Utilising Edge Computing in a 3-Tier System between Client Devices and a Data Centre

A dissertation submitted in partial fulfilment of the requirements for the degree of BACHELOR OF ENGINEERING in Computer Science

in

The Queens University of Belfast

by

Connor Dickson

9th May 2017

# Acknowledgements

I would like to thank Dr Hans Vandierendonck for supervising this project and for his support throughout.

I would like to thank Automated Intelligence for funding the Raspberry Pi’s used in this project.

I would also like to thank Cassio P. de Campos for his support with the Machine Learning application.

# Abstract

An investigation was done into the effects of utilising edge computing to provide benefits through use in different applications to both the client and the data centre. These benefits can be quantified though different metrics such as latency, computational resources required, and bandwidth utilised. The reason for this investigation is because of emerging problems that could be mitigated by the implementation of edge computing. These effects will be investigated in a 3-tier network by implementing a system that can utilise caching, voice recognition, and machine learning applications to provide experimental results. What was found is that edge computing provided benefits but there are trade-offs to consider in some scenarios when implementing the solution. These results are significant because of a growing strain on resources caused by the increased popularity of client devices.

Table of Contents

[1 Acknowledgements 2](#_Toc480270482)

[2 Abstract 2](#_Toc480270483)

[3 Introduction and Problem Specification 4](#_Toc480270484)

[4 System Requirements Specification 6](#_Toc480270485)

[5 Design 7](#_Toc480270486)

[5.1 Architectural Description 7](#_Toc480270487)

[5.2 User Interface Design 8](#_Toc480270488)

[5.3 Software System Design 10](#_Toc480270489)

[5.3.1 Client 10](#_Toc480270490)

[5.3.2 Edge 12](#_Toc480270491)

[5.3.3 Data Centre 13](#_Toc480270492)

[6 Implementation and Testing 14](#_Toc480270493)

[6.1 Choice of implementation languages and development environments 14](#_Toc480270494)

[6.2 How each component was implemented 15](#_Toc480270495)

[6.2.1 Client 15](#_Toc480270496)

[6.2.2 Edge 18](#_Toc480270497)

[6.2.3 Data Centre 22](#_Toc480270498)

[6.3 Use of software libraries 23](#_Toc480270499)

[6.4 Discussion of test approach 24](#_Toc480270500)

[7 System Evaluation 24](#_Toc480270501)

[7.1 Machine learning application 24](#_Toc480270502)

[7.2 Voice recognition and caching applications 25](#_Toc480270503)

[8 Experimentation 26](#_Toc480270504)

[8.1 Does utilising edge computing reduce the latency of requests for the client? 26](#_Toc480270505)

[8.2 Voice recognition application experimentation 29](#_Toc480270506)

[8.2.1 Does utilising edge computing reduce the computational load on the client? 31](#_Toc480270507)

[8.2.2 Does utilising edge computing increase the latency of the request for the client? 32](#_Toc480270508)

[8.2.3 Does implementing edge computing reduce the network utilisation? 34](#_Toc480270509)

[8.3 Load balance application experimentation 35](#_Toc480270510)

[8.3.1 Does implementing load balancing improve the effectiveness of edge computing? 36](#_Toc480270511)

[8.3.2 Does deploying more than one instance of the application improve the effectiveness of edge computing? 39](#_Toc480270512)

[9 Conclusion 40](#_Toc480270513)

[10 References 43](#_Toc480270514)

[11 Appendices 47](#_Toc480270515)

[11.1 Caching System Tests 47](#_Toc480270516)

[11.2 Voice Recognition System Tests 47](#_Toc480270517)

[11.3 Machine Learning System Tests 49](#_Toc480270518)

[11.4 Load Balancing System Tests 51](#_Toc480270519)

[11.5 Experiment Results 52](#_Toc480270520)

[11.5.1 Caching Results 52](#_Toc480270521)

[11.5.2 Voice Recognition Results 55](#_Toc480270522)

[11.5.3 Load Balancing Results 62](#_Toc480270523)

[11.6 Minutes of meetings 80](#_Toc480270524)

# Introduction and Problem Specification



Figure – 3-tier network

An edge device is named as such as it is physically situated on the edge of a network, what this means is that it is closer in proximity to the client than the client is to the data centre. This allows edge computing [2] to pre-process some data produced by or being requested by the client on an edge device. This has the potential to reduce bandwidth [3] which is becoming more important as the number of connected devices increases rapidly and places an increased strain on the network which in turn creates a bottleneck [4]. Edge computing could also deal with problems relating to latency and computational strain [5] by reducing the distance that the request from the user has to travel and by reducing the workload executed on the client or the data centre. This would allow for more computationally demanding tasks to be performed or for the same task to be executed by more users as the workload capacity has increased.

A 3-tier-network can be seen illustrated in Figure 1. In this project, the tiers will consist of a client, connected to an edge node, which in turn is connected to the data centre and the effects of the edge computing will be investigated using this setup. The client should be the least powerful device as this will be used to represent devices such as mobile phones or tablets. The edge will be a device that should consider factors such as geographical and physical location [48] restrictions, cost, and processing capabilities. The data centre should be the most powerful device in the setup as there is usually one data centre to process the same number of requests that can be processed by many edge devices.

There is currently a growing demand for edge computing [6] as the strain being placed on data centres is growing with the increase of client and internet of things (IoT) devices. This project will investigate the effects of edge computing in different scenarios such as multiple users concurrently accessing the resources. In this network, it is expected that the client will send data to the edge node that will process the request and forward it to the data centre or handle this request and return it directly to the user.

Edge computing is a relatively new area of study [7] and research is still being performed as to investigate its effects. Performing study and experimentation in this area is a good extension of some existing research [8] and could lead to even further research being performed.

The system being implemented at the edge could be required to last up to double the expected life of a server at a data centre [9] as they are not in a geographically central location, the edge devices could be dispersed widely as to allow for many users to benefit from the effects. This means that from an administrator’s perspective the system must be easily maintained, such as the ability to host multiple edge computing solutions in one environment as it allows for the easy deployment and control of the applications. Some aspects of edge computing are not well understood, more investigation can be performed into how edge computing can be utilised in different ways, for example, to benefit the client though reductions in request latency and computational power required and the costs associated with doing so. Another area of investigation needed is the scalability of these edge solutions and how well they can handle traffic during computationally demanding tasks. These topics will affect how useful and edge computing will be and quantify the benefits provided.

It is difficult to match state-of-the-art performance on hardware that would be feasible to implement as edge nodes. Investigation will be performed into these effects on systems such as voice recognition [7] and if their effects are justifiable with lower cost hardware. The client device will be physically closer to the edge node than it is to the data centre as this allows for reduced request latency which could lead to faster processing of requests. The edge node could be located at local content distribution servers [48] as this would allow for the effects of edge computing to be shared among many customers but still a smaller subset than what the data centre will handle.

Edge computing might not be required or useful in every scenario [5] as occasionally the stronger computational processing capabilities of the data centre could be required or the added latency of utilising edge computing might outweigh the benefits. Investigation into the effects of introducing edge computing solutions on metrics such as CPU utilisation, network utilisation and request latency is required. There are a few likely candidates for this technology, the first is caching where the edge receives a request from a client and processes and stores the request data so that it can be retrieved quickly next time the user requests it. Another candidate is voice recognition where a computationally intense task is offloaded to the edge node instead of the client or data centre. A final candidate is machine learning where the edge can be used by the data centre for assistance in producing a prediction. There are a few areas of investigation for edge computing [5] such as the effects of shorter response times from geographically closer edge servers, the benefits to user experience, and the benefits to data centres. These will be investigated by using the implemented solutions.

# System Requirements Specification

There are some assumptions made about the problem area, this is that the client device and edge devices are less powerful than the data centre, this is so that the beneficial effects of introducing edge computation can be evaluated realistically. The system developed is for research purposes so it must be made clear that the system is designed to produce results that can be recorded and analysed.

The requirements for this system;

1. An application hosted on the client device that will allow for interaction between the edge node and the data centre and display results to the user
2. The development and implementation of applications capable of performing edge computing tasks on the edge nodes
3. A data centre application capable of processing requests from the edge or from the client
4. The ability to run multiple applications on the same edge node
5. Edge node applications can be easily deployed, controlled, and updated

Requirements for the caching application;

1. A user interface that can receive user data and display result metrics to the user
2. A low latency connection between the client and edge node
3. An application developed for the edge that caches requests
4. Request data returned to the client if it exists already or retrieved, stored, and returned if it does not

Requirements for the voice recognition application;

1. A user interface that can record voice data and execute voice recognition locally or remotely and display metrics about the request to the user
2. An application developed for the edge that can process a voice request
3. Voice recognition capabilities at the data centre for handling processed and unprocessed requests
4. Can handle multiple requests concurrently on both the edge and the data centre

Requirements for the machine learning application;

1. A client user interface that can receive user input and make requests to the machine learning applications
2. An application developed for the edge that can handle machine learning requests
3. A subset of the data from the data centre available on the edge
4. A recommendation to be produced for the client on the edge

The goal of this project is to produce an edge computing infrastructure capable of producing results that can be processed and analysed to provide answers to the proposed research questions. Specifically, this system will be used to be investigate;

1. Latency time for the client to obtain a response
2. The computational strain placed on the client
3. The computational strain placed on the data centre
4. Implementation of load balancing

# Design

## Architectural Description

A high-level overview of how the client, edge and data centre will be implemented is shown in Figure 2. The communication is colour coded for clarity per application traffic between the devices in the network.



Figure – System diagram

The entry point for the system will be the Electron [10] application which runs on the client Raspberry Pi 3 [11]. There will be 4 pages that the user can access from the homepage.

The edge will be comprised of 3 Raspberry Pi 3’s which will form a Docker swarm [12, 13]. The applications run in containers that can be deployed to any of the worker nodes.

There are multiple data centres depending on what application is making the request. The WebAPI [14] project will be hosted in IIS [15] and have multiple endpoints to allow for interactions from different applications

## User Interface Design



Figure – The homepage



Figure – Caching page



Figure – Voice recognition page

Figure – Machine learning login page



Figure – Machine learning page

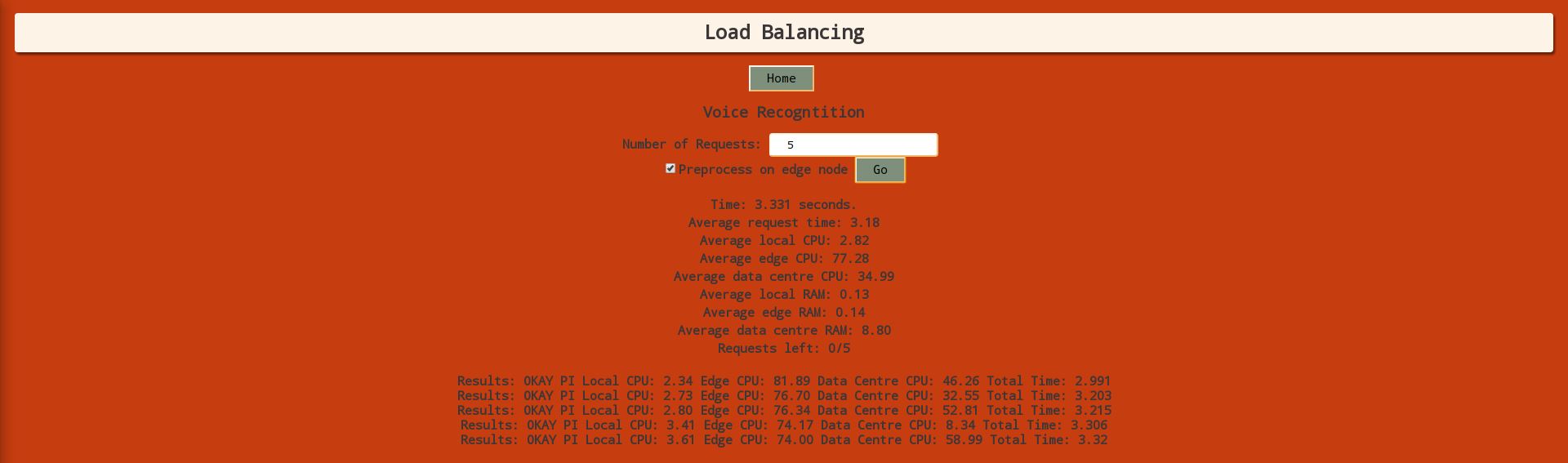


Figure – Load balancing page

The purpose of the user interface (UI) is to allow interaction with the edge nodes and data centre to perform tests and demonstrate functionality as per the requirement stated above. The homepage shown in Figure 3 shows what the user first sees when they launch the application.

