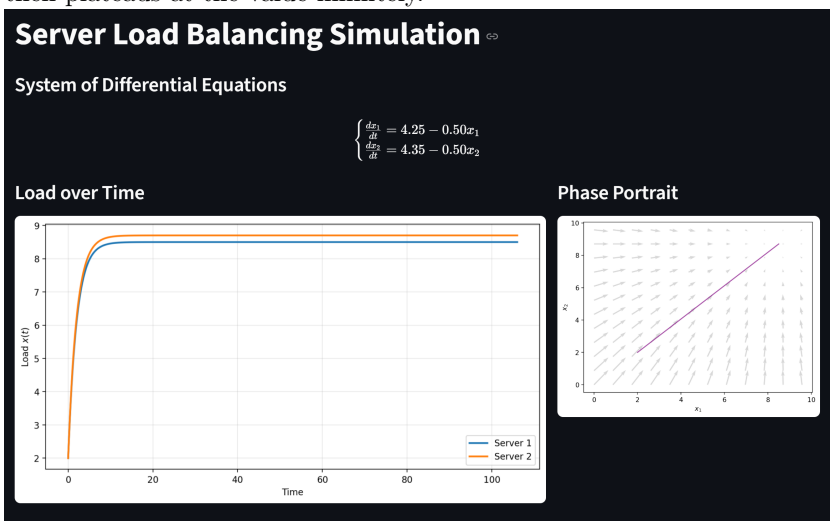
- λ_i is the arrival rate,
- $\mu_i x_i$ is the processing rate.

At equilibrium, the arrival rate equals the processing rate:

$$\lambda_i = \mu_i x_i^*.$$

2.2.4 Graph Analysis

As seen in the graph depicted below, the system approaches exactly $\mu_i x_i^*$. It then plateaus at the value infinitely.



2.3 Limited (Satured) Processing Rates

The main issue with the previous model is it assumes that processing power can scale infinitely in one server. This is obviously not the case, so we must correct for this.

2.3.1 Model

Let p_i = represent the **max** processing rate. This is the max amount of jobs per unit time server i can process.

We define $S(x_i) = \frac{x_i}{x_i+1}$. Note that $S(x_i)$ is bounded between 0 and 1 for all positive real numbers and that $S(0) = 0$, and $\lim_{x_i \to \infty} S(x_i) = 1$. This is our new processing multiplier.

Our model is as follows:

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \lambda_1 - p_1 S(x_1) \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \lambda_2 - p_2 S(x_2) \end{cases}$$

We will leave this in terms of $S(x_i)$ instead of simplifying so the model is as straightforward as possible.

2.3.2 Simulation Settings

To view this in a simulation, select:

- **Arrival Algorithm:** Decoupled Static
- **Processing Mode:** Saturated
- **Number of Servers:** 2
- **Coupling Gain** (k)**:** 0

2.3.3 Analysis

Equilibria occur when $\frac{dx_i}{dt} = 0$. Thus:

$$0 = \lambda_1 - p_1 S(x_1),$$
$$0 = \lambda_2 - p_2 S(x_2).$$

Solving:

$$S(x_i^*) = \frac{\lambda_i}{p_i}.$$

Since $S(x) \in [0, 1)$, equilibria exist only if:

$$0 \leq \frac{\lambda_i}{p_i} < 1 \quad \iff \quad \lambda_i < p_i.$$

If $\lambda_i < p_i$, then solving

$$\frac{x_i^*}{x_i^* + 1} = \frac{\lambda_i}{p_i}$$

gives:

$$x_i^* = \frac{\lambda_i}{p_i - \lambda_i}.$$

Thus, the equilibrium point is:

$$(x_1^*, x_2^*) = \left(\frac{\lambda_1}{p_1 - \lambda_1}, \frac{\lambda_2}{p_2 - \lambda_2} \right), \quad \text{provided } \lambda_i < p_i.$$

We can see that the existence of equilibria depends on the following:

