Università Ca'Foscari Venezia

Scalability and Performance Analysis of a Web Application

Laureanda/Laureando
Federico Carraro, 847386
Andrea Brion, 860595

Anno Accademico
2024/2025

Università
Ca'Foscari
Venezia

2:
introduzione e scopo del progetto

Step 1
Create a web application that allows for searching a movie in the database and provides the information on the movies, the main actors and directors and the average rating, length etc.
Design your application with the tools that you prefer.

Step 2
Create a query set. Let us assume that the probability that a film is searched is proportional to the number of rating that it has received.
Create a list of 10,000 queries of movie titles (where entries can be duplicated) sampled according to this rules.

The goal of the project is to identify the element in the network that limits its performance, the so called bottleneck.
Afterwards we'll define the optimal parameters and show how targeted resource increases can reduce the bottleneck or even shift it to another component.

Step 3
Perform a load test to assess the scalability of your web app in closed-loop.
How many users can it handle?
Use the queries that you previously created as input of your load test.

Step 4
Design the system in such a way that its scalability increases.
To this aim, identify the bottleneck in your implementation (either experimentally or from its model).
Then, test your designed system using JMT to study the expected response time as function of the number of users. What is the optimal number of users?

3:System architecture:

Our web application is made up of a javaScript file, given that our goal was solely to test performance, the back-end logic was implemented within this file: the application was designed to send 10,000 queries to the database.
Node.js enabled the JavaScript file to interact with the database, which was set up locally using PostgreSQL.
In accordance with the project guidelines, the data was sourced from the IMDB website.

Università
Ca'Foscari
Venezia

4:Load testing setup:

In order to perform load testing in our application we relied on Apache Jmeter load tester software.
We created a test plan, which defines the overall settings for the test, then added to it two distinct thread groups:

- one to simulate server-to-database requests
- one to simulate user-to-server requests

Now, we'll take a closer look at both.

The former one simulates the interaction between the web application and the database and it contains two main components:

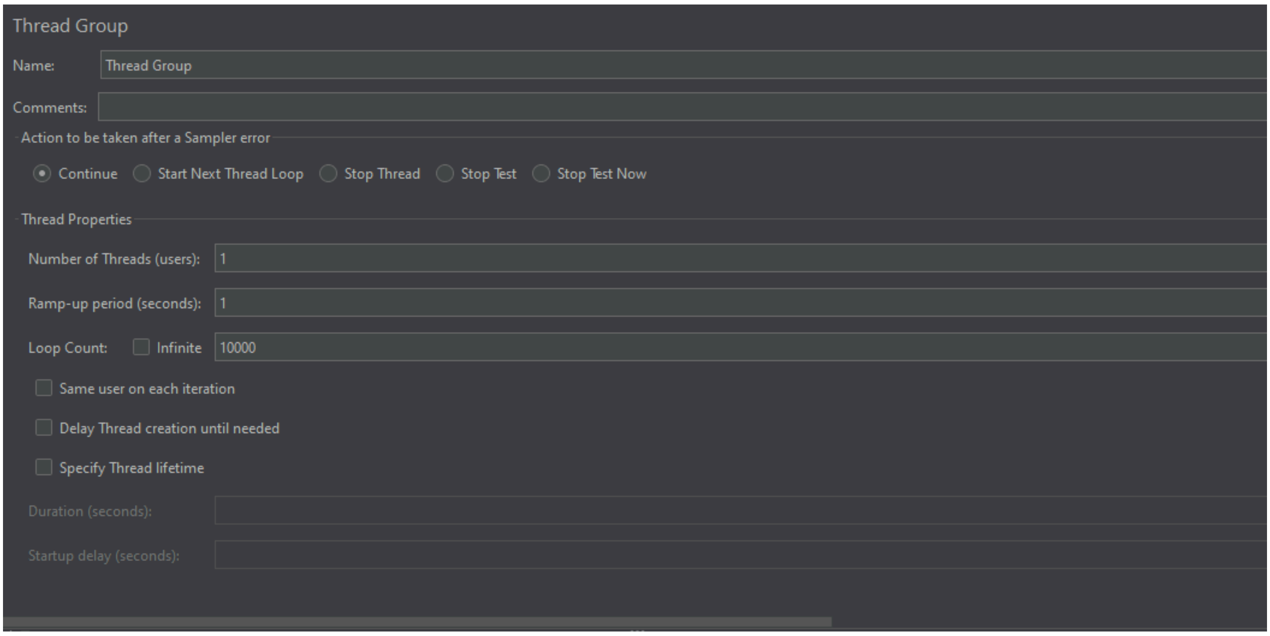
- **Configuration element**, that establishes a connection to the database
- **JDBC sampler**, in which we've defined a query the web application use to retrieve data from the database and show it in the web application.

Worth mention that we've decided to set a maximum number of connections (into the configuration element) of 50, this because if the default value (1) was kept we've seen a error rate too high regarding the requests.

The latter simulates HTTP interactions between users and the server and it contains two main components:

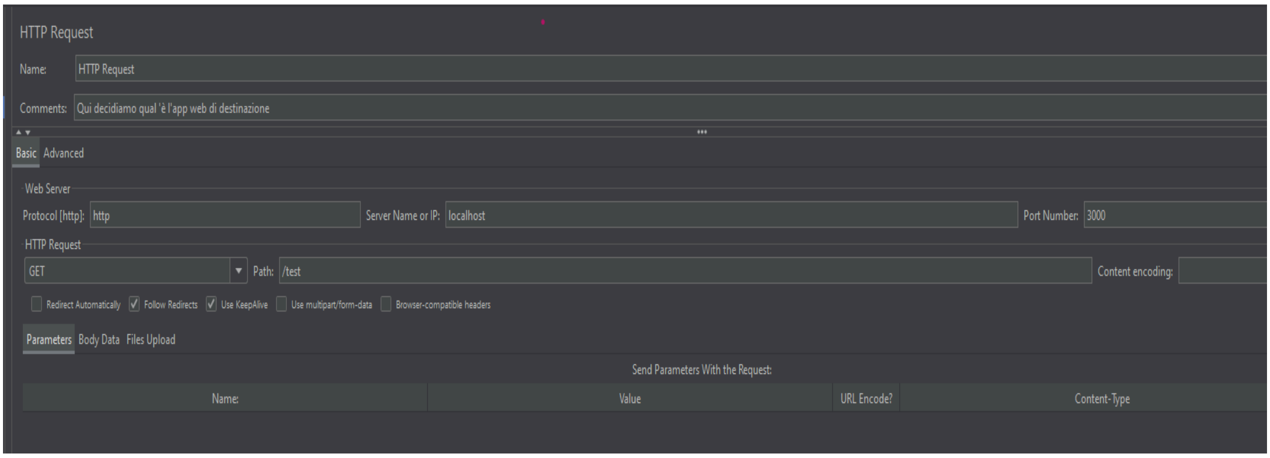
- **HTTP sampler**, which allows sending HTTP/HTTPS requests to a web server and also parse embedded resources (e.g., images) when instructed, simulating realistic HTML content retrieval.
- **Flow Control Action sampler** configured for Pause only to add a **Constant Throughput Timer** that is the key factor in evaluating system behavior under a controlled load.
  This timer regulates the number of requests sent per minute or second (it dynamically adjust pacing to achieve a defined throughput), here we've set a random delay with an average and deviation to achieve a more realistic user interactions