The caching application page displayed in Figure 4 is required to have a place to type a URL and buttons to navigate to the stated website, navigate back to the homepage or clear the cache on the edge application. There is a section below the URL bar that presents the user with the time it took to complete the request. Then finally, the rest of the space is used to display the requested webpage.

The voice recognition application page displayed in Figure 6 requires more interaction from the user to execute a test. First is the home button that is consistent throughout the entire application. There is a button for the user to record and save their voice so the system can process it. There are a final two buttons below that which allow the user to choose between executing the voice recognition locally or remotely. The checkbox below the remote button is a modifier which indicates during the request if it should be pre-processed on the edge or processed at the data centre. The results of the experiments are shown in an easy to read table with separate sections for local and remote requests.

When the machine learning application is first launched, the user is asked to enter their username as shown in Figure 7. After the user enters their username the screen is populated with the average of their previous results and a recommendation as shown in Figure 8. There are two buttons available for the user to press, one is to watch a movie at random and the other is to watch the recommendation that was produced for them. When the user has selected either button the results about the movie they have just watched with be displayed in a third column and both the average and recommendation columns will be updated with the new information.

The final link from the homepage brings the user to the load balancing application page as displayed in Figure 8. This application has the standard home button, it them had a space for a user to enter a numerical value for the number of requests to be executed. The checkbox here performs the same action as it does in the voice recognition page, it dictates if the requests will be pre-processed or not. The “Go” button executes the requests and the screen is updated as each request finishes. There is a standard results section just below the “Go” button that displays statistics about the load balancing such as total time and average CPU use.

There is a consistent theme kept throughout the application, both in the colour scheme and the layout. It was designed to be clear and functional as to keep focus on producing results from the application.

## Software System Design

### Client

The role of the client application is to run tests and get results from the edge and the data centre. It is a requirement of the system to provide an application that allows a user to interact with the edge node and data centre and to display data to the user. This will be achieved by providing an Electron [10] application that will contain separate pages for each of the applications running on the edge as to keep everything distantly separate but accessible. The client should be able to make POST requests to the data centre and proxy them through the edge node.

Caching

The caching page will be the first page application page to be implemented. The caching application requires an input area and buttons for navigating back home, executing the request, and clearing the cache. The “Go” button will read the entered URL and attempt to navigate to it. There should be mechanisms in place for handling navigation failures if the URL entered is incorrect. The clear cache button should send a request to the edge application for it to clear the cache and display the results of this operation to the user. The last section of the screen is where the webpage itself will be displayed to the user.

Voice Recognition

A requirement of the voice recognition page is the ability to record your voice and use this recording during experiments. The results area should be split into two distinct sections to clearly differentiate between local processing and remote processing along with buttons to allow the user to execute these commands. The remote side should also have a checkbox to allow the user to indicate to the system if pre-processing should occur on the edge node.

The local processing button should execute a script locally and display the results. The remote processing button should send a POST request to the data centre and account for the pre-processing checkbox. This request will also display results to the user once completed.

Machine Learning

The machine learning page when first launched should contain only a button to navigate home, a textbox, and a login button. The user must enter a username before they can test this system. After this has been entered a request should be sent to the data centre to obtain the user’s previously watched movies and this data can be used to get a recommendation from the edge node. This should happen without any user interaction and they should see the results of the operations displayed. There should be two buttons available to the user, one to watch a movie at random and one to watch the movie that was returned as a recommendation. Each of these buttons should also update the user’s average results with the new movie included and display metrics about the watched movie to the user.

### Edge

The role of the edge applications is to pre-process requests for the user. It is a requirement of the system to pre-process requests quickly before passing on data to the data centre or returning it to the client. Another requirement of the edge applications is the ability to run multiple applications on the same edge node to allow for extensibility. This will be achieved by running the applications in Docker Swarm [12, 13] and hosting them on unique ports which will also fulfil the requirement of easily deploying, controlling, and updating the applications. Node.js [16] will be the web server that will host the code for each edge applications.

Caching

The caching application should be running behind a proxy, this means that the client can send a request to the edge with the intention of it being forwarded elsewhere and the caching application will be able to read this URL and use that to return data to the user.

The caching application will use Redis [17] as the in-memory database to store request data for later retrieval. As the caching application will be using Redis there needs to be a mechanism in place to allow the user to clear the cache, this will be implemented by using the Redis Command Line Interface (CLI) [18]. To make this easier for deployment and to keep with the guidelines of Docker a caching base image will be created that will contain the Redis CLI. This means that all further development can be layered upon this image.

The caching application will have to be able to receive proxied requests and use this information to query the Redis cache. If the data does not exist it will make a request to obtain, store and return this information to the user. If the data already exists in the cache the information will be retrieved and returned directly from the edge application.

A Redis instance should be available on each of the edge nodes and the caching application should access each of these directly and in a fair manner. This is to utilise the available resources and not burden one entry point with requests.

Voice Recognition

When a request is received from the client the voice data should be saved to disk. The file that is saved to disk should be named something unique so that multiple requests can be handled at once. A flag will be checked in the request to check if the application needs to pre-process the request or forward the data to the data centre for processing. If the request needs to be pre-processed the application should spawn a process and execute the voice recognition and forward the pre-processed data to the data centre along with a flag that states the data was pre-processed. If the request does not need to be pre-processed the request should be forwarded to the data centre right away.

It may not be efficient to store the voice data on disk before processing it but the way that the voice recognition library PocketSphinx [19] works is that a file path must be passed as a parameter rather than the data stream itself.

The voice recognition system should implement load balancing. It should be implemented in such a way that even if the flag for pre-processing is set, if the system CPU is currently over the threshold the request will be forwarded to the data centre anyway.

Machine Learning

Upon deployment, the edge machine learning application should request a subset of movies from the data centre to store locally on the edge node. It will later use this data to produce recommendations. The full list of movies will be stored at the data centre.

The machine learning application should only ever receive post requests, the request should contain the user ID and depending on the path that the client posted to the edge application should request a random movie to be watched or a specific one that the user requested. This could be implemented in a reusable way as the requests to the data centre will all take similar formats apart from a few variables being changed.

The main function of the edge machine learning application is to produce recommendations based on the data received from the user. The edge application should never keep any user specific data locally as this would mean it is vulnerable to attack, the data it uses to produce a recommendation is stored at the data centre and the client application. The application will receive an average of all the user’s movies and use this information to return a movie recommendation to the requesting client.

### Data Centre

The role of the data centre as per the requirements stated is to handle the request from the client or the pre-processed request from the edge, for example the client wants voice recognition to occur when they click a button so the fact it was pre-processed before reaching the data centre does not matter to them if they receive the result of the operation.

The data centre will be designed and implemented utilising dependency injection, this will make maintenance, control of dependencies and code reuse easier.

Voice Recognition

There should be one entry point for the voice recognition application. It will receive a POST request from the edge node and detect if the pre-processing has occurred based on a flag. If the data has been pre-processed it can be returned to the edge node and consequently back to the client. If the data is not pre-processed it must be stored and processed before being returned to the edge node. The voice recognition being performed should be the same as that being performed on the edge node.

Machine Learning

The machine learning application must be able to return a subset of the movies to the edge node based on a clustering algorithm. This is to ensure that the edge device contains a broad range of available movies to recommend to the user. The data centre machine learning application must also keep a record of all the movies that the user has watched so that it can be queried by the user and eventually processed to produce a recommendation.

# Implementation and Testing

## Choice of implementation languages and development environments

Client

The client needs to be a less powerful device than the device that will be used as a data centre in this system. It also had to be low priced as to make it accessible, this made the Raspberry Pi 3 [11] computer a good choice [20]. Reasons for this being a good choice are that the Raspberry Pi 3 has a low price, a quad-core ARM CPU, 1GB RAM, and an ethernet port.

Electron [10] was chosen as the framework that would be used to implement the client application. Electron utilises Node.js [16] for the backend and standard HTML, CSS, and JavaScript for the user interface. This allows for the same application to run cross platform as the Electron framework runs in Windows, MacOS and Linux. The client hardware for this project is a Raspberry Pi 3 so it is a requirement that the framework to run on an ARM CPU. As Electron is based on Node.js it allowed for more interaction with the host PC than pure JavaScript, this was necessary as to fulfil the requirement of measuring CPU use and executing the voice recognition script locally.

Edge

The hardware that makes up the 3 edge nodes is 3 Raspberry Pi 3’s, the reason for this is that they are small so that they fit in a small space somewhere between the client and the data centre. They are also cheap so that they are viewed as a viable option to distribute to many different areas, the reason for this is that the clients will be widely distributed and the data centre is usually in one central location so for a client to have an edge node between it and the data centre there needs to be edge nodes distributes across the network. The quad-core CPU and 1GB RAM means that the Raspberry Pi 3 is capable of processing resource intensive requests and hosting multiple applications, which was defined in the requirements.

It is a requirement of the system that the applications on the edge are easily maintainable, to meet this requirement Docker was utilised. Docker is a platform that allows for applications to run in isolated containers. As there are multiple edge nodes it was specifically Docker Swarm that was used as multiple worker nodes could be added to the system and it was possible to write scripts that would build, deploy, and control different aspects of the application. For example, the caching application needed to have a Redis instance on every node and Docker provides a deployment flag, ‘--mode=global’, that can be used for this purpose.

The edge node applications are written in JavaScript and run in a Node.js environment. This was chosen because the client was written in JavaScript and Node.js it was a familiar language. As the client application also used Node.js is meant that there were transferable skills and there was a possibility of code reuse such as the module that was used to record an average CPU use.

Data Centre

It’s a requirement of the data centre that it is more powerful than the edge node, this is partly because in some scenarios, such as the voice recognition application, the data centre had to be able to process requests that the edge was not capable of processing. The data centre was written using WebAPI [14] and hosted in IIS [15] as this was readily available in a Windows development environment.