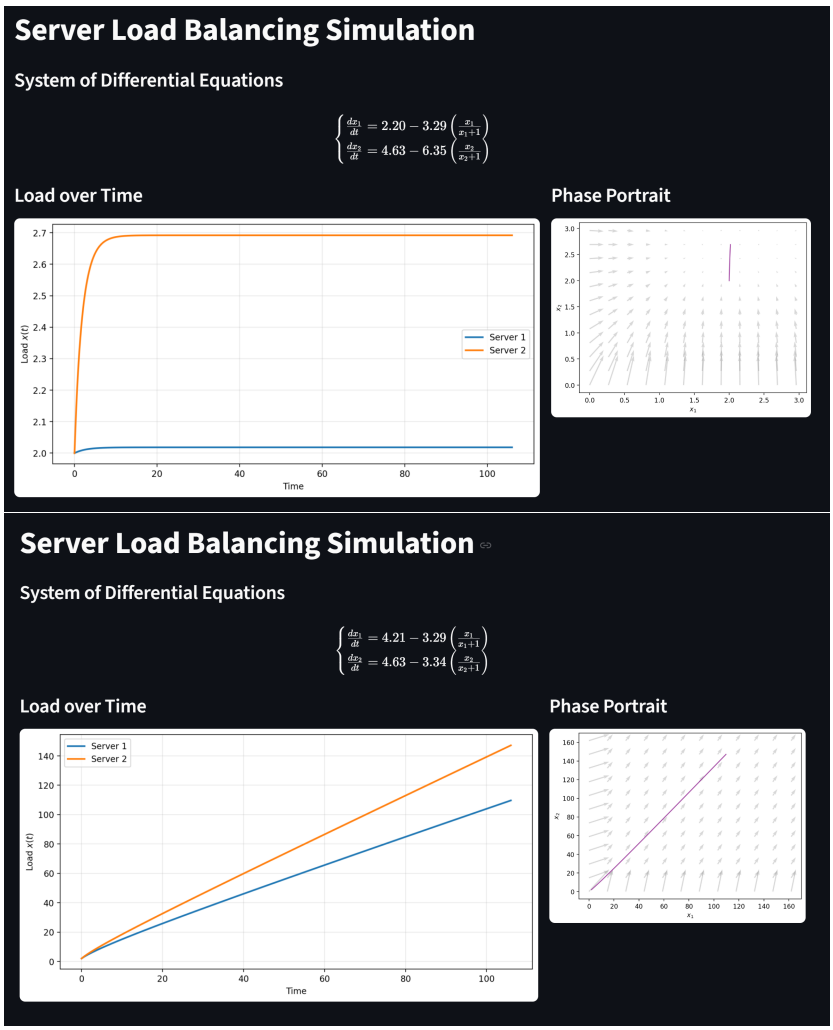
- If $\lambda_i < p_i$: a unique positive equilibrium exists.
- If $\lambda_i = p_i$: no finite equilibrium exists and $x_i(t) \to \infty$.
- If $\lambda_i > p_i$: the system cannot keep up with arrivals and $x_i(t) \to \infty$.

Thus, the effective processing rate is bounded above by p_i. Unlike the linear model, increasing x_i does not increase processing indefinitely; instead, it approaches a maximum capacity.

- If arrivals are below capacity $(\lambda_i < p_i)$, the system stabilizes.
- If arrivals meet or exceed capacity $(\lambda_i \geq p_i)$, the queue grows without bound.

2.3.4 Graph Analysis

Notice in the graphs depicted below that there is indeed a unique positive equilibrium (calculate above) when $\lambda_i < p_i$. In the case that $\lambda_i >= p_i$, however, there is no finite equilibrium and the system approaches ∞



3 Coupled Load Balancing System

Suppose that we have two servers. One may have a very high load, while the other is sitting idle. In this case, we can offload some of the tasks onto the other server. This is the basic idea of coupling. This section will look at a direct coupling.

3.1 Basic Model

We define k as the coupling factor. A high k indicates we will try to even out jobs as fast as possible.

3.1.1 Model

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \lambda_1 - \mu_1 x_1 + k(x_2 - x_1) \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \lambda_2 - \mu_2 x_2 + k(x_1 - x_2) \end{cases}$$

3.1.2 Simulation Settings

To view this in a simulation, select:

- **Arrival Algorithm:** Decoupled Static
- **Processing Mode:** Proportional or Saturated (this exact model uses proportional), however saturated is more realistic
- **Number of Servers:** 2
- **Coupling Gain** (k)**:** > 0

3.1.3 Analysis

At equilibrium, $\frac{dx_1}{dt} = \frac{dx_2}{dt} = 0$. This gives

$$\begin{aligned} (\mu_1 + k)x_1 - kx_2 &= \lambda_1, \\ -kx_1 + (\mu_2 + k)x_2 &= \lambda_2. \end{aligned}$$

This is a system of two linear equations in two unknowns, so there exists a unique equilibrium (x_1^*, x_2^*).