Right below is possible to see all the settings of the tests plan above described:

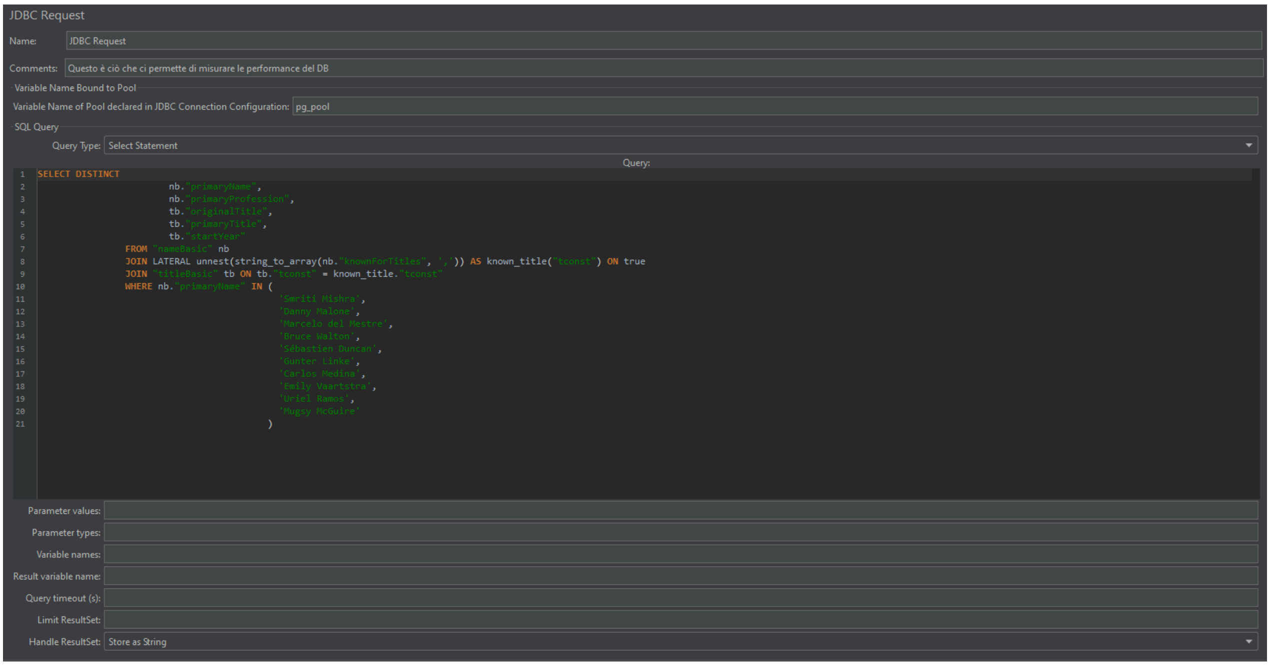


The thread group used for the server-to-database requests.
Here we've only set the number of queries that a user sends: 10000, as indicated in the project specification.

Università
Ca'Foscari
Venezia

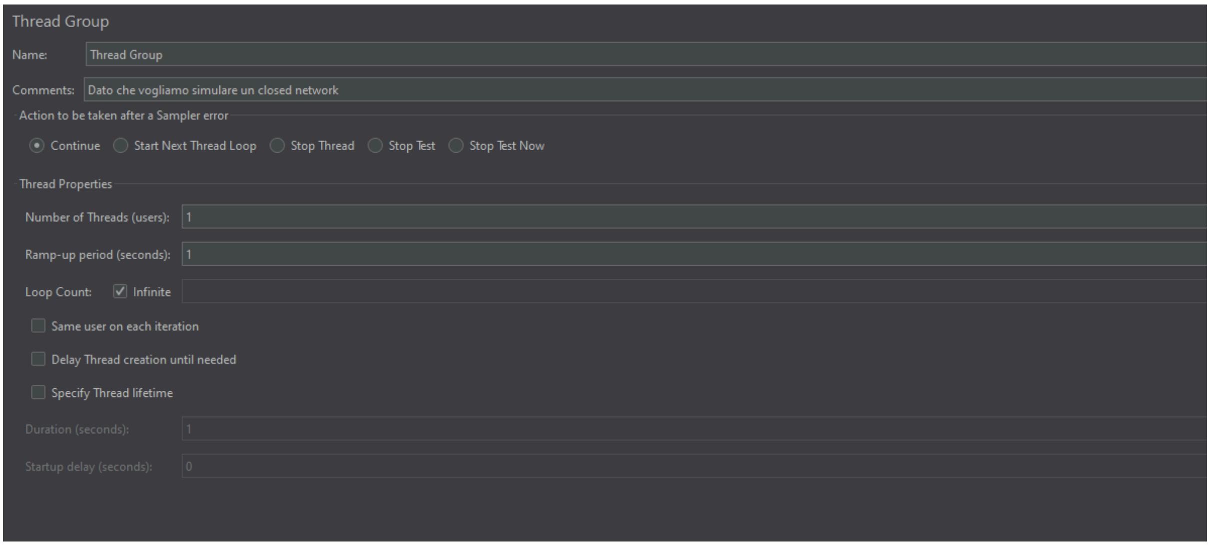


The JDBC configuration element, where it's possible to see the database's connection settings.
We've used a local database in the configuration, in fact the URL specified is a local one