## How each component was implemented

### Client

The first task associated with setting up the client was the setup and configuration required for the client Raspberry Pi. Raspbian Jessie [21] was chosen as the operating system (OS) as it is designed and optimised for the Raspberry Pi. To make development of the client application easy XRDP [22] was installed which allows for a remote desktop connection from a development PC. NPM [23] was used to install the Electron framework to allow the client application to run.

Caching

A problem that needed solved for the caching application was recording the total time that the request took. This was accomplished by attaching event handlers to the web view element that would start and stop a custom stopwatch object and display the total time for the entire request to the user. There is also an event handler attached for when requests fail altogether.

The first iteration of the caching application passed the user requested URL as a query string to the edge node which meant that the client code had to account for this to keep it hidden from the user. This was then updated when the edge application was developed further to include a proxy so that the request could be made directly but the Electron application would proxy the request through the edge node.

Voice Recognition

The first issue that needed to be solved for voice recognition to occur was for audio to be recorded on the client. Investigation into how audio was handled on Linux systems and in JavaScript applications was done. Once the microphone was setup on the client device a third-party library [24] was used to capture audio using JavaScript. This recording had to be in a specific format so that PocketSphinx [19] could use it. When the user finishes recording their voice the audio file is saved to disk, this recording is then used when the user requests voice recognition to occur, whether locally or remotely.

To perform local voice recognition, the renderer thread, which can be viewed as the client thread that would normally execute JavaScript in a browser, needs to communicate with the main thread, which can be viewed as the server thread that would normally exist in a Node.js server. This happens through a communication channel called IPC [25]. The renderer thread sends a command to execute the voice recognition as it does not possess the capabilities to do so itself. Both the renderer thread and the main thread have custom event handlers attached for receiving requests and responses within this IPC channel. Once the main thread has received this request it spawns a new process, executes the voice recognition script, and waits for the response. The main thread will then send this response back to the original sender, which is the renderer thread, that then displays it to the user.

If the user decides they want remote voice recognition to occur they can select if the request should be pre-processed or not, what this does is set a custom header on the request that the edge node will receive.

Machine learning



Figure – First machine learning page

The machine learning page went through two iterations, the first page had buttons that allowed a user to watch a movie of that genre as shown in Figure 9 however this did not show off the full capabilities of this system as well as it should have so the system was updated to perform more advanced recommendations and the UI was updated to what was displayed in **Error! Reference source not found.**.

The system communicates by passing JSON [26] objects with required information such as results, averages of movies watched, and recommendations. The client will retrieve and store all the users previously watched movies when they login directly from the data centre.

Load Balancing

The load balancing application is used to test the voice recognition system with a multitude of requests, this is achieved by reading the value that the user has entered and creating a thread per request, in JavaScript this means executing a method in 0 milliseconds repeatedly so that they all run concurrently and ensuring that the reading of the voice data is done asynchronously as to not block any calls. The result of each request and an average of the all the requests is then displayed to the user.

### Edge

Figure – All applications deployed

Figure – Unfair Redis deployment

Figure – Fair Redis deployment

Figure – All Applications Deployed

Figure – Unfair Redis Deployment

Figure – Fair Redis Deployment

The first step in setting up the edge environment is preparing the Raspberry Pi’s. First the 3 edge nodes had Raspbian OS installed on them and had the graphical user interface turned off and the SSH server enabled so the edge nodes were only controllable though the command line interface and it was possible to do this remotely. Docker was first installed on the first edge pi (EdgePi01) as this would become the manager node, a Docker Swarm was initialised and after Docker was installed on the other two edge nodes they were added to the swarm as worker nodes. This can be seen at the top of Figure 11 where ‘EdgePi01’ is the manager and ‘EdgePi02’ is a worker node alongside ‘EdgePi03’. After Docker was installed the commands to manage containers and images were learnt so that it was possible to administer the system correctly.

For the edge to handle the hosting of multiple applications the deployment scripts for each application ensured to use different internal and external port numbers. This means that the same node can host multiple applications and Docker handles the direction of traffic to where the application is hosted within its network layer.

To quickly view what containers were running on what machine a visualizer [27] was used and can be seen with all the applications deployed in Figure 11.

By default, when an image is pulled into the Docker environment it will try to pull it from the Docker Hub [28] which is a central store of Docker images. For this system, the images should be stored locally, what this meant for setup of the development environment was the creation of a private registry. Unfortunately, the official Docker registry [29] is built to run on x86 processors and the Raspberry Pi 3 that will be hosting the private registry would be running an ARM processor. An alternative repository image was found [30] and once it was confirmed to function as expected the applications could be tested throughout the swarm.

For each application, there is a quick deploy script and a launch script. The purpose of the quick deploy script is to build the application using the specified Dockerfile and add all the files that the application needs to run. This script then tags that new build with the private registry name and the name of the application and pushes the new build to the private registry. The purpose of this is to make the image available to all the nodes in the swarm as otherwise the build would only exist on the manager node where the image was built. If a worker node other than the manager gets a request to launch the application they can search the private registry hosted on the manager and download this image. Finally, the script removes the old service if it exists and deploys the new service. With every build, the previous version of that image is left behind and this quickly adds up when testing multiple builds, they are also left behind on any edge node that previously ran that build. To solve this problem a clean-up script was added so that a command to remove unused builds could be executed on the edge nodes.

Caching

The first iteration of the caching application passed the user requested URL as a query string, which meant that the application read this query string and made a request. This was not a realistic implementation of a caching application as it should not be queried directly, so the application was altered to act as a proxy. This was solved by using the http-proxy library [31] available for Node.js. What this means for the client is that a URL can be requested without knowledge of caching application.

The caching application had some challenges. The application began by utilising a single Redis cache. There is a Redis image [32] available on the Docker Hub so once it was fetched it the command to deploy it needed to be added to the deployment script. However, the caching application needed to be updated because with every webpage request there are many small requests for each item on the page, such as images. This means that when the caching application handled all those requests it also needed to make a separate request to the manager node which would inundate the Docker Swarm manager node with requests. A way to rectify this problem was to deploy a Redis cache to each of the worker nodes so that there were 3 instances up and running, this could be expanded if required. When the deployment was first set to three instances there was a problem where the Redis caches were not guaranteed to be deployed to all three edge nodes as shown in Figure 12, which would mean that the caching application could not rely on being able to contact a Redis cache on all the nodes. This problem was solved by researching into Docker deployment commands and using the “--mode global” command that guarantees that there will be an instance on every node as shown in Figure 13. This means that the caching application can rely on the fact that each worker node has a Redis instance.

Inside the caching application a hashing function was performed on the URL of the data being requested by the client. The purpose of this was to workout which of the 3 Redis caches should be queried directly for the content, as to reduce strain on the manager node.

The caching application must be able to clear the cache across many Redis instances, this was handled by keeping a collection of all the Redis servers the application is currently connected to. This allows the clear cache request to iterate over all of them and spawn a process for remotely clearing the cache on each of the servers using the Redis CLI tools [18].

To implement the caching application a base image was used, this was to adhere to the methodologies associated with Docker images to keep them flexible, lightweight, and reusable [33]. The base image for the caching application was used to install the Redis CLI tools required to interface with the Redis server, this means that all further images that are based of this image would can communicate with Redis servers. Another benefit of using this base layer was that it made deployment times quicker as it did not have to install the tools with every new build.

Voice Recognition

The voice recognition application made use of a base image in a similar way to the caching application. The voice recognition base image has PocketSphinx [19] installed, including all the libraries required for it to work, as it was time consuming and unnecessary to install this with each new build.

The voice recognition application runs within a proxy, this is so the client application can appear to make a request directly to the data centre and it is intercepted and processed if required.

The voice recognition application was required to execute the voice recognition script when requested. A process was spawned to execute the custom bash script that takes a filename as a parameter, the reason for the parameter is because the file has a custom name with each request and the script can be executed on multiple threads concurrently. PocketSphinx required some setup to perform voice recognition, the system should be given a dictionary and language model produced from the Sphinx Knowledge Base [34]. The request then waits on the results of the script to be written to the standard output stream and sends this result on to the Data Centre.

Saving the file locally on disk and making it possible to handle multiple requests concurrently was solved by appending a GUID to the end of the filename. Another aspect of the request that caused concurrency issues with multiple requests was the synchronous calls to the File System [35]. The solution to this was to perform the call asynchronously.

One of the metrics that needed to be recorded for this application was the CPU use, to record this a CPU library [36] was altered and adapted into a Node.js custom module. What this means is that it can be required at the stop of the server in the same fashion as the installed Node.js modules. This allowed the custom module to be reused in the client application for recording local CPU use.

If this application was sent many requests concurrently from the load balancing application it slowed down as executing the voice recognition is CPU intensive. To solve this problem load balancing was implemented. This means that even if the server is requested to pre-process the request, if the current CPU use is over the threshold the request will be forwarded to the Data Centre.

Machine Learning

The machine learning application on the edge went through multiple iterations in the same way as the client. The first iteration was going to produce a recommendation based on the probability of movie type they watched most frequently. The application was then updated to produce a recommendation by performing a more complex calculation discussed below.

The machine learning application would have to deal with 3 different requests, the first is a request to create a recommendation, the last two are requests to watch a specific movie or a random movie.

When the edge application is first launched, it requests a subset of 10,000 movies from the data centre and stores them locally to do evaluation with. During a recommendation request the client will send an average of their previously watched movies which will be used to find the 3 nearest neighbours, the basis for this evaluation was found online [37] and altered to produce an evaluation based on a movie object. Once the 3 nearest neighbours have been found the first one is returned to the client. During a request to watch a movie the edge node will make a request to the data centre so that the full list of movies can be queries. Once the movie has been obtained from the datacentre it is added to the users existing movie average and a recommendation is generated based on this data. If the recommended movie is the one the user just watched the second movie is chosen as the recommendation. The movie watched and updated recommendation are then returned to the client.

### Data Centre

For the data centre to handle requests for different applications multiple endpoints were added to the same API which allowed for distinct separation within the same application. Dependency injection meant that the distinction between the two endpoints was very clear during implementation.

To get the WebAPI deployed a publish profile needed to be made so that the project could be hosted in IIS, this was unique to both the PC and Laptop that the application would be deployed to at different stages.

Voice Recognition

The voice recognition endpoint receives post data and this data can potentially be pre-processed. If the request received is pre-processed the data is read and returned to the user. The purpose of the data centre receiving this request is so that in the future if this application needed to be extended to do any processing on the user’s voice request all the data is available, for the purposes of this project it is enough to show that the voice request has been received and the data is valid. If the data is not pre-processed voice recognition must be performed. This was slightly more challenging on the data centre as PocketSphinx can only be built from source and it was missing dependencies that it required to build in Visual Studio 2015. There was a project available from source [38] that had a built version of the exe available so this was utilised once verified. The next challenge was learning how to execute a process from within WebAPI and receive the output, once achieved the users voice data can be handled in aa similar way to the pre-processed data.

Some of the metrics that need to be returned from this request are network utilisation details, this is the size of the received and sent request, and CPU [39] use of the Data Centre.

Machine Learning

The first part of creating the data centre machine learning application was creating 1,000,000 movies as test data to produce evaluations with. Once these were generated and stored a k-means [40] evaluation was performed across all of them. This evaluation had to be customised to produce clusters based on movie objects. As there were 10,000 movies that needed to be returned as a subset to the edge node 10 clusters were created from the 1,000,000 movies. This meant that 1000 movies from each of the 10 clusters were chosen to be returned to the edge, this would allow for a wide range of available movies to be selected during recommendation evaluation on the edge.