The system can be written as

$$\mathbf{x}' = A\mathbf{x} + \mathbf{b},$$

where

$$A = \begin{pmatrix} -(\mu_1 + k) & k \\ k & -(\mu_2 + k) \end{pmatrix}.$$

To determine stability, we examine the trace and determinant of A:

$$\mathrm{tr}(A) = -(\mu_1 + \mu_2 + 2k) < 0,$$

$$\det(A) = (\mu_1 + k)(\mu_2 + k) - k^2 > 0.$$

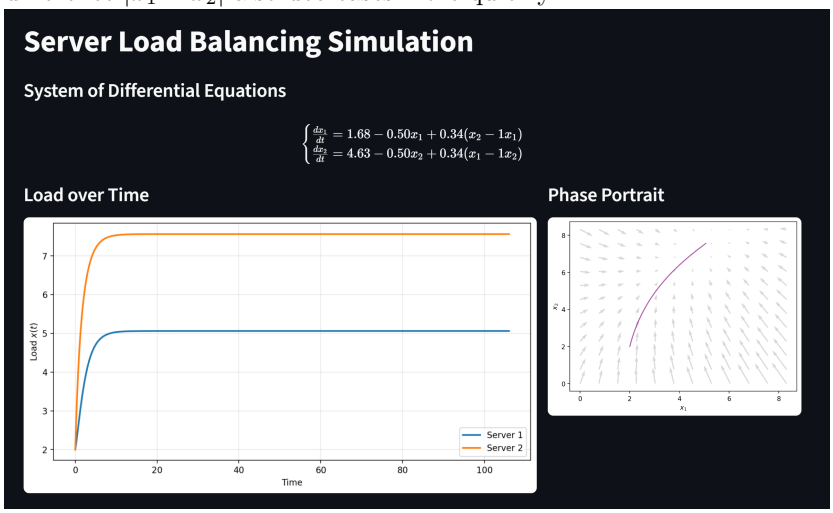
Since $\text{tr}(A) < 0$ and $\det(A) > 0$, both eigenvalues are negative.

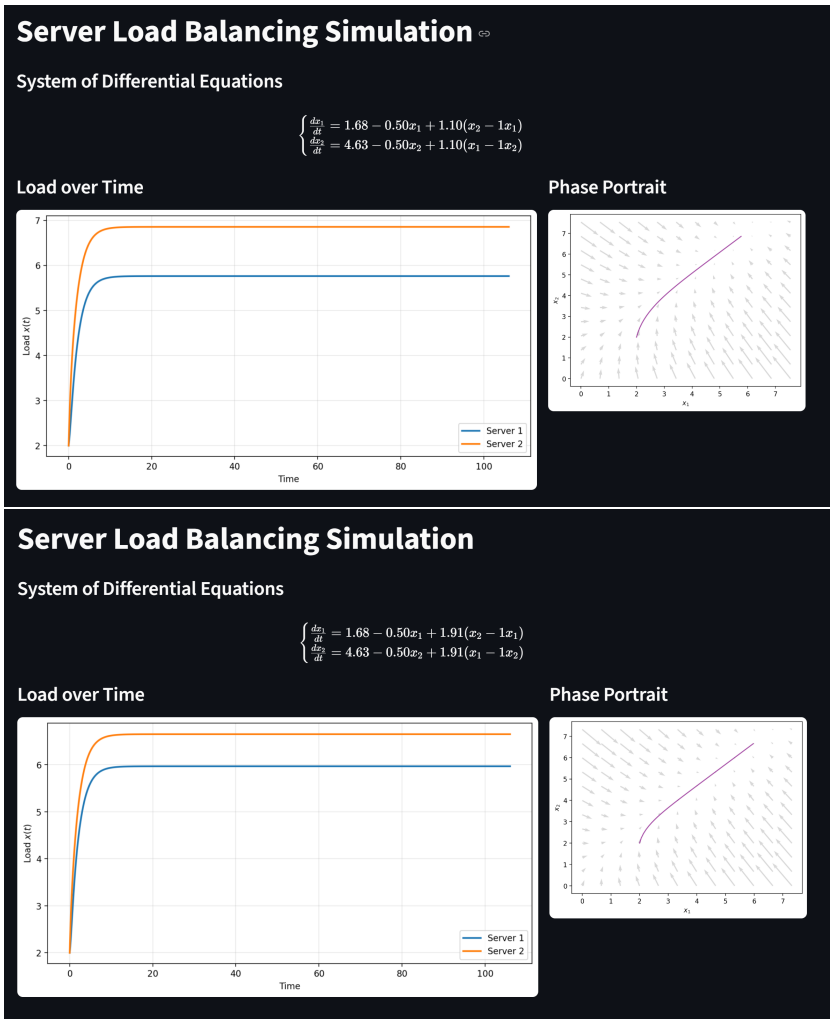
The equilibrium is an **asymptotically stable node**. All solutions converge to (x_1^*, x_2^*) as $t \to \infty$.