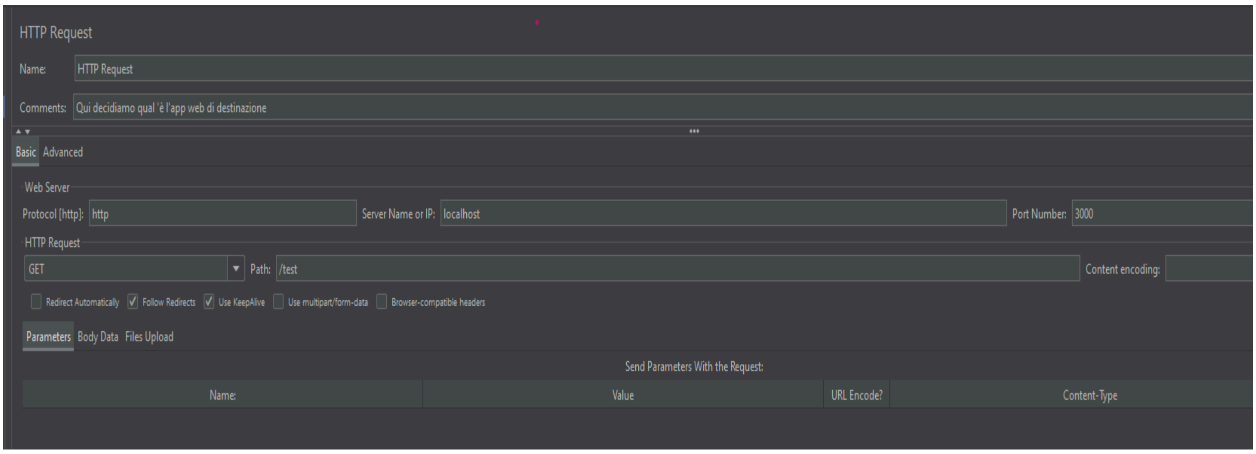


The JDBC sampler.

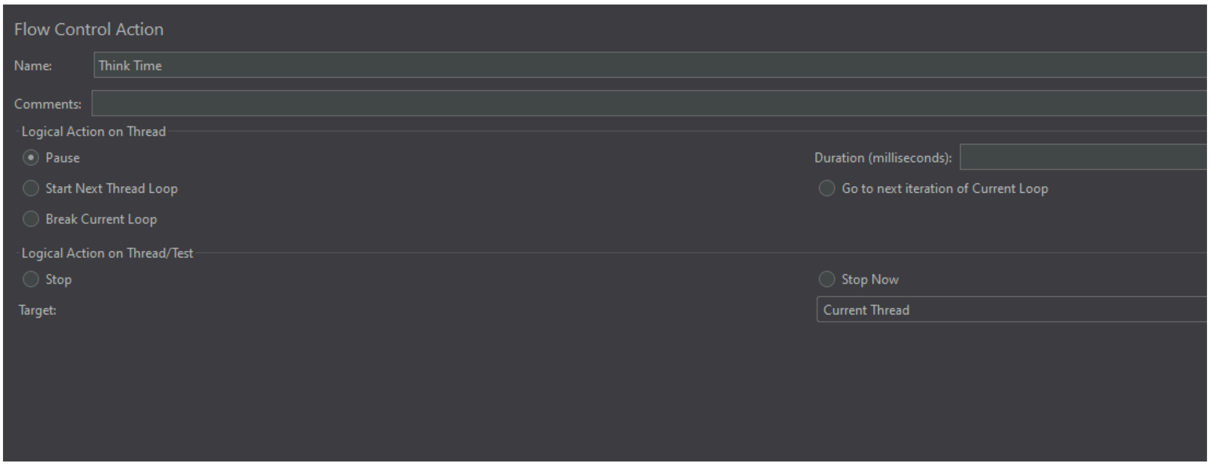
Università
Ca'Foscari
Venezia



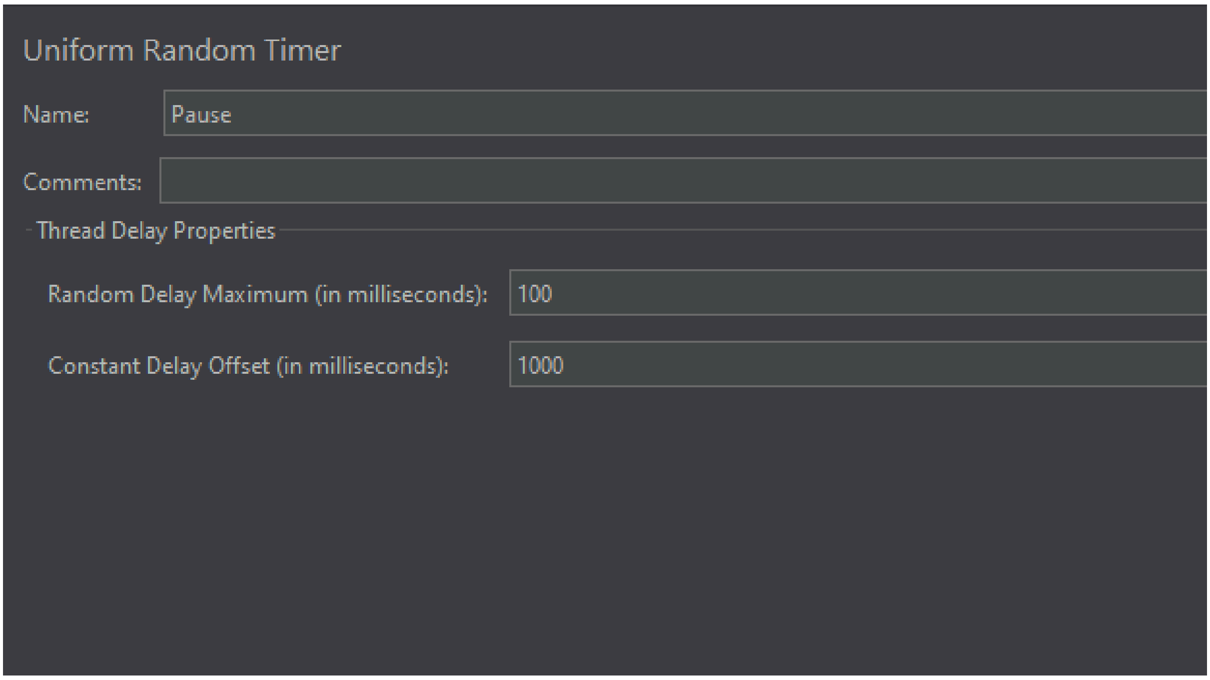
The thread group used for the user-to-server requests.
We've put the flag to "Loop count: Infinite" to simulate a real situation where the users are not limited in the number of requests they can make.



Here we see the HTTP Sampler, where we specified the protocol, the server name, the port (of the web application server), the type of request and the path to the resource that allows to the users to get the data.



Here we see Flow Control Action.



Here we see the Constant Throughput Timer, where we’ve set the two delays.