There were problems when implementing the k-means evaluation as it took over 25 minutes to execute the request as it performed 474 loops of clustering over the 1,000,000 movies to produce the 10 required clusters of movies. The solution to this was to store the clustered data once it has been produced so further requests for the clustered data can return this information directly and if movies are added the k-means evaluation can be run over the updated collection of movies.

The data centre is also responsible for storing user’s view data so that it can be retrieved by clients once the user has logged in to the client application, the purpose of this is so that the edge node never stores user specific data. There are also endpoints on the data centre for returning a random movie and returning a movie with a specific ID.

## Use of software libraries

* Docker [12]
  + Docker ARM registry [30]
  + Docker ARM visualizer [27]
* Electron [10]
* PocketSphinx [19]
* Node.js [16]
  + Node.js proxy [31]
  + NodeRedis [41]
  + Request [42]
* Recorder.js [25]
* JSON.Net [43]
* Redis [17]
  + Redis command line interface [18]
* KMeans [40]
* Nearest neighbour [37]
* WebAPI [14]

## Discussion of test approach

As the client and the edge node both utilised Node.js meant that it would have proved more difficult to write unit tests for than standard JavaScript and as the nature of the code is to make web requests to different services there were few areas that needed unit tested in isolation. A unit testing library QUnit [44] was used but would throw an exception when the first Node.js keyword, ‘require’, was found. For these reasons a manual testing approach was taken as this would be adequate for the system because it is for research purposes and designed to produce test results. Unit tests in WebAPI also proved difficult as a lot of the code deals with actual requests so these would be tested manually along with the rest of the system.

A lot of the tests used a happy path approach, the reason for this is that the only interaction with the edge node and the data centre is through the client application which allows for very tight control over the data sent. However, anything that the user can do in the client application should be tested.

For the caching application, there were a total of 15 tests with a 100% pass rate. For the voice recognition application, there was a total of 29 tests. The reason for this is that there is more user interaction for this application so there are more scenarios that need to be tested. There was also a 100% pass rate for this application. For the machine learning application, there were a total of 26 tests with a 100% pass rate. The full testing suite can be found in the appendix.

# System Evaluation

## Machine learning application

Figure – Recommendation after one movie

Figure – Recommendation after 10 movies

The machine learning application lends itself well to an evaluation as a concept. When the user first launches the application, they are told to enter a username. Currently the username does not already have to exist though registration could be added. If the user is new to the system the only option they have is to watch a random movie however if this was to be productised they could have a selection of popular movies or a selection of movies from each of the clusters produced from the data centre as to display a large range. If the user has already watched movies when they login they are presented with an average of the movies they watched that were retrieved from the data centre and a recommendation generated from the edge node based on this average.

Upon deployment, the edge node requests a subset of movies from the data centre. This subset is generated using K-Means clustering. The data is split into 10 clusters and as 10,000 movies are returned to the edge node, 1000 movies from each cluster are selected to be returned. This means that the edge node has a wide range of movies available for selection as recommendations.

As shown in Figure 17 this system can produce a recommendation that is close to what the user watched. It can do this even if the user has watched multiple movies (Figure 18) by taking an average of what the user likes and using that to generate the recommendation. It is possible to see in both cases that the movies are very close in most metrics.

The concept of implementing machine learning in a 3-tier network whilst utilising edge computing to produce recommendations can be further explored by continuing development of the system and producing a quantifiable analysis of the accuracy of the predictions. The number of clusters produced at the data centre and the number of movies returned to the data centre could be altered to evaluate the accuracy of the prediction. If 1,000,000 movies are stored at the edge it is likely to produce a better prediction but the computational power required to parse through all available movies will likely cause the request to take too long and if there are only 100 movies available, 10 from each cluster, the processing is likely to be very quick but the accuracy of the prediction will be diminished as a result.

## Voice recognition and caching applications

These applications will be evaluated though the experimentation below.

# Experimentation



Figure – Experimentation Network Diagram

The network setup used to run these experiments is depicted in Figure 18, the edge nodes and client are all attached to the same router and this means that there is very low latency between them as this is a requirement of the system.

To run these experiments fairly the possibility of human error needs to be reduced. To accomplish this goal automated running of the experiment was implemented where possible.

## Does utilising edge computing reduce the latency of requests for the client?

**Setup;**

* Connect the Raspberry Pi’s as shown in Figure 18
* Run the caching application on the client device
* Deploy the caching service to one of the edge nodes
* Deploy a Redis instance to each of the edge nodes

**Isolate Variables;**

Constants:

* Interface connecting Raspberry Pi’s to the router
* The caching application
* The other applications deployed to the edge nodes

Variables:

* Whether a webpage is cached or not
* If the caching application is being used
* The webpage being loaded

**Method;**

Measurements recorded will be the time between when the WebView element in the client application starts a request and ends a request as this is the value that will matter most to the user.

Recording network use for this application without special hardware or software could present an unfair reading on the request time as this data would have to be recorded at the edge node and returned to the client device. The mechanism to perform this task that would be implemented in software would hinder the accuracy of the results and may not be feasible given the lack of control over the WebView element.

The next part has been automated per run, a single run of the experimentation can occur with no human interaction. After the caching service and Redis service have been deployed, 10 warmup requests will be performed without wiping the cache. After the warmup has completed the cache will be cleared before recording the first response time. A second request will be made and recorded when the webpage has already been cached, and then the cache will be cleared again. Per run of the experiment there will be 20 requests in total made up of 10 sets of 2, a request without the webpage cached and a request with the webpage cached. The cache will be cleared in-between each set.

There will be requests made to 3 different websites to test differences in the benefits of caching requests across different webpages. The 3 webpages will be;

1. http://www.bbc.co.uk
2. http://news.sky.com
3. http://a.singlediv.com

The reason for choosing these webpages is that the first two are sites that are static and frequently accessed so seeing how they react to caching is important. The last site is a static website that might be accessed infrequently so seeing the effects of caching on this site would be useful.

There will be 5 runs of each experiment, with the caching application and Redis application being redeployed in-between each run, and an average of the request times will be taken.

**Hypothesis;**

Webpages that are cached will be returned to the client quicker as the latency between the client and the data centre should be reduced in this setup.

**Results;**

Figure – Caching request times chart

The raw results from the 5 runs of experimentation are available in the appendix.

**Analyse;**

The difference in the “BBC” request time when comparing a cached request with a non-cached request is an improvement of 30% however the effects of adding caching means a 26% increase in request times when comparing a request with no caching to a cached request. There are similar results when comparing the caching application results for the “Sky News” request. There is a 34% improvement between a request that has not yet been cached and a request that has been cached, however by adding the caching application there is less pronounced although still detrimental effect of just 1% increase in request time. This pattern is found a third time when analysis is performed on the last request to “A Single Div”, there is an improvement of 18% when comparing the times of a non-cached request with a cached request however the difference between a request not utilising cache caching and a cached request is an increase of 20% in request time.

After 5 rounds, it is clear that the results follow a pattern as the request time improved when reading the request from Redis rather than making an external request by an average of 27% however the cost of this caching increased the request time of even a cached request by an average of 15%. This means that the cost of caching is enough to offset the benefits gained and ends up having a detrimental effect to the end users. An area for further research would be recording the network use during these requests to see if there are use cases where this type of caching would be worthwhile such as areas where the price of internet is high so the increase in request time would be worth the decrease in internet costs.

The average standard deviation for each round of requests was quite small apart from the first request to “BBC” and “Sky News”, the first request to the “A Single Div” site had a small standard deviation when compared to the other sites. This means that there is some variation between the time taken to make a request from the edge node and return it to the requesting client however once the request is cached it is quite consistent with the amount of time taken to retrieve and return the data.

There is always a cost associated with caching as more execution time is required to query the cache however the benefits of returning a cached item should out-weigh the cost of caching the request as in an ideal system the cached data will be requested many more times than the data that is not requested.

**Conclusion;**

The hypothesis for this experiment was correct to an extent however it was unexpected that removing the caching altogether would have the lowest request time. More investigation could be done with a slower connection speed. It could be argued that the connection speed to the edge will not differ much as they should be physically close to one another but if a website responded slowly or the connection to the internet was slow it could make the caching application more effective.

This was not the expected outcome from the experiment and more investigation could be done to see if there are conditions that could be met to make the application worthwhile to implement.

## Voice recognition application experimentation

This same method will be used to answer the research questions below.

**Setup;**

* Connect the Raspberry Pi’s as shown in Figure 18
* Run the voice recognition application on the client device
* Save a voice recording using the client application
* Deploy the voice recognition service to one of the edge nodes
* Publish the data centre WebAPI with the voice recognition endpoint

**Isolate Variables;**

Constants:

* The voice recording
* The language model and dictionary being used for the processing
* The other applications deployed to the edge nodes

Variables:

* Where the voice processing is occurring

The variable in this experiment is whether the request is processed locally on the client device, remotely pre-processed on the edge, or remotely processed at the data centre.

**Method;**

Measurements recorded for both local and remote requests will be:

* Request time in seconds
* CPU load as a percentage of the total CPU on the client device
* RAM use of the client device in GB
* File size of recording in bytes

Additional metrics recorded for remote requests will be:

* CPU load as a percentage of the total CPU on the edge node
* CPU load as a percentage of the total CPU on the data centre
* RAM use of the edge node in GB
* RAM use of the data centre in GB
* Quantity of data received at the data centre in bytes
* Quantity of data sent from the data centre in bytes

The knowledge base that the voice recognition systems were trained with are all identical and produced using the Sphinx knowledge base tool [34]. This keeps the processing consistent across all the applications.

After the voice recognition application has been deployed, 10 warmup requests will be performed on client device, a further 10 will be performed on the edge node and a final 10 on the data centre.

A run of the experiment consists of 10 requests being handled locally, 10 requests being handled without pre-processing and 10 requests being pre-processed. A single run has been automated for each of the three different request types.

The measure of time will be recorded on the client device from as soon as the request starts to when the request finishes. The same voice recording will be used across all three systems as to ensure consistency.

There will be 5 runs of the experiment in total and an average used during analysis.

### Does utilising edge computing reduce the computational load on the client?

**Hypothesis;**

Introducing edge computing should reduce the CPU strain on the client device regardless of if the request is pre-processed on the edge device or not.

**Results;**

Figure – Client CPU use chart

The raw results from the 5 runs of experimentation are available in the appendix.

**Analyse;**

It is clear to see that the introduction of edge computing dramatically decreases the strain on the client CPU as was expected from the hypothesis. There is an 80% reduction in CPU use between local processing and a remote request that is not pre-processed and a reduction of 89% between local execution and a remote request that is pre-processed.

The standard deviation of 2.26 and 1.61 from the two remote requests can be explained by the first request being the most computationally demanding. The reason for this is because of the way the experimental automation is designed. The requests are all queued up and then executed when they were requested to, the requests are all being queued as the first request begins to execute which causes the first CPU measurement to be higher.

Accurate measurements of CPU use from within a C# application is difficult [33] which explains the high standard deviation numbers (4.96 and 16.91) for the data centre CPU, this means that it would not be sensible to use these numbers during evaluation of the system. It seems to be that to accomplish a higher level of accuracy for this CPU reading in the WebAPI project there needs to be a second delay [45] in the request but this would skew the time that is being measured. This is the reason that an accurate comparison of the CPU in the data centre is not possible in this system.