The coupling term $k(x_2 - x_1)$ works to balance the loads between the two systems. As k increases, the difference $|x_1 - x_2|$ decreases more quickly, leading to faster load balancing.

3.1.4 Graph Analysis

Notice in the following 3 graphs that as I increase the coupling gain term k, the system tends to converge to a faster load balancing. As said above, the difference $|x_1 - x_2|$ also decreases more quickly.





3.2 3 Server Model

3.2.1 Model

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \lambda_1 - \mu_1 x_1 + k(x_2 - x_1) + k(x_3 - x_1) \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \lambda_2 - \mu_2 x_2 + k(x_1 - x_2) + k(x_3 - x_2) \\ \frac{\mathrm{d}x_3}{\mathrm{d}t} = \lambda_3 - \mu_3 x_3 + k(x_1 - x_3) + k(x_2 - x_3) \end{cases}$$

3.2.2 Simulation Settings

To view this in a simulation, select:

- **Arrival Algorithm:** Decoupled Static

- **Processing Mode:** Proportional or Saturated (this exact model uses proportional), however saturated is more realistic
- **Number of Servers:** 3
- **Coupling Gain** (k)**:** > 0

3.2.3 Analysis

Simplified Analysis of the 3-Server System

To determine stability, we compute the characteristic equation:

$$\det(A - \lambda I) = 0.$$

That is,

$$\begin{vmatrix} -(\mu_1 + 2k) - \lambda & k & k \\ k & -(\mu_2 + 2k) - \lambda & k \\ k & k & -(\mu_3 + 2k) - \lambda \end{vmatrix} = 0.$$

Rather than expanding this determinant, we note:

- All diagonal entries are negative
- Off-diagonal entries are positive
- Each row is diagonally dominant: $\mu_i + 2k > 2k$

These imply that all eigenvalues λ are negative.

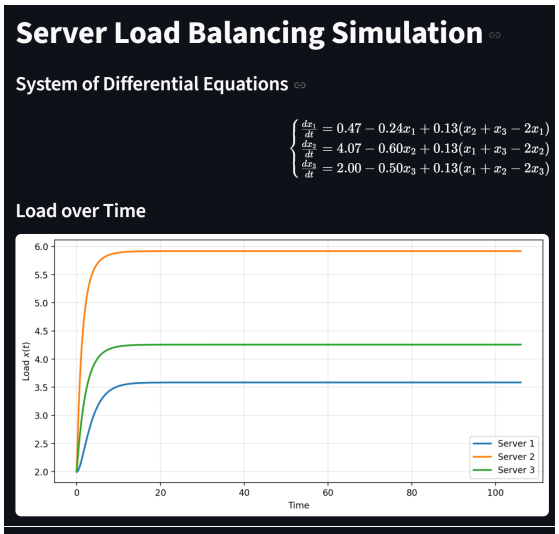
The system is **asymptotically stable**. All solutions converge to (x_1^*, x_2^*, x_3^*) as $t \to \infty$.