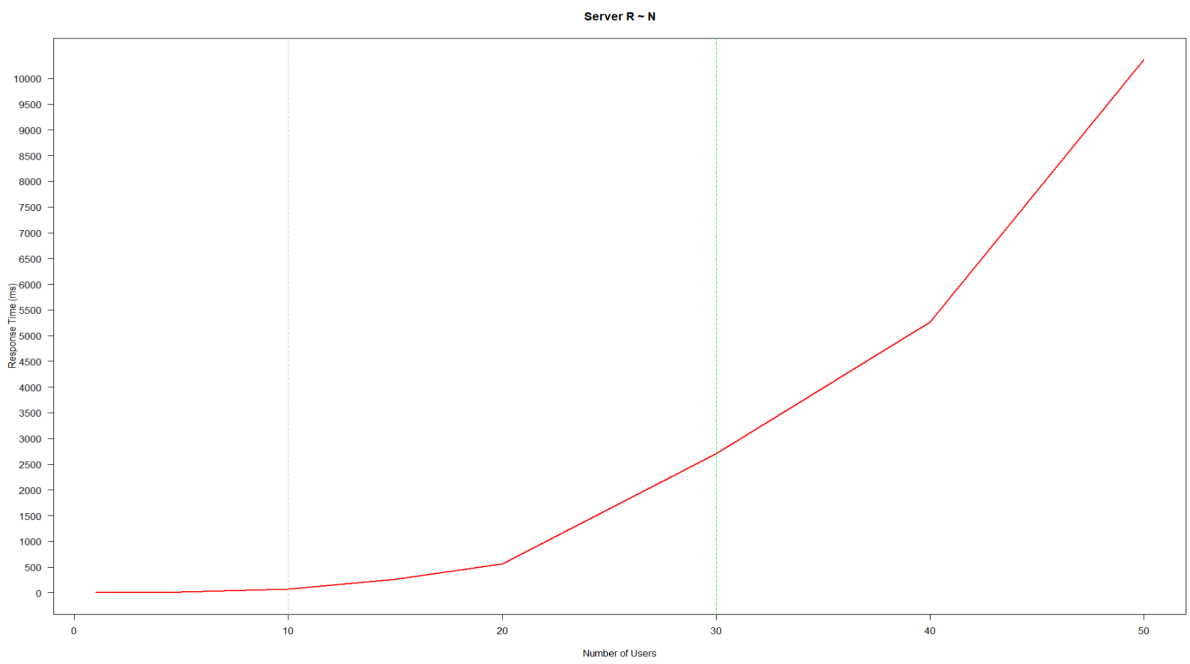
Università
Ca'Foscari
Venezia

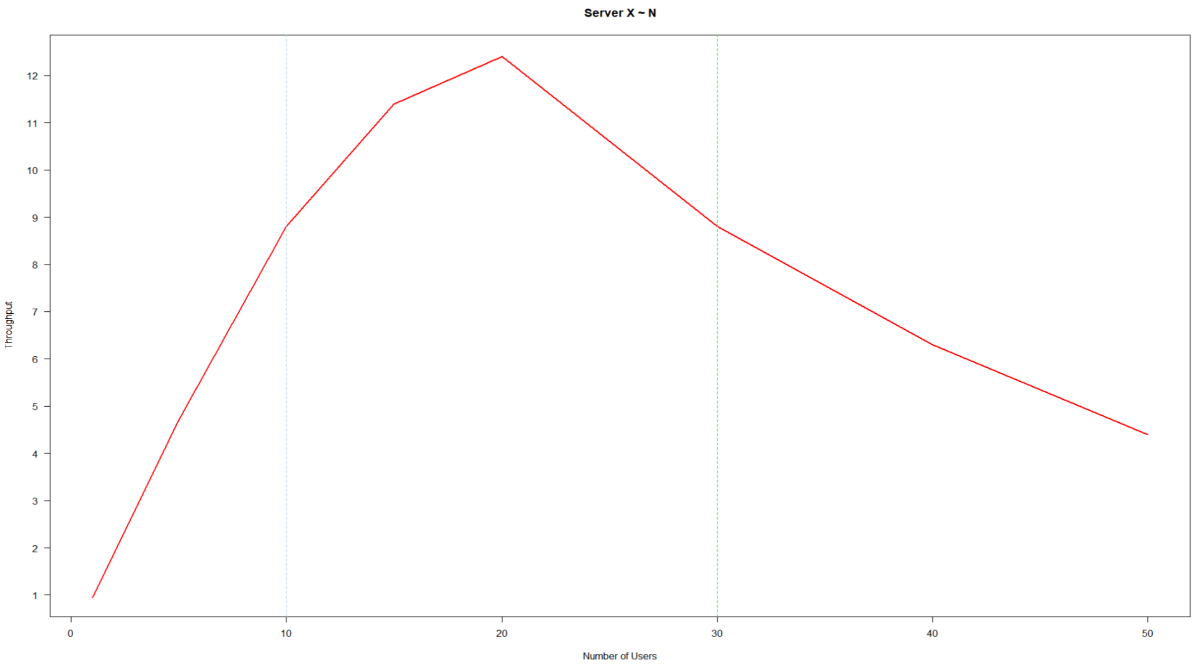
5:

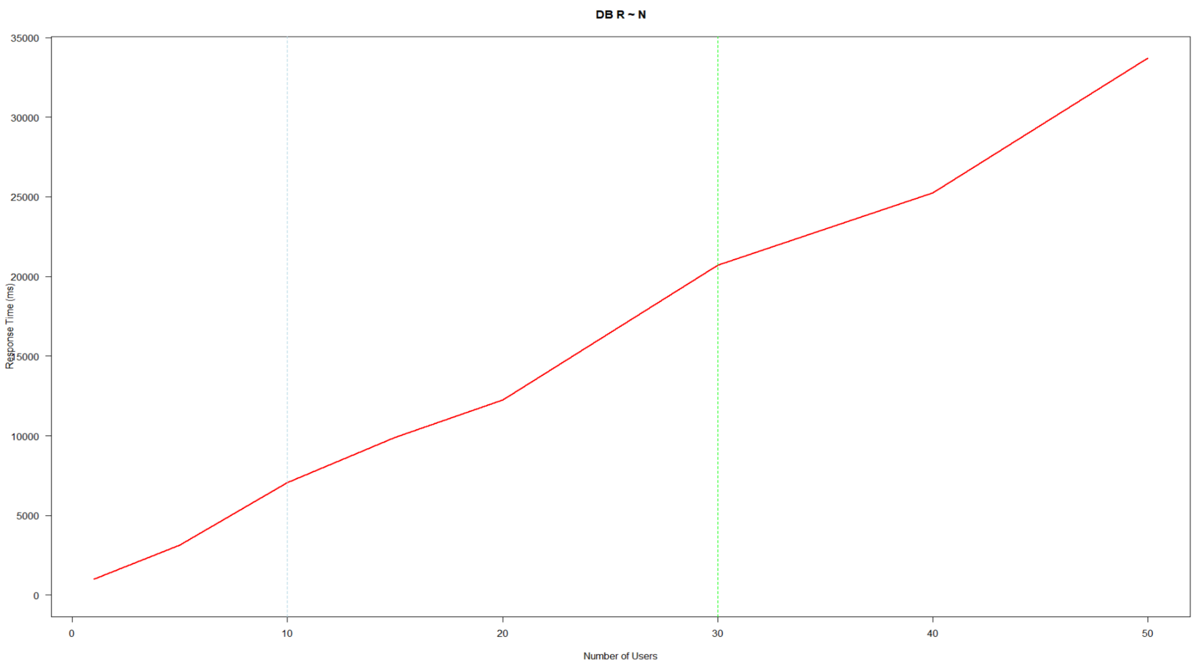
The results of **jMeter** obtained with the increase of the number of customer.
To better readability we've used rStudio to make the plots.
Analyzing the results we've found that