**Conclusion;**

In this experiment, the results that were obtained matched the hypothesis set out. With the introduction of edge computing the strain on the client device is lessened.

### Does utilising edge computing increase the latency of the request for the client?

**Hypothesis;**

The latency of the request should not increase much if the request is pre-processed on the edge device as both devices are Raspberry Pi 3’s however the request should be processed much quicker at the data centre as it has a faster processor although this means that the data centre is performing more work.

**Results;**

Figure – Request execution time chart

The raw results from the 5 runs of experimentation are available in the appendix.

**Analyse;**

There is a decrease in request time of 42% between local execution and a remote request without pre-processing and an increase of 19% between the local execution and a remote request with pre-processing. This increase of 0.29 seconds can be argued as a worthy trade-off when Figure 20 is evaluated to discover the 89% reduction in client CPU utilisation.

The 42% decrease (0.45 seconds) in request time caused by using the more powerful datacentre for voice recognition might not be viewed as worthwhile when there are many requests getting sent to the data centre for processing. The implementation of the voice recognition application at multiple edge computing nodes would help reduce the strain placed on the data centre and only slightly increase request time for the client.

The analysis of the data centre doing more work is difficult to quantify in this system as the CPU measurement of the WebAPI application is difficult to obtain [45] so the results for this metric can be viewed as inaccurate.

**Conclusion;**

The first part of the hypothesis was that the request time should not increase much if the request is pre-processed on the edge device compared to the client device as both are identical machines. As shown above there is a 19% increase in request time but this only amounts to 0.29 seconds which is acceptable as the time taken to send data over the network is accounted for. The second part of the hypothesis is that the request should be processed much quicker at the data centre which can be seen to be true as it is the fastest of the three evaluations and 42% quicker than processing the request on the client alone.

There are two trade-offs to be evaluated in this experiment the result of which will be unique to each implementation of the application. The first is the increase in request time when comparing local execution to remote execution against the decrease in CPU use on the client device. The second is the increase in latency when comparing pre-processing on the edge node to processing the request at the data centre against the increase in request time for the client.

### Does implementing edge computing reduce the network utilisation?

**Hypothesis;**

Utilising edge computing will reduce the use of the network as less data needs to be sent to the data centre for processing, the pre-processed data is much smaller than the raw data, which in this scenario is a voice recording.

**Results;**

Figure – Network utilisation chart

The raw results from the 5 runs of experimentation are available in the appendix.

**Analyse;**

There is a 3075538% reduction in data sent to the data centre when pre-processing occurs on the edge. This is a reduction of 246043 bytes as the data being sent to the data centre is no longer the voice recording, it is just the 8 bytes of data that make up the text spoken.

There will not be a direct correlation between size of data received and data returned, this is because two recordings that say the exact same thing could be different in size. The recording must be in a specified format with a specific bitrate so the quality of the recording will not change but the amount of silence recorded or the speed of speech could affect the length of the recording which would affect file size which in turn must be sent over the network.

There is no difference in bytes sent from the data centre so even if a larger recording that says the same thing as a smaller recording is received the bytes returned from the data centre should be identical. There is a small difference between the bytes sent from the edge and the bytes sent from the data centre, the reason for this is whitespace at the end of the text being returned from PocketSphinx that is trimmed at the data centre.

**Conclusion;**

This is quite straightforward as it is clear the voice recording is only sent as far as the data centre and then the data required to be sent to the data centre from the edge node is dramatically decreased. This confirms the hypothesis as there is a large reduction in data sent from the edge to the data centre.

## Load balance application experimentation

**Setup;**

* Connect the Raspberry Pi’s as shown in Figure 18
* Run the load balancing application on the client device
* Deploy the voice recognition service to one of the edge nodes
* Publish the data centre WebAPI with the voice recognition endpoint

**Isolate Variables;**

Constants:

* The voice recording
* The software package being used
* The language model and dictionary being used for processing
* The other applications deployed to the edge nodes

Variables:

* Number of requests
* Whether the deployed edge node service is utilising custom load balancing
* The number of instances of the voice recognition application

**Method;**

Measurements recorded will be:

* Processing time in seconds for all requests to finish
* Average request time in seconds
* Average CPU load as a percentage of the total CPU on the client device
* Average CPU load as a percentage of the total CPU on the edge node
* Average CPU load as a percentage of the total CPU on the edge node
* RAM use of the client device in GB
* RAM use of the edge node in GB
* RAM use of the data centre in GB

The same voice recording will be used throughout the experiment to allow for a fair comparison of computational load. The knowledge base that the voice recognition systems were trained with are all identical and produced using the Sphinx knowledge base tool [34]. This keeps the processing consistent across all the applications.

There are 3 different types of requests in this experiment. The first is a request with no pre-processing, the second is a request with pre-processing and no load balancing, and the final is a request with pre-processing and load balancing.

The first run of the experiment will be the edge application deployed without load balancing implemented. There will be 10 sets of 10 warmup requests executed without pre-processing and 10 sets of 10 more requests with pre-processing. For a single run of the experiment, 10 sets of 10 requests will occur without pre-processing then a further 10 sets of 10 requests with pre-processing. There will be 5 runs of the experiment executed altogether.

The edge application will then be deployed with load balancing enabled, and after 10 sets of 10 warmup requests a further 5 runs of 10 sets of 10 requests to pre-process the data made.

The 3 different types of request, no pre-processing, pre-processing without load balancing, and pre-processing with load balancing will be made again but with 5 runs of 10 sets of 20 requests to see the comparison of a larger number of concurrent requests.

### Does implementing load balancing improve the effectiveness of edge computing?

**Hypothesis;**

Adding load balancing will improve response times with large number of requests as if the edge CPU utilisation is over 70% the request will be redirected to the data centre. The strain on the edge CPU should also be lessened by introducing load balancing.

**Results;**

Figure – Request time chart

Figure – Edge node CPU use chart

The raw results from the 5 runs of experimentation are available in the appendix.

**Analyse;**

There is no improvement on request time from implementing load balancing when there are just 10 concurrent requests. The request time for pre-processing without load balancing and the request time for pre-processing with load balancing stay the same to 0.006 of a second, what can be inferred from this information is that the edge server is capable of processing all these requests at once and has no use for the implemented load balancing. However, the improvement can be seen when the number of concurrent requests is increased to 30 as there is a 61% reduction in request time when load balancing is implemented, this means that the edge application has exhausted its resources and needs to offload some execution of the voice recognition to the data centre which is also obvious to see when the edge application logs are investigated.

The edge CPU utilisation can be seen to decrease by 1% and 2% during the sets of 10 and sets of 30 requests respectively, this shows that implementing load balancing does improve performance slightly. The CPU is never shown to reach 100% during the experimentation however this can be attributed to the requests all being received at once and maxing the CPU during voice recognition but then easing up with the less resource intensive POST request to the data centre that occurs after pre-processing. The CPU measurement is taken after the data centre request has occurred as this is the end of the processing that occurs on the edge.

When the CPU measurements are viewed alongside the request time measurement it is clear to see that the implementation of load balancing improves the request time and reduces the CPU utilisation of the edge device. What this means is that some strain is removed from the data centre, it is difficult to show this as per the problems with recording data centre CPU use [45] but it is clear to see that the voice recognition is occurring on the edge node so it follows that less processing is occurring at the data centre. If there are too many requests received on the edge the computation is offloaded to the data centre so that the user on the client does not see a larger delay in processing time.

It should be mentioned that when the edge application is under stress and not implementing load balancing that PocketSphinx occasionally returns empty text as the voice result. Implementing load balancing causes this to happen less frequently. When debugging the system, it seems that everything executed correctly and as expected apart from the voice recognition system returning no text and it only seems to happen when the processor is at max capacity.

**The hypothesis for this experiment was that adding load balancing would improve the response times with a larger number of requests. This is correct as when 30 requests were executed there was a** 61% increase in response time which could not be observed when just 10 requests were executed. The second part of the hypothesis was that the strain on the CPU would be lessened by the introduction of load balancing which was observed only to a small degree of 1% and 2% which was not to the degree that was expected.

**Conclusion;**

There is a decrease in request time to the user which matches the hypothesis and will improve the user experience of the system, however as there is only a small increase in edge CPU use on the edge it must mean that the extra processing that is responsible for improving response times is coming from the data centre which means less of the strain is being removed.

### Does deploying more than one instance of the application improve the effectiveness of edge computing?

**Method;**

There is a slight change in method for this experiment as the application with no load balancing is deployed with 3 instances running on each node instead of 1.

**Hypothesis;**

The response time will be closer to that of the 10 requests run as there are 3 instances of the application running across 3 different nodes. This means that there are 3 times the resources available to process the requests which should reduce the strain on each individual node.

**Results;**

Figure – Request time chart using 30 requests and no load balancing

Figure – Edge CPU use using 30 requests and no load balancing

The raw results from the 5 runs of experimentation are available in the appendix.

**Analyse;**

It is clear to see that scaling this application has dramatic improvements. The voice recognition application is stateless [46], which means that it does not record information about the requesting client to alter processing during the next request. This allows for the application to be scaled.

The average request time was improved by 121% when the application was scaled to 3 instances. There was also a 27% improvement in CPU use on the edge node, and as load balancing is not implemented when comparing these numbers all the requests were pre-processed on the edge. What this means is that there are dramatic improvements when the application is scaled.

The average request time for 10 requests on a single instance is 5.39s, the average request time for 30 requests on 3 instances is 6.82s which compared to the 15.06s it would take to process 30 requests on 1 instance is a vast improvement. In theory with a fair load balancer there will still be 10 requests per instance which is why the request times are close to one another.

**Conclusion;**

The benefits of scaling the application are very clear and match what was set out in the hypothesis, however this means that a lot of the processing capabilities of that node are being used by the voice recognition application so it should be verified that other applications can perform optimally while the voice recognition application is under expected load.

# Conclusion

The 3-tier network was implemented as discussed in the introduction and problem specification. There is a client, edge, and data centre where the edge pre-processes or handles requests to ease the computational strain on the other tiers. The system is easily administered and maintainable and the benefits of adding edge computing to different applications has been quantified and analysed such as the effect on response times and computational load. The 3 applications developed for the system are caching, voice recognition and machine learning as per the candidates proposed.

As there are a few different research questions proposed and results obtained to answer these questions so it is useful to discuss the significance of them individually. The first is if utilising edge computing reduces latency of requests for the client, the purpose of this is to discover if the implementation of edge computing could be used to benefit the client and ultimately the end users experience. What was found was that the implementation of the caching application increased the request time even with a cached request which means that the cost of querying the cache outweighs the time benefits of utilising the cache. An experiment performed with a technology called Edge Accelerated Browsing (EAB) [5] found that with some requests the render time would increase however their results show that in most cases implementing EAB decreases render time. These results can be compared as they are both experimenting with web browsing while implementing edge computing solutions however it should be noted that EAB performs processing such as rendering and optimised transmissions whereas the application developed for this system performs caching across multiple instances so they are different approaches to the same objective.