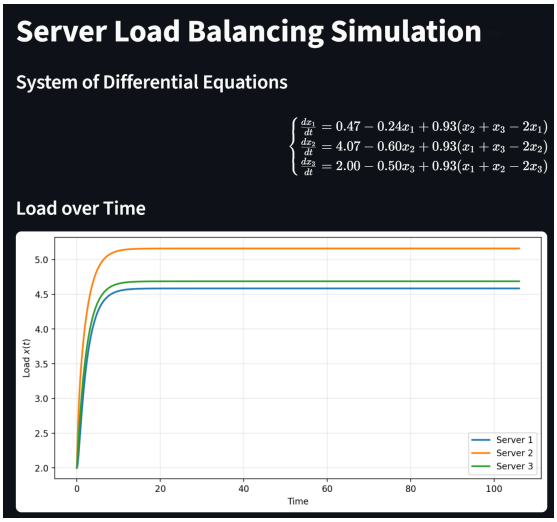
The coupling terms $k(x_j - x_i)$ balance the loads across the systems. As k increases, the differences between x_1, x_2, and x_3 decrease more quickly, and for large k,

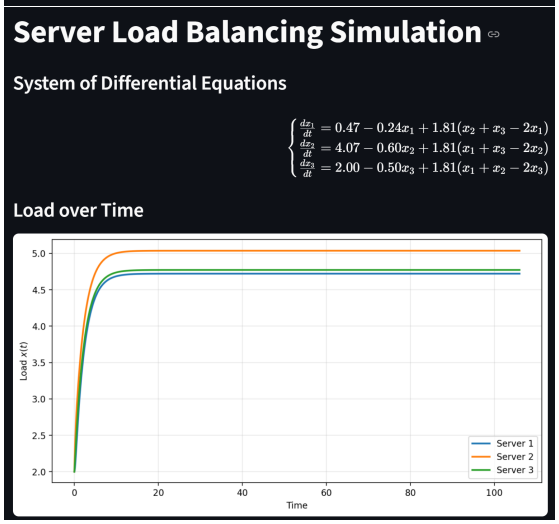
$$x_1 \approx x_2 \approx x_3.$$

3.2.4 Graph Analysis

Notice that the same phenomenon that we saw in the 2 server model appears in the 3 server model as depicted below.







3.3 N Server Model

We can expand this to deal with n number of servers by applying the same logic. That is

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \lambda_i - \mu_i x_i + k \sum_{j=1}^{n} (x_j - x_i)$$

We will not be covering this in depth, so this is a more of a note. The same logic we apply for the rest of the report can be expanded to any $n \geq 2$ servers.

4 Load Balancing Methods

Right now, when analyzing two servers, we use a method where each server gives off tasks to the other server if it has more in proportion to the constant k. While the model works, it is not realistic to how actual load balancing is done. That is done through load balancing algorithms.

Instead of relying on a coupling factor, like we did previously, we can instead put a load balancing "server" before our servers. In this case, we can deligate tasks to each other. So, we will now be changing our incoming job rate, λ_i. While we can completely get rid of k sometimes a server needs to be rescued, that is it may have a low processing rate, and may be given too many jobs. So, we will keep a small coupling factor k.

For the rest of this section, we will define Λ as the total number of incoming jobs, and distribute them accordingly.

Due to the complexity of these balancing methods, we will resort to just pure graph analysis rather than algebraic methods.

4.1 Round Robin

Round Robin means we delegate tasks equally to server. If four tasks come in, 2 will go to server 1, and 2 will go to server 2.

Simplified, this means that $\lambda_i = \frac{\Lambda}{n}$. For our two server model, $\lambda_i = \frac{\Lambda}{2}$.

4.1.1 Model

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \frac{\Lambda}{2} - \mu_1 x_1 + k(x_2 - x_1) \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \frac{\Lambda}{2} - \mu_2 x_2 + k(x_1 - x_2) \end{cases}$$