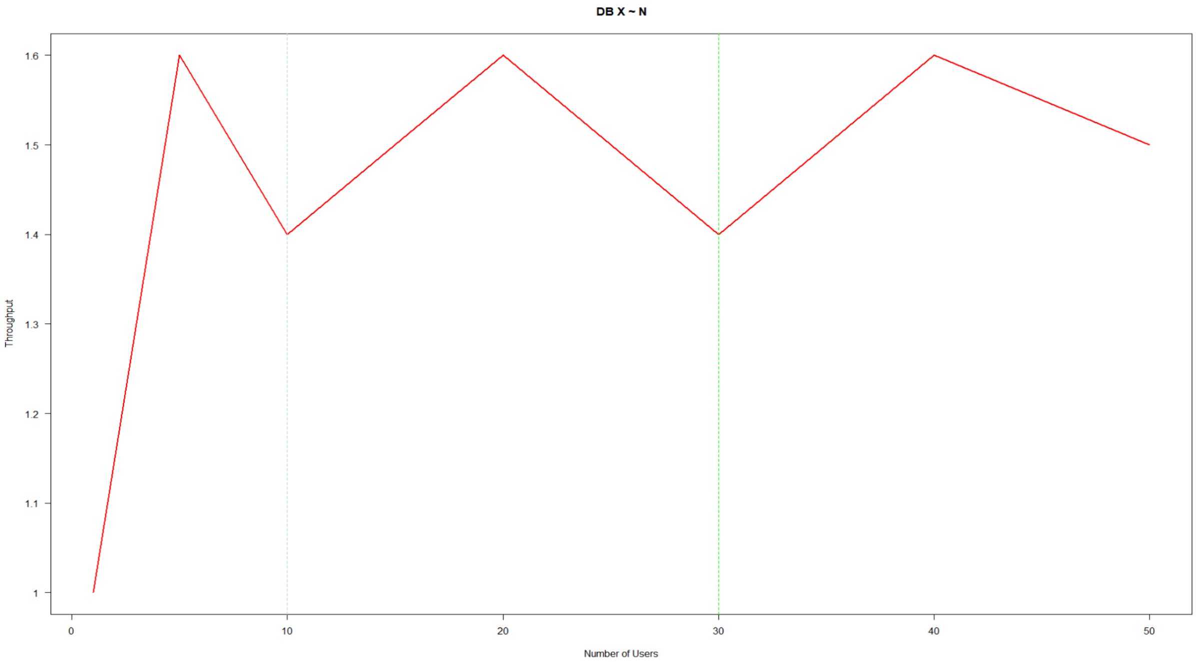
- the low load is with a number of customer below 10
- the moderate load is with a number of customer below 30
- over 50 customer we have heavy load

As indicated by the dashed lines in the graphs that follows:



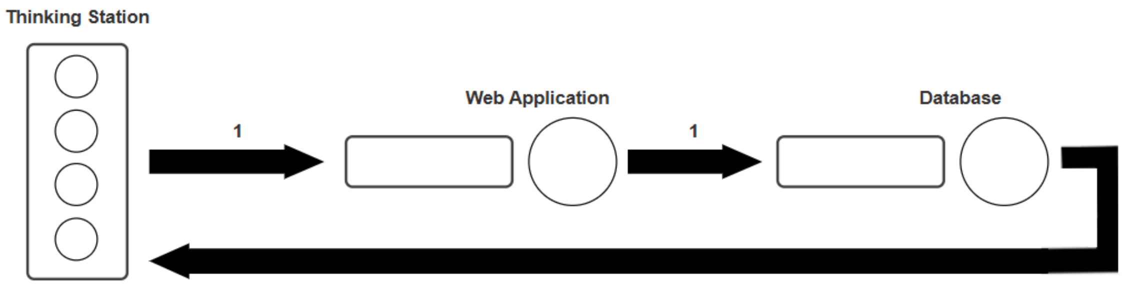

Server X ~ N
Throughput
Number of Users
0
10
20
30
40
50
1
2
3
4
5
6
7
8
9
10
11
12


DB R ~ N
Response Time (ms)
Number of Users
0
10
20
30
40
50
0
5000
10000
15000
20000
25000
30000
35000



6:*Analytical model:*

In order to perform load testing we decided to structure the system under test in an **interactive single-class closed-system** depicted in the following way:



Being a closed system queueing network the web server is going to serve a fixed number of users placed in the **thinking station** in which they send parallel and independent requests to the **web application station** that processes them.
The **database station** produces a response for each one and send back to each related user which will consult it for a certain amount of time called thinking time before doing another request.

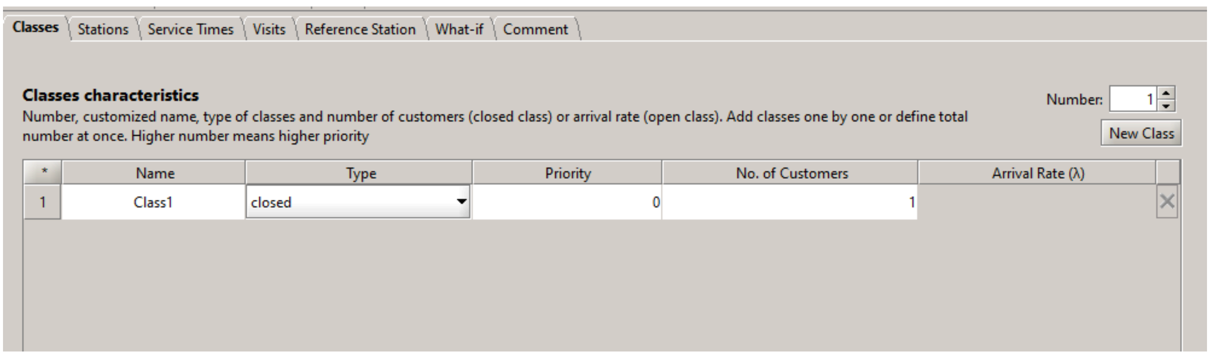
We assume for the sake of simplicity that the web application and the database stations are single server with constant speed while users are statistically identical and so single-class.
In addition, each station is always visited by every user so the relative visit ratio for each one is equal to 1.