The second research question proposed was if utilising edge computing would reduce the computational load for a client, this is useful to understand as battery life on client devices is becoming more of an issue [47] and by performing less tasks that require a large amount of CPU time the battery life should improve. What was found is that there are drastic improvements to computational load on client devices by offloading resource intensive tasks to edge computing solutions.

After revealing the benefits on client CPU use it was useful to discuss the effects of implementing this solution, so the third research question proposed was if edge computing increased the latency of request for the client device. This would compare the execution time of remote processing vs local processing. The latency of requests for processing occurring remotely has been discussed before [48] however the on-demand gaming platform that was being used to investigate these effects requires much more emphasis on latency as per the nature of multiplayer gaming. What was found in that study is that the latency decreased with the implementation of edge solutions that are closer in geographical proximity to the clients. With the solution implemented here for voice recognition what was found is that a small increase in latency could be justified by the large benefits in CPU use on the client device and can be attributed to the extra steps needed to perform the remote execution.

As the benefits of using remote executing for processing that previously would have occurred on client devices were found, the effects of this remote processing on network utilisation was investigated. This means the comparison of executing the request on an edge node which is geographically closer to the client, than on the data centre which could mean sending data further which has a negative effect on bandwidth [7]. There was seen to be a drastic decrease in data being sent the entire way to the data centre when edge computing was introduced however the data returned from the data centre was not affected as the result of the operation still needed to be returned. Benefits in decreasing bandwidth use such as cost [4] are achievable by implementing edge computing.

The scalability of edge computing had to be investigated to verify that the solution implemented is scalable to handle multiple requests from users. The resource intensive task of voice recognition had a large decrease in request time when load balancing was implemented however this means that the data centre is handling some of the requests which removes the benefits gained from edge computing such as network utilisation. A solution to this was to scale the application to have multiple instances running concurrently on multiple edge nodes. Both these approaches to load balancing produced positive results which is what was expected.

Raspberry Pi is a cost-effective way to learn about the importance of performance as it is a low powered device when compared to the data centre. Docker had no issues running on the ARM processor of the Raspberry Pi and the benefits of edge computing were apparent which agrees with the results discussed in a paper discussing the performance of a low power Raspberry Pi cloud [20]. The system provided enough power to run multiple instances of Node.js servers in parallel. There were a few networking challenges with this system but once the system is setup it is easy to administer and control.

The conclusions met for the research questions proposed are significant as they show that there is potential for edge computing applications to benefit the client and the data centre that can be deployed to a geographically wide area on hardware that is affordable and easily maintainable.

Development of this system was continuous throughout the year, there were short term goals that were met so that the larger goals, such as completing one of the applications in its entirety, were achieved. The system was easy to break into smaller components because of its nature, there are a distinct section for the client, edge, and data centre for each of the 3 applications developed. One of the main learning points for project management was to clearly define what the application should be and ensure that all members involved are clear before starting development, though this can also be difficult to achieve because of changing requirements as the system under development takes shape. An example of this is the machine learning application as once development was nearing completion it was clear that the system was not showing its full potential so it was adapted and redesigned so that the resulting application was better than what was laid out at the start.

It would be useful to see experimental results of the machine learning application to see quantifiably what benefits it provides however it is useful to show the different ways that edge computing can be implemented. The first way shown is a request being handled entirely at the edge with the caching application without the data centre having to be queried. The second way is having the option of doing the processing/pre-processing at any of the 3 tiers in the network with the voice recognition application. The final way shown is the tiers working in conjunction with one another to produce a result with the machine learning application. It shows the flexibility of such a system depending on what the requirements are, such as reducing strain on the data centre or the client.

This project has great potential to be taken further, the machine learning application has some very interesting concepts that could be expanded upon for further research such as more complex problems being solved by utilising multiple edges. The caching and voice recognition applications quantifiably show the benefits of edge computing and could be developed further into real world robust systems to be analysed.

Overall the results show that it is not always beneficial to use edge computing depending on the individual use cases however there are benefits that can be obtained if it is utilised correctly and the trade-offs evaluated.

# References

[1] C. Dickson, “Connor Dickson / FinalYearProject · GitLab.” [Online]. Available: https://gitlab.eeecs.qub.ac.uk/40103631/FinalYearProject.

[2] L. M. Vaquero and L. Rodero-Merino, “Finding your Way in the Fog,” ACM SIGCOMM Comput. Commun. Rev., vol. 44, no. 5, pp. 27–32, Oct. 2014.

[3] A. Espenson, “Is Fog Computing the Future? - Business.com.” [Online]. Available: https://www.business.com/articles/what-is-fog-computing-the-low-down-on-cloud-computings-newest-iteration/.

[4] W. Shi and S. Dustdar, “The Promise of Edge Computing,” Computer (Long. Beach. Calif)., vol. 49, no. 5, pp. 78–81, May 2016.

[5] N. Takahashi, H. Tanaka, and R. Kawamura, “Analysis of Process Assignment in Multi-tier mobile Cloud Computing and Application to Edge Accelerated Web Browsing,” in 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 2015, pp. 233–234.

[6] R. Banta, “Why Edge Computing is Gaining Popularity Along with IoT | Lifeline Data Centers.” [Online]. Available: https://lifelinedatacenters.com/data-center-strategy/edge-computing-gaining-popularity/.

[7] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The Case for VM-Based Cloudlets in Mobile Computing,” IEEE Pervasive Comput., vol. 8, no. 4, pp. 14–23, Oct. 2009.

[8] M. Patel et al., “Mobile-Edge Computing,” 2014.

[9] D. LeClair, “The Edge of Computing: It’s Not All About the Cloud.” [Online]. Available: http://insights.wired.com/profiles/blogs/the-edge-of-computing-it-s-not-all-about-the-cloud#axzz4dG9ofEqG.

[10] “Electron.” [Online]. Available: https://electron.atom.io/.

[11] “Raspberry Pi 3 Model B.” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[12] “Docker.” [Online]. Available: https://www.docker.com/.

[13] “Docker Swarm mode.” [Online]. Available: https://docs.docker.com/engine/swarm/.

[14] “ASP.NET Web API.” [Online]. Available: https://www.asp.net/web-api.

[15] “Microsoft IIS.” [Online]. Available: https://www.iis.net/overview.

[16] “Node.js.” [Online]. Available: https://nodejs.org/en/.

[17] “Redis.” [Online]. Available: https://redis.io/.

[18] “redis-cli.” [Online]. Available: https://redis.io/topics/rediscli.

[19] “PocketSphinx.” [Online]. Available: https://github.com/cmusphinx/pocketsphinx.

[20] W. Hajji and F. Tso, “Understanding the Performance of Low Power Raspberry Pi Cloud for Big Data,” Electronics, vol. 5, no. 2, p. 29, Jun. 2016.

[21] “Raspbian.” [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/.

[22] “xrdp.” [Online]. Available: http://www.xrdp.org/.

[23] “npm.” [Online]. Available: https://www.npmjs.com/.

[24] “Electron IPC.” [Online]. Available: https://github.com/electron/electron/blob/master/docs/api/ipc-main.md.

[25] “Recorderjs.” [Online]. Available: https://github.com/mattdiamond/Recorderjs.

[26] “JSON.” [Online]. Available: http://www.json.org/.

[27] “alexellis2/visualizer-arm - Docker Hub.” [Online]. Available: https://hub.docker.com/r/alexellis2/visualizer-arm/.

[28] “Docker Hub.” [Online]. Available: https://hub.docker.com/.

[29] “library/registry - Docker Hub.” [Online]. Available: https://hub.docker.com/\_/registry/.

[30] “vdavy/registry-arm - Docker Hub.” [Online]. Available: https://hub.docker.com/r/vdavy/registry-arm/.

[31] “http-proxy.” [Online]. Available: https://www.npmjs.com/package/http-proxy.

[32] “hypriot/rpi-redis - Docker Hub.” [Online]. Available: https://hub.docker.com/r/hypriot/rpi-redis/.

[33] “Understand images, containers, and storage drivers - Docker Documentation.” [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/.

[34] “Sphinx Knowledge Base Tool VERSION 3.” [Online]. Available: http://www.speech.cs.cmu.edu/tools/lmtool-new.html.

[35] “File System | Node.js v7.9.0 Documentation.” [Online]. Available: https://nodejs.org/api/fs.html.

[36] “Calculate current CPU load with Node.js.” [Online]. Available: https://gist.github.com/bag-man/5570809.

[37] “Machine Learning in JS: k-nearest-neighbor.” [Online]. Available: https://www.burakkanber.com/blog/machine-learning-in-js-k-nearest-neighbor-part-1/.

[38] “CMU Sphinx - pocketsphinx.” [Online]. Available: https://sourceforge.net/projects/cmusphinx/files/pocketsphinx/5prealpha/.

[39] “Real-time system resource monitor with SignalR, MVC, Knockout and WebApi | Steve Hobbs.” [Online]. Available: https://stevescodingblog.co.uk/real-time-system-resource-monitor-with-signalr-mvc-knockout-and-webapi/.

[40] “K-Means Data Clustering Using C# -- Visual Studio Magazine.” [Online]. Available: https://visualstudiomagazine.com/Articles/2013/12/01/K-Means-Data-Clustering-Using-C.aspx?admgarea=features&Page=1.

[41] “NPM - Request.” [Online]. Available: https://www.npmjs.com/package/request.

[42] “NodeRedis.” [Online]. Available: https://github.com/NodeRedis/node\_redis.

[43] “Json.NET - Newtonsoft.” [Online]. Available: http://www.newtonsoft.com/json.

[44] “QUnit.” [Online]. Available: https://qunitjs.com/.

[45] “Retrieving Accurate CPU Usage In C#.” [Online]. Available: http://gavindraper.com/2011/03/01/retrieving-accurate-cpu-usage-in-c/.

[46] “Stateless App.” [Online]. Available: http://whatis.techtarget.com/definition/stateless-app.

[47] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, “Saving portable computer battery power through remote process execution,” ACM SIGMOBILE Mob. Comput. Commun. Rev., vol. 2, no. 1, pp. 19–26, Jan. 1998.

[48] S. Choy, B. Wong, S. Gwendal, and C. Rosenberg, “The brewing storm in cloud gaming: a measurement study on cloud to end-user latency,” Proc. 11th Annu. Work. Netw. Syst. Support Games, 2012.