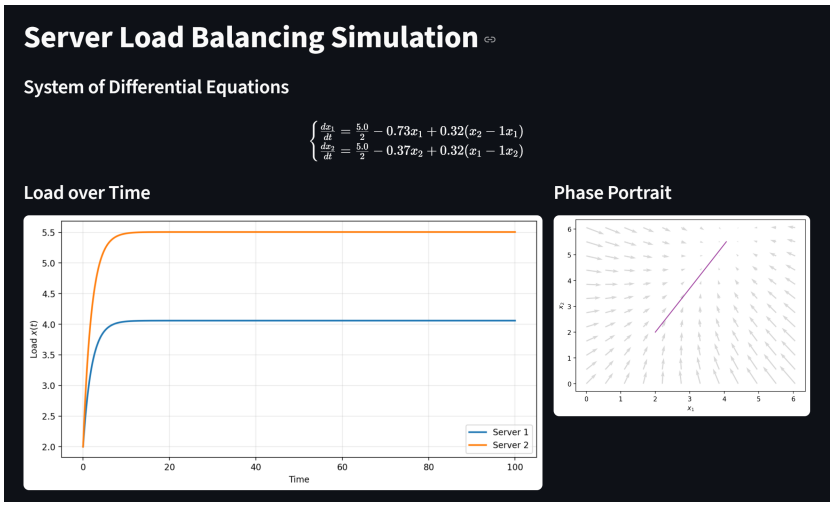
4.1.2 Simulation Settings

To view this in a simulation, select:

- **Arrival Algorithm:** Round Robin
- **Processing Mode:** Proportional or Saturated (this exact model uses proportional), however saturated is more realistic
- **Number of Servers:** 2
- **Coupling Gain** (k)**:** Keep k low

4.1.3 Graph Analysis

Because tasks are distributed evenly, the server with the higher processing rate will finish more tasks and converge to a higher limit, as seen below.



4.2 Weighted Round Robin

A standard round robin does not take into account the processing power that a server has. A server with more processing power can handle more jobs, so we delegate it more jobs. That is,

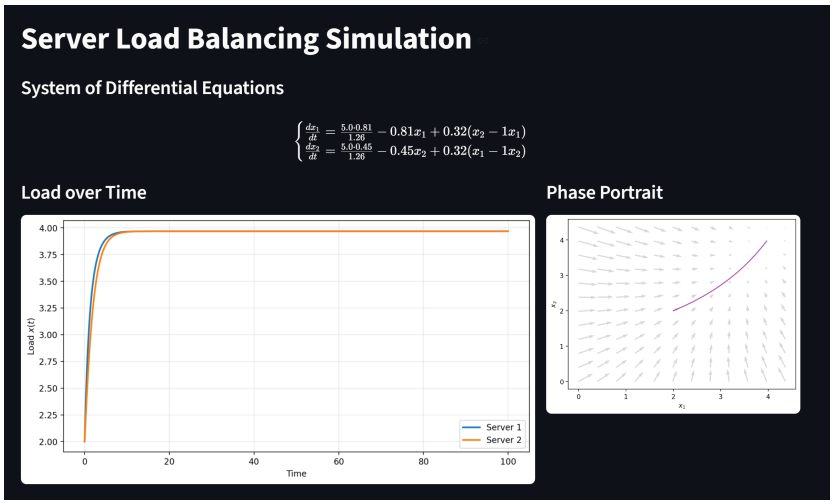
$$\lambda_i = \Lambda \left(\frac{\mu_i}{\sum \mu_j} \right)$$

4.2.1 Model

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \frac{\Lambda \mu_1}{\mu_1 + \mu_2} - \mu_1 x_1 + k(x_2 - x_1) \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \frac{\Lambda \mu_2}{\mu_1 + \mu_2} - \mu_2 x_2 + k(x_1 - x_2) \end{cases}$$

4.2.2 Graph Analysis

In contrasts with regular round robin, weighted round round takes into account that a server with higher processing power can handle more jobs and vice versa. Because of this, we expect the two servers to converge to the same load, since the system makes sure to optimize each server. We can see this interaction below.



4.3 Least Connections

Least Connections is similar to Weighted Round Robin, but instead it focuses on giving more tasks to servers with the least amount of tasks.