6.1:Model analysis by JMT

The settings in JMT touch various sections, let's dive into each of them:

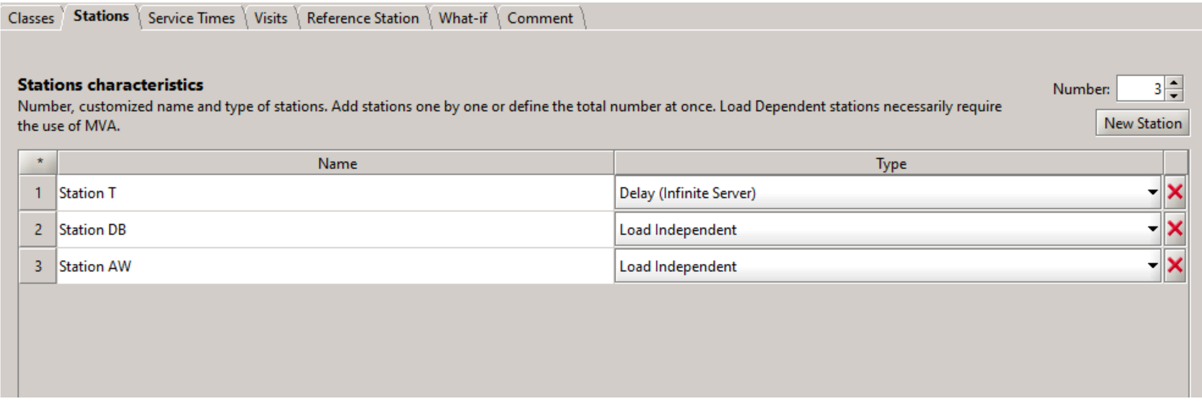
- Classes, here we've set the network type and the types/classes of the requests, respectively: closed, single-class
- Stations, here the number and type of stations are set, respectively: thinking station, web-application station, database station
- Service Times, here the results obtained with jMeter are used to set each station service time
- Visits, here the visit ratios are set as one visit per station each visit to the reference station
- Reference station, here which stations is used as reference station: the thinking station
- What-if, here we set the bound of the experiment and the parameter to increase, respectively: 1 to 50 customer, number of customer

Right below the settings we've used described above:

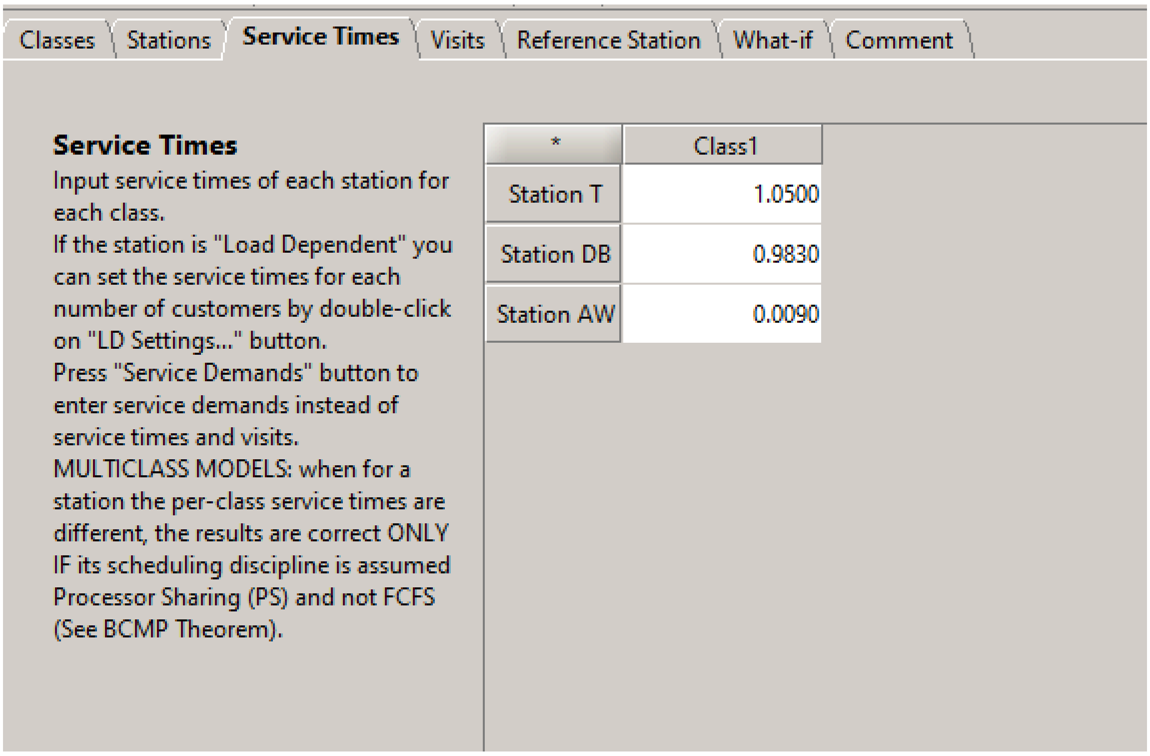


The classes.