# Appendices

## Caching System Tests

Client Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can reach Caching Application page | Click on the caching application button on the home page | The Caching Application page is now launched | True |
| Home button | Click home button | Should be returned to the home page | True |
| URL bar is present | On the caching page ensure there is a URL bar | The URL bar is present | True |
| Can type into URL bar | Click into the URL bar and type an address | The address typed will show up in the URL bar | True |
| Go button without URL typed | Click the Go button when there is no URL typed out | Nothing, there will be no error | True |
| Go button with an invalid URL typed | Click the Go button when an invalid URL is typed into the URL bar | A message will appear warning the user that the request failed | True |
| Go button with a valid URL typed | Click the Go button when a valid URL is typed | The requested webpage will be fetched and displayed to the user | True |
| The timer | Click the go button with a valid URL | The timer will start and when the round is finished it will record the total time to complete the request | True |
| Clear cache button | Click the clear cache button | Information will appear to indicate that the cache on the edge node was cleared successfully | True |
| Can connect to the proxy | Make a web request | Ensure that the proxy is utilised | True |

Edge Node Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can connect to all the Redis instances | Deploy the web server and check the logs | The server will automatically connect to all the Redis servers | True |
| Can make request to Caching Web Server | Make a curl request to the Edge Node using the host name and port number | Receive Data back | True |
| Clear cache request | Call the Edge Node with a ClearCache request | The clear cache command is executed using the redis-cli and the cache is cleared from all Redis instances. This information is then returned to the user | True |
| Receive new request | Call the Edge Node with a new URL | The content is retrieved, stored in the correct Redis instance, and returned to the client | True |
| Receive a request for the second time | Call the Edge Node with a URL that has already been requested | The content is retrieved from the correct Redis instance and returned to the user | True |

## Voice Recognition System Tests

Client Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can reach the Voice Recognition page | Click on the Voice Recognition button on the home page | The Voice Recognition Application page is now launched | True |
| Home button | Click home button | Should be returned to the home page | True |
| Recording status after page launch | Launch the Voice Recognition application | The recording status should be “Not Recording” | True |
| Recording status after clicking record | Click the record button | The recording status should change to “Recording” | True |
| Recording status after stopping recording | Click the stop recording button after starting a recording | The recording status should change to “Finished recording” | True |
| Recording button when not recording | Stop a recording or don’t start a recording | The recording button should read to “Start Recording” | True |
| Recording button when recording | Start a recording or don’t stop a recording | The recording button should read “Stop Recording” | True |
| Execute local recognition button without recording | Press the Execute Local Recognition button without recording a phrase | The results should say you said nothing | True |
| Execute local recognition button after recording | Press the Execute Local Recognition button after recording a phrase | The execution of the recording should be performed locally | True |
| Execute remote recognition button with no recording | Press the Execute Remote Recognition button without recording a phrase | Error should be shown to user | True |
| Execute remote recognition button after recording without edge processing | Press the Execute Remote Recognition button after recording a phrase and don’t tick the pre-process checkbox | The Execution of the recording should be performed remotely on the Data Centre | True |
| Execute remote recognition button after recording with edge processing | Press the Execute Remote Recognition button after recording a phrase and tick the pre-process checkbox | The Execution of the recording should be performed remotely on the Edge Node and the results sent to the Data Centre | True |
| Local Results section when no local evaluation has occurred | Launch the Voice Recognition Application and don’t perform local evaluation | The results section is empty | True |
| Local Results section when the local evaluation has occurred | Perform local evaluation | The local results section should show statistics | True |
| Remote Results section when no remote evaluation has occurred | Launch the Voice Recognition Application and don’t perform remote evaluation | The results section is empty | True |
| Remote Results section when no remote evaluation has occurred | Perform remote evaluation | The remote results section should show statistics | True |

Edge Node Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can make request to Voice Recognition Web Server | Make a curl request to the Edge Node using the hostname and port number | Receive Data back | True |
| Receive Get Request | Make a get request to the Edge Node | Message about only allowing POST requests | True |
| Receive Put Request | Make a put request to the Edge Node | Message about only allowing POST requests | True |
| Receive post request with invalid data | Make a post request with an invalid audio file or incorrect data | No error and gracefully handled | True |
| Receive post request with valid recording | Make a request with a valid voice recording | A JSON object is returned with information about the request and the audio spoken | True |
| Receive valid request with the pre-process header | Make request with the pre-process request header set | The processing of the voice file happens on the Edge Node | True |
| Receive valid request without the pre-process header | Make request without the pre-process request header set | The voice recording is sent to the data centre for processing and the results returned to the user | True |
| Request information from the Data Centre when the WebAPI is not published | Remove the WebAPI and make a request | The service should record an error | True |
| Request information from the Data Centre when the WebAPI is published | Publish the WebAPI and make a request | The information should be returned correctly | True |
| Measure CPU use | Make a regular request | The CPU use on the edge node should be returned as part of the request | True |
| Measure Time of request | Make a regular request | The Time taken to process the request should be recorded | True |

Data Centre Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Receive valid request to process voice | Make a valid POST request with invalid voice data | The voice is processed correctly and a valid response is produced | True |
| Receive valid request to record information | Make a valid POST request with textual data from a pre-processed request | The request should be handled gracefully and a response sent | True |

## Machine Learning System Tests

Client Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can reach the Machine Learning page | Click on the Machine Learning button on the home page | The Machine Learning Application page is now launched | True |
| Home button | Click home button | Should be returned to the home page | True |
| Username box is present | On the Machine Learning page ensure the username box is visible | The Username box is visible | True |
| Can type into the username box | Click into the username box and type a username | The username typed will show up in the username box | True |
| Login button without username typed | Click the Login button when there is no username typed out | User will not be shown a logged in page and an error will appear | True |
| Login button with a valid username typed | Click the Login button when a valid username is typed | The username box and login button will be replaced with the machine learning application page | True |
| Login as a user who has never watched a movie | Type a username that does not exist in stored user data in the data centre | The previous results section and recommendation section will not be populated | True |
| Login as a user who has previously watched a movie | Type a username that does exist in the stored user data in the data centre | The previous results will contain an average and count of all previously watched movies and a recommendation will be presented | True |
| Watch a random movie | Click the button to watch a random movie | The movie that you just watched will be shown. There will also be an updated recommendation. The average results will also be updated | True |
| Watch a recommended movie when there is no movie recommendation | Click the button to watch the recommended movie when there is no movie recommendation | A warning will be displayed to watch a movie | True |
| Watch a recommended movie when there is a valid recommendation | Click the button to watch the recommended movie when a movie recommendation is available | The movie you have been recommended will be watched and you will receive an updated recommendation | True |

Edge Node Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can make request to Voice Recognition Web Server | Make a curl request to the Edge Node using the hostname and port number | Receive Data back | True |
| Receive Get Request | Make a get request to the Edge Node | No error and message returned | True |
| Receive Put Request | Make a put request to the Edge Node | No error and message returned | True |
| Receive post request with invalid data | Make a post request with incorrect JSON | Error message is returned | True |
| Get movies on launch | Deploy the application | A request is performed to download movies from the Data Centre | True |
| Write received movies to disk | Deploy the application | The received movies from the Data Centre are stored on disk | True |
| Get Recommendations | Make a request to get recommendations | A movie is returned to the client as a recommendation | True |
| Watch Random Movie | Make a request to watch a random movie | The request is forwarded to the Data Centre and a recommendation is produced then all the data is returned to the Client | True |
| Watch Movie | Make a request to watch a specific movie | The request is forwarded to the Data Centre and a recommendation is produced then all the data is returned to the Client | True |
| Produce recommendation | Make a request that will return a recommendation | A recommendation is produced by performing Nearest Neighbour evaluation | True |

Data Centre Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Receive request to GetMovies | Make a GET request for “GetMovies” | The data returned is a subset of the movies available at the data centre | True |
| Receive request to GetPreviousMovies with a UserID that doesn’t exist | Make a POST request for “GetPreviousMovies” with a UserID that does not exist | The system should handle the fact that the user has not watched any movies yet and return nothing | True |
| Receive request to GetPreviousMovies with a UserID that exists | Make a POST request for “GetPreviousMovies” with a UserID that exists | The system should return a collection of the movies the user previously watched | True |
| Receive request to WatchRandomMovie | Make a POST request for “WatchRandomMovie” with a user ID | Watches a movie at random and records the fact the user watched the movie and return this data | True |
| Receive request to WatchMovie | Make a POST request for “WatchMovie” with a user ID and movie ID | Store the fact the user watched this movie and return the movie data. | True |

## Load Balancing System Tests

Client Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can reach the Load Balance page | Click on the Load Balance button on the home page | The Load Balance Application page is now launched | True |
| Home button | Click home button | Should be returned to the home page | True |
| Number of Requests box is present | On the Load Balance page ensure the number of requests box is visible | The number of request box is visible | True |
| The checkbox for pre-processing requests can be selected | There is a checkbox for enabling pre-processing of requests and it can be ticked | The checkbox is visible and can be ticked | True |
| The go button is visible | Check the button for executing the requests is visible | The button for executing the requests is visible | True |
| Press go when nothing is entered into the Number of requests box | Don’t type anything in the “Number of requests” box and press “Go” | An error should appear | True |
| Press go when text is entered | Type a value that is not numerical | An error should appear | True |
| Press go when numbers are entered | Type a value that is numerical, non-negative and less than 50 | The number of requests entered should start executing and display in the results section | True |
| Press go when a number greater than 50 is entered | Type a value that is greater than 50 | An error message should appear warning you to enter a valid number | True |