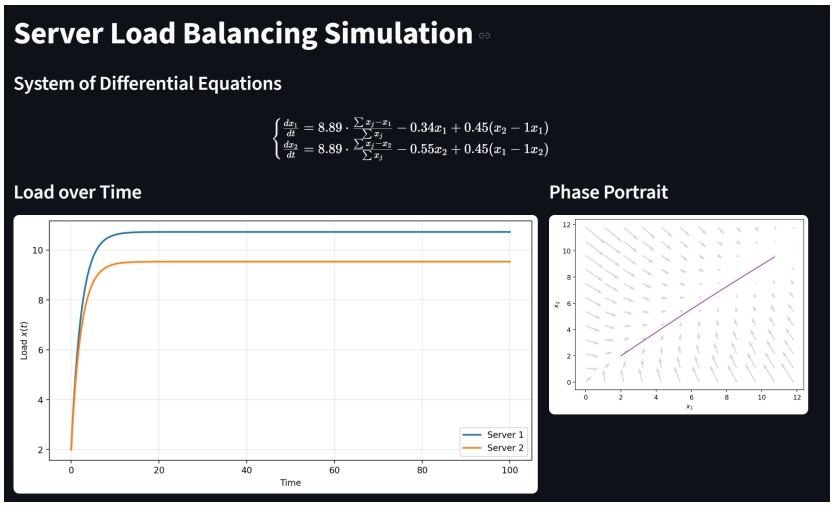
$$\lambda_i = \Lambda \left(\frac{\sum(x_j) - x_i}{\sum x_j} \right)$$

4.3.1 Model

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \frac{\Lambda x_2}{x_1 + x_2} - \mu_1 x_1 + k(x_2 - x_1) \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \frac{\Lambda x_1}{x_1 + x_2} - \mu_2 x_2 + k(x_1 - x_2) \end{cases}$$

4.3.2 Graph Analysis

Least Connections should still yield a converging point for each server, which may or may not have the same converging point depending on the processing rate. Different processing rates with still yield to different loads as seen below.



4.4 Weighted Least Connections

We can actually have the best of both worlds in a sense. We can instead of prioritizing least jobs or highest processing rate to give new jobs to, we can prioritize expected waiting time. Define

$$W_i = \frac{x_i}{\mu_i}$$

This is the amount of time we expect server i needs to finish its current amount of jobs. Servers with lower W_i should get more jobs, so by following the same logic as least connections, we get:

$$\lambda_i = \Lambda \left(\frac{\sum(W_j) - W_i}{\sum W_j} \right)$$

4.4.1 Model

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = \frac{\Lambda W_2}{W_1+W_2} - \mu_1 x_1 + k(x_2 - x_1) \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = \frac{\Lambda W_1}{W_1+W_2} - \mu_2 x_2 + k(x_1 - x_2) \end{cases}$$

4.4.2 Graph Analysis

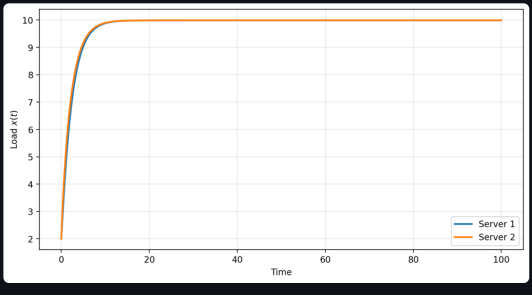
Again, like the Weighted Round Robin, using Weighted Least Connections optimizes the processing rate of each server. This allows both graphs to converge to the same point, since each server is optimized to a certain load capacity.

Server Load Balancing Simulation

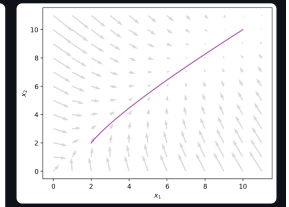
System of Differential Equations

$$\begin{cases} \frac{dx_1}{dt} = 8.89 \cdot \frac{\sum W_j - W_1}{\sum W_j} - 0.34x_1 + 0.45(x_2 - 1x_1) \\ \frac{dx_2}{dt} = 8.89 \cdot \frac{\sum W_j - W_2}{\sum W_j} - 0.55x_2 + 0.45(x_1 - 1x_2) \end{cases}$$

Load over Time



Phase Portrait



5 Appendix

5.1 More on the Simulation

5.1.1 View the Simulation

You can view the simulation here: `https://loadbalancing.streamlit.app/`. Please note that this is hosted on Streamlit and may need to be "woken up", which can take a few minutes.

The source code can also be viewed here: `https://github.com/krishrenjen/M252-Load-Balancing`

5.1.2 Simulation Method

We use the RK4 method for simulating instead of Euler's method as it is more accurate and stable as we do not need the speed from Euler's.

5.1.3 More Models

This report only covers some of the main models. The simulation allows you to combine a lot more methods together, and explore them. For instance, you can use a basic coupled model with fixed processing rates, which is not mentioned in this report.