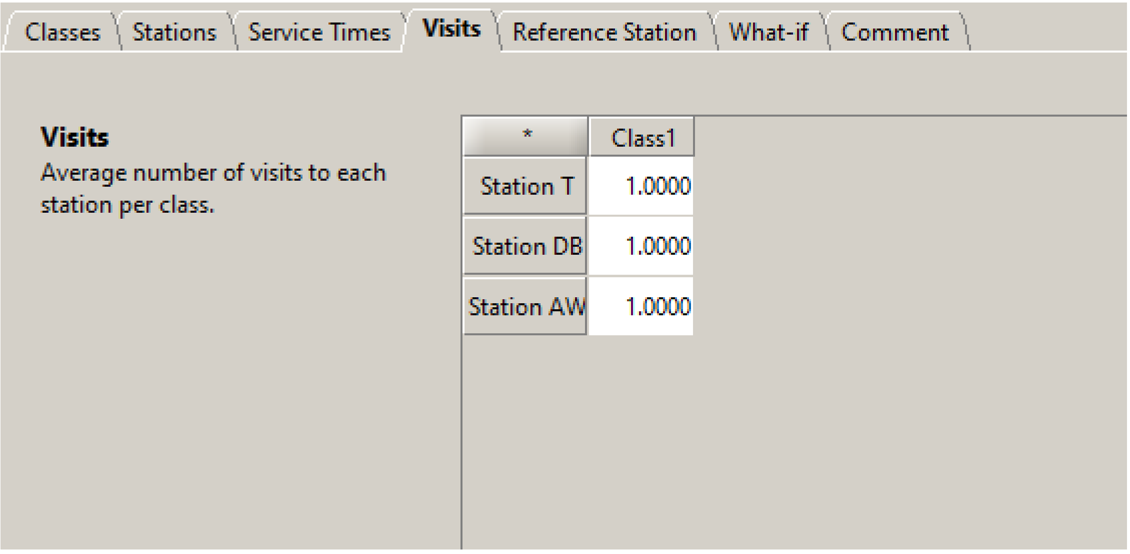
Università
Ca'Foscari
Venezia



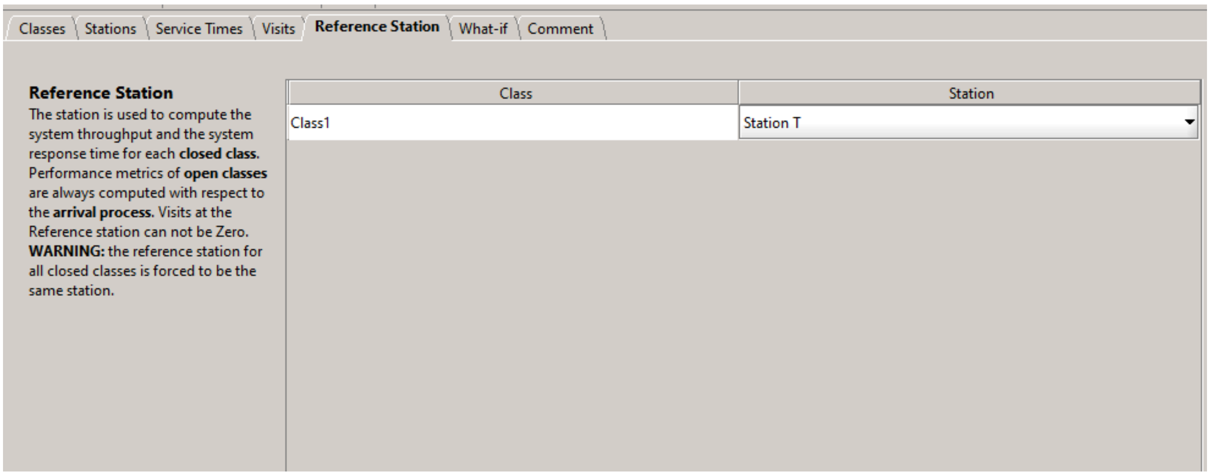
The stations.



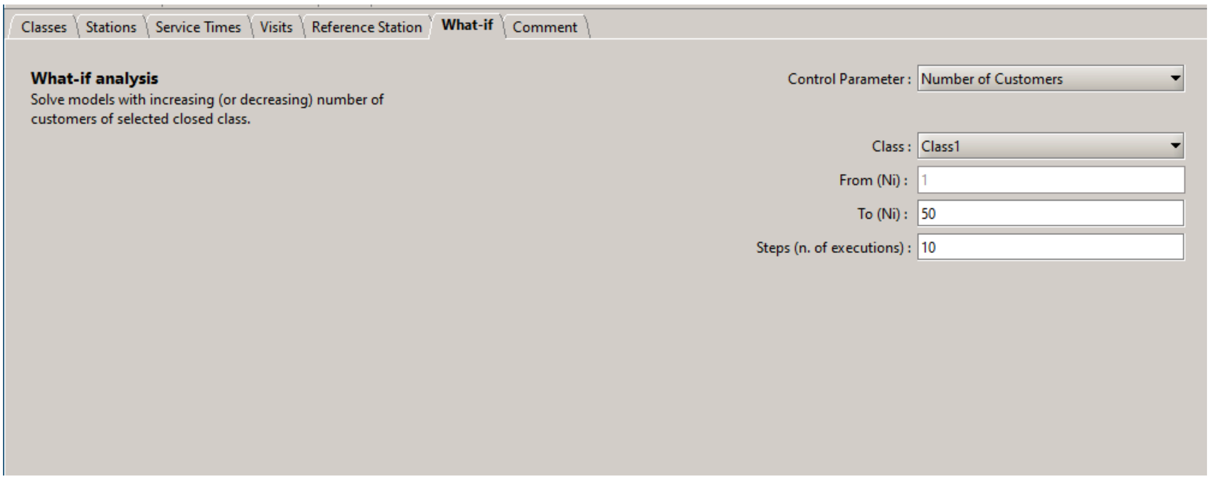
The service times.



The visit ratio.



The reference station



The What-If.

Università
Ca'Foscari
Venezia

grafici di throughput,response time ed utilization
low,moderate ed heavy load

6.2:Analisi bottleneck mediante traffic equations,forced flow law e bottleneck law e grafici

Bottleneck

In order to spot the bottleneck we determine the average service demand for each station that is the total amount of service required by a customer to each station **i** for each visit it does at the reference station,assuming single server systems and independent service time distribution from the number of visits performed by a job at queue:

$$E[D_i]=E[V_i]*1/\mu_i$$

and then the highest one determines the bottleneck station.

In our system we have that:
each station i is visited with visit-ratio: $E[V_i] = 1, \quad i = 1,2,3$
service demand web application: $E[D_{app}] = E[V_{app}]*1/\mu_{app} = 1*0.0090=0.0090$
service demand database: $E[D_{DB}] = E[V_{DB}]*1/\mu_{DB} = 1*0.9830=0.9830$

Therefore the database station is the bottleneck!

8:Calcolo optimal number of users sia analitico che grafico mettendo in risalto il throughput e response time con i relativi bound

Optimal number of users:

Università
Ca'Foscari
Venezia

After having identified the bottleneck, what happens to our system when the number of customers increases?

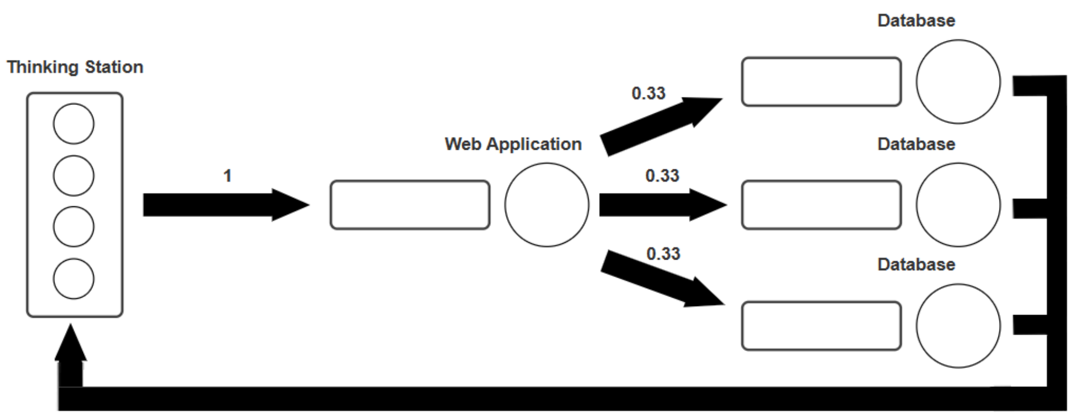
is going to be the first to have its utilization equal to 1,because the jobs tend to accumulate at that station and as a consequence limiting the system performances,namely the throughput.

In fact,the forced flow law states that the system throughput X_{sys} which corresponds to the reference station one is equal to $\mu_{bottleneck}/E[V]_{bottleneck}$

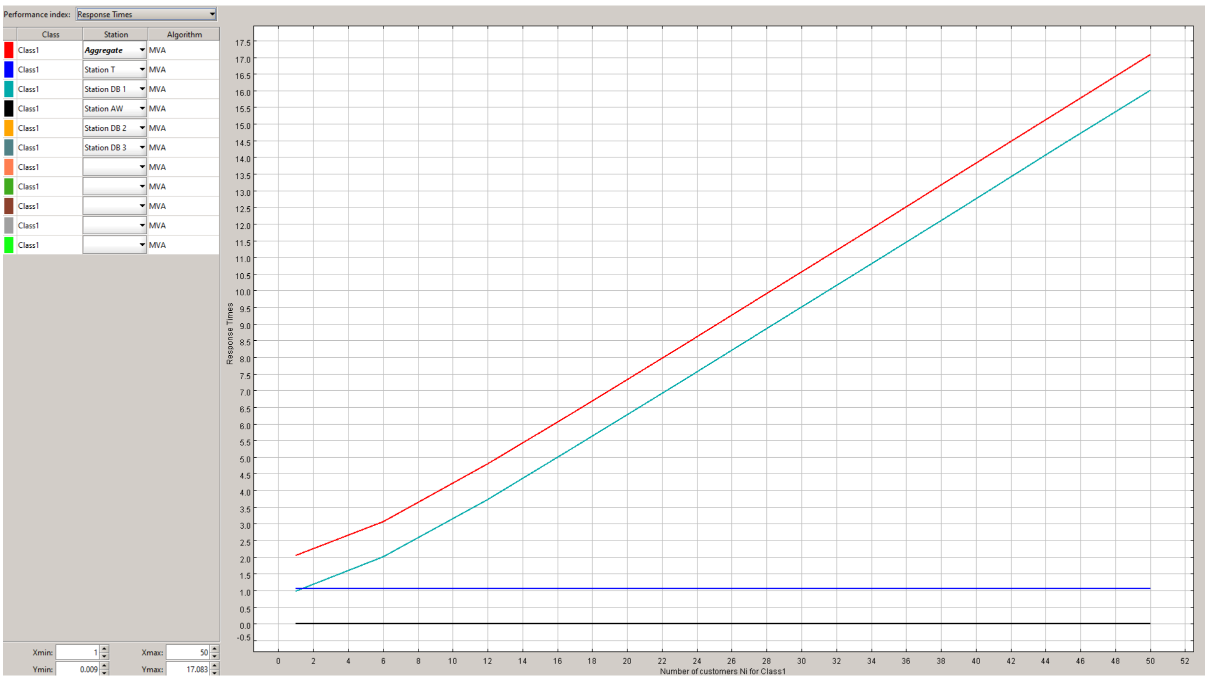
which imposes asymptotic limits to our system

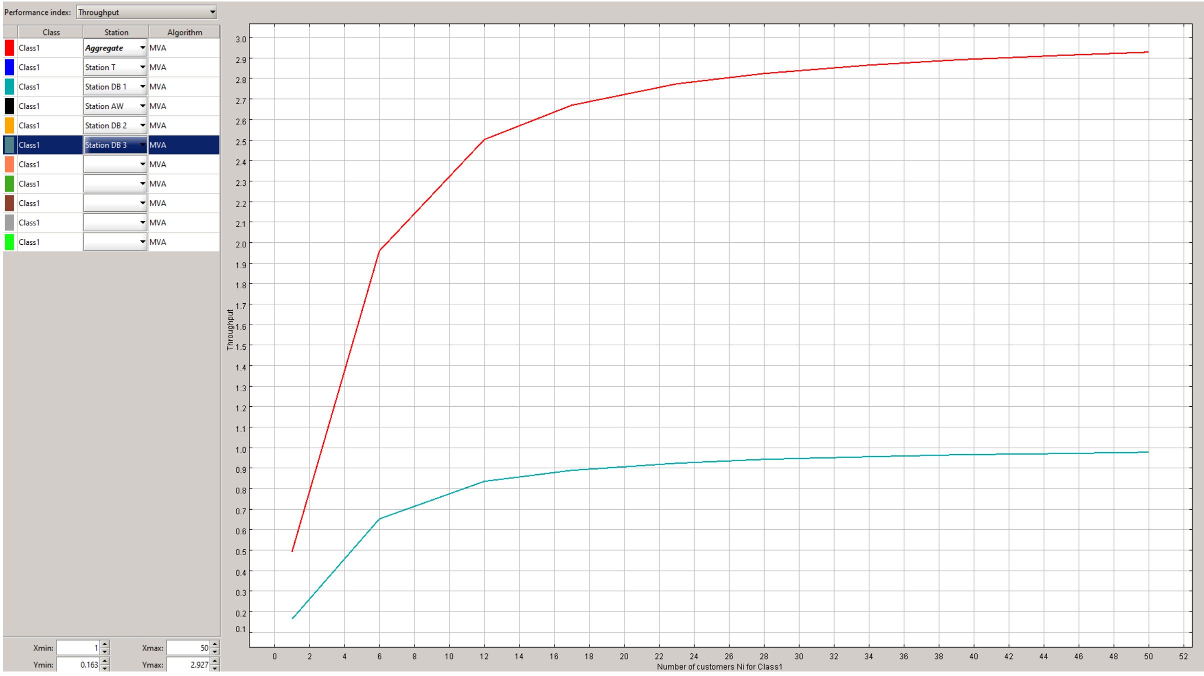
9:ottimizzazione bottleneck via JMT con relative analisi throughput,response time e optimal number of users.

Following the literature, we have attempted (and succeeded) in reducing the bottleneck effects by increasing the number of stations of the same type as the bottleneck station.
We kept all settings the same for the additional stations, except for the visit ratio, which was set to 0.33 for each database station.
This value is derived by assigning an equal probability for a request to be routed to one of the three available stations.



Right below the results in graphs:





As the reader can see, the throughput with three database stations increases by nearly a factor of three; similarly, the response time decreases by approximately the same factor.