## Experiment Results

### Caching Results

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run 1 |  |  |  |  |  |  |  |  |  |
| Iteration |  | BBC | |  | Sky News | |  | A Single Div | |
|  | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request |
| 1 | 3.46 | 4.4 | 4.71 | 3.77 | 4.79 | 3.46 | 1.59 | 2.65 | 1.84 |
| 2 | 3.93 | 8.49 | 4.65 | 3.72 | 6.21 | 3.55 | 1.58 | 2.34 | 1.86 |
| 3 | 3.62 | 8.24 | 4.5 | 3.93 | 3.93 | 3.64 | 1.62 | 2.24 | 1.86 |
| 4 | 3.34 | 4.93 | 4.64 | 3.75 | 4.87 | 3.44 | 1.66 | 2.21 | 1.88 |
| 5 | 6.66 | 4.9 | 4.94 | 3.56 | 9.73 | 3.45 | 1.61 | 2.19 | 1.86 |
| 6 | 3.44 | 8.08 | 4.66 | 3.73 | 3.93 | 4.05 | 1.68 | 2.21 | 1.87 |
| 7 | 4.04 | 4.25 | 4.93 | 3.57 | 4.34 | 3.5 | 1.58 | 2.59 | 1.88 |
| 8 | 3.57 | 8.88 | 4.95 | 3.68 | 9.8 | 3.9 | 1.62 | 2.25 | 1.87 |
| 9 | 3.53 | 8.32 | 4.89 | 3.65 | 3.87 | 3.58 | 1.58 | 2.37 | 1.86 |
| 10 | 3.94 | 4.35 | 4.54 | 3.54 | 4.64 | 3.46 | 1.66 | 3.15 | 1.92 |
| Avg | 3.953 | 6.484 | 4.741 | 3.69 | 5.611 | 3.603 | 1.618 | 2.42 | 1.87 |
| Standard Deviation | 0.980623951 | 2.043369766 | 0.171817086 | 0.118133634 | 2.293388807 | 0.20928715 | 0.037357135 | 0.302948474 | 0.021081851 |
| Run 2 |  |  |  |  |  |  |  |  |  |
| Iteration |  | BBC | |  | Sky News | |  | A Single Div | |
|  | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request |
| 1 | 3.63 | 4.67 | 4.57 | 3.84 | 4.22 | 3.5 | 1.63 | 2.51 | 2 |
| 2 | 3.48 | 5.08 | 4.74 | 3.7 | 6.27 | 3.8 | 1.63 | 2.24 | 2.17 |
| 3 | 3.56 | 9 | 4.48 | 3.56 | 4.9 | 4.24 | 1.58 | 2.14 | 2.32 |
| 4 | 3.43 | 4.67 | 5.01 | 3.63 | 4.25 | 3.82 | 1.6 | 2.13 | 2.12 |
| 5 | 3.73 | 9.23 | 4.21 | 3.69 | 4.74 | 3.83 | 1.61 | 2.31 | 2.16 |
| 6 | 3.76 | 4.26 | 4.46 | 3.7 | 8.53 | 3.72 | 1.61 | 1.89 | 2.36 |
| 7 | 3.84 | 4.71 | 4.58 | 3.7 | 3.97 | 3.5 | 1.59 | 2.15 | 2.11 |
| 8 | 3.53 | 9.59 | 4.57 | 3.76 | 12.35 | 3.9 | 1.63 | 2.18 | 2.18 |
| 9 | 3.54 | 4.62 | 4.51 | 3.68 | 4.62 | 3.82 | 1.62 | 3.46 | 2.33 |
| 10 | 3.57 | 4.6 | 4.71 | 3.76 | 4.26 | 3.59 | 1.63 | 2.6 | 2.13 |
| Avg | 3.607 | 6.043 | 4.584 | 3.702 | 5.811 | 3.772 | 1.613 | 2.361 | 2.188 |
| Standard Deviation | 0.131153006 | 2.242092525 | 0.209135788 | 0.075836081 | 2.677488251 | 0.218062173 | 0.018287822 | 0.435340225 | 0.114387256 |
| Run 3 |  |  |  |  |  |  |  |  |  |
| Iteration |  | BBC | |  | Sky News | |  | A Single Div | |
|  | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request |
| 1 | 3.54 | 7.88 | 4.59 | 3.79 | 4.21 | 3.8 | 1.62 | 2.3 | 1.88 |
| 2 | 3.87 | 4.54 | 4.39 | 3.78 | 4.49 | 4.03 | 1.66 | 2.18 | 1.76 |
| 3 | 3.84 | 8.35 | 4.72 | 3.76 | 9.87 | 3.88 | 1.59 | 1.96 | 1.74 |
| 4 | 3.61 | 4.5 | 4.79 | 3.64 | 4.66 | 3.54 | 1.6 | 2.18 | 1.76 |
| 5 | 4.14 | 4.67 | 4.53 | 3.72 | 4.64 | 4.86 | 1.59 | 2.16 | 1.78 |
| 6 | 3.64 | 7.88 | 4.61 | 3.86 | 4.73 | 3.58 | 1.55 | 1.87 | 2.45 |
| 7 | 3.41 | 4.06 | 4.13 | 3.89 | 4.45 | 3.74 | 1.58 | 2.99 | 2.21 |
| 8 | 4.07 | 4.75 | 4.58 | 3.75 | 8.39 | 3.56 | 1.59 | 3.33 | 1.8 |
| 9 | 4.12 | 9.53 | 4.68 | 3.75 | 4.46 | 3.71 | 1.61 | 1.92 | 1.81 |
| 10 | 3.61 | 8.45 | 4.48 | 4.25 | 3.91 | 3.64 | 1.6 | 2.2 | 2.02 |
| Avg | 3.785 | 6.461 | 4.55 | 3.819 | 5.381 | 3.834 | 1.599 | 2.309 | 1.921 |
| Standard Deviation | 0.260906539 | 2.118859494 | 0.18761663 | 0.166563301 | 2.020502963 | 0.391895678 | 0.028460499 | 0.476035713 | 0.237180194 |
| Run 4 |  |  |  |  |  |  |  |  |  |
| Iteration |  | BBC | |  | Sky News | |  | A Single Div | |
|  | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request |
| 1 | 3.46 | 3.86 | 4.5 | 3.84 | 5.07 | 3.88 | 1.6 | 2.31 | 1.81 |
| 2 | 3.35 | 4.11 | 4.78 | 3.8 | 4.39 | 3.85 | 1.58 | 2.09 | 2.15 |
| 3 | 3.63 | 5.34 | 4.38 | 3.73 | 3.68 | 4.08 | 1.64 | 1.97 | 1.84 |
| 4 | 3.85 | 4.88 | 4.94 | 3.43 | 4.19 | 3.7 | 1.59 | 2.1 | 1.85 |
| 5 | 4.04 | 4.07 | [8 - Obvious Error] | 3.65 | 4.42 | 3.56 | 1.6 | 2.6 | 1.82 |
| 6 | 3.43 | 9.54 | 4.58 | 3.75 | 3.9 | 4.05 | 1.57 | 2.67 | 1.87 |
| 7 | 3.76 | 4.82 | 4.65 | 3.85 | 4.63 | 3.67 | 1.58 | 2.13 | 1.75 |
| 8 | 3.46 | 8.11 | 4.32 | 3.49 | 4.53 | 3.53 | 1.59 | 2.07 | 1.83 |
| 9 | 3.56 | 4.16 | 4.57 | 3.85 | 3.75 | 3.68 | 1.59 | 2.04 | 1.8 |
| 10 | 3.34 | 8.44 | 4.77 | 3.71 | 4.75 | 3.66 | 1.59 | 2.37 | 1.84 |
| Avg | 3.588 | 5.733 | 4.61 | 3.71 | 4.331 | 3.766 | 1.593 | 2.235 | 1.856 |
| Standard Deviation | 0.231170932 | 2.123126887 | 0.198934663 | 0.147798662 | 0.450812846 | 0.191380946 | 0.018885621 | 0.243139192 | 0.108341025 |
| Run 5 |  |  |  |  |  |  |  |  |  |
| Iteration |  | BBC | |  | Sky News | |  | A Single Div | |
|  | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request | No Cache Request | First Request | Second Request |
| 1 | 3.59 | 4.95 | 4.96 | 3.83 | 3.46 | 3.61 | 1.58 | 1.8 | 1.76 |
| 2 | 3.66 | 4.35 | 4.7 | 3.72 | 4.86 | 3.62 | 1.61 | 1.8 | 2.08 |
| 3 | 3.33 | 8.34 | 4.74 | 3.76 | 3.64 | 4.06 | 1.61 | 2.8 | 1.86 |
| 4 | 3.63 | 4.84 | 4.57 | 3.88 | 4.14 | 3.62 | 1.65 | 1.75 | 1.73 |
| 5 | 3.42 | 4.68 | 4.8 | 3.7 | 4.67 | 4.01 | 1.63 | 2.29 | 1.74 |
| 6 | 3.64 | 4.07 | 4.84 | 3.76 | 3.91 | 4 | 1.6 | 2.48 | 1.77 |
| 7 | 3.55 | 4.33 | 4.64 | 3.68 | 4.13 | 3.8 | 1.57 | 1.75 | 1.79 |
| 8 | 3.47 | 4.73 | 4.75 | 3.7 | 5.14 | 3.59 | 1.6 | 1.78 | 1.82 |
| 9 | 3.63 | 8.46 | 4.6 | 3.76 | 3.88 | 4.36 | 1.6 | 2.11 | 1.75 |
| 10 | 3.65 | 4.71 | 4.7 | 3.75 | 3.87 | 3.74 | 1.64 | 1.78 | 1.73 |
| Avg | 3.557 | 5.346 | 4.73 | 3.754 | 4.17 | 3.841 | 1.609 | 2.034 | 1.803 |
| Standard Deviation | 0.113436228 | 1.631422011 | 0.116809436 | 0.061680179 | 0.547336987 | 0.257571996 | 0.02514403 | 0.373696252 | 0.105835302 |

### Voice Recognition Results

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run 1 |  | | | |  | | |  | | |  | |
| Iteration | Local Execution | | | | | | | | | | | |
|  | Local CPU | | | | Local RAM | | | Execution Time | | | File Size | |
| 1 | 28.5 | | | | 0.15 | | | 1.62 | | | 245804 | |
| 2 | 28.23 | | | | 0.15 | | | 1.565 | | | 245804 | |
| 3 | 27.18 | | | | 0.15 | | | 1.556 | | | 245804 | |
| 4 | 28.41 | | | | 0.15 | | | 1.566 | | | 245804 | |
| 5 | 27.21 | | | | 0.15 | | | 1.547 | | | 245804 | |
| 6 | 27.76 | | | | 0.15 | | | 1.545 | | | 245804 | |
| 7 | 26.97 | | | | 0.15 | | | 1.591 | | | 245804 | |
| 8 | 28.01 | | | | 0.15 | | | 1.596 | | | 245804 | |
| 9 | 27.09 | | | | 0.15 | | | 1.535 | | | 245804 | |
| 10 | 27.29 | | | | 0.15 | | | 1.577 | | | 245804 | |
| Avg | 27.665 | | | | 0.15 | | | 1.5698 | | | 245804 | |
| SD | 0.58652 | | | | 0 | | | 0.026418848 | | | 0 | |
| Run 2 |  | | | |  | | |  | | |  | |
| Iteration | Local Execution | | | | | | | | | | | |
|  | Local CPU | | | | Local RAM | | | Execution Time | | | File Size | |
| 1 | 29.11 | | | | 0.14 | | | 1.591 | | | 245804 | |
| 2 | 27.64 | | | | 0.14 | | | 1.547 | | | 245804 | |
| 3 | 27.57 | | | | 0.14 | | | 1.544 | | | 245804 | |
| 4 | 28.94 | | | | 0.14 | | | 0.1547 | | | 245804 | |
| 5 | 27.49 | | | | 0.14 | | | 1.566 | | | 245804 | |
| 6 | 27.72 | | | | 0.14 | | | 1.588 | | | 245804 | |
| 7 | 27.47 | | | | 0.14 | | | 1.588 | | | 245804 | |
| 8 | 27.38 | | | | 0.14 | | | 1.542 | | | 245804 | |
| 9 | 28.06 | | | | 0.14 | | | 1.551 | | | 245804 | |
| 10 | 27.9 | | | | 0.14 | | | 1.613 | | | 245804 | |
| Avg | 27.928 | | | | 0.14 | | | 1.42847 | | | 245804 | |
| SD | 0.614253 | | | | 0 | | | 0.448215063 | | | 0 | |
| Run 3 |  | | | |  | | |  | | |  | |
| Iteration | Local Execution | | | | | | | | | | | |
|  | Local CPU | | | | Local RAM | | | Execution Time | | | File Size | |
| 1 | 28.66 | | | | 0.15 | | | 1.632 | | | 245804 | |
| 2 | 27.83 | | | | 0.15 | | | 1.545 | | | 245804 | |
| 3 | 26.33 | | | | 0.15 | | | 1.567 | | | 245804 | |
| 4 | 27.42 | | | | 0.15 | | | 1.532 | | | 245804 | |
| 5 | 26.83 | | | | 0.15 | | | 1.518 | | | 245804 |
| 6 | 27.48 | | | | 0.15 | | | 1.52 | | | 245804 | |
| 7 | 27.41 | | | | 0.15 | | | 1.543 | | | 245804 | |
| 8 | 27.6 | | | | 0.15 | | | 1.518 | | | 245804 | |
| 9 | 28.01 | | | | 0.15 | | | 1.533 | | | 245804 | |
| 10 | 27.11 | | | | 0.15 | | | 1.529 | | | 245804 | |
| Avg | 27.468 | | | | 0.15 | | | 1.5437 | | | 245804 | |
| SD | 0.641107 | | | | 0 | | | 0.034467537 | | | 0 | |
| Run 4 |  | | | |  | | |  | | |  | |
| Iteration | Local Execution | | | | | | | | | | | |
|  | Local CPU | | | | Local RAM | | | Execution Time | | | File Size | |
| 1 | 27.79 | | | | 0.15 | | | 1.619 | | | 245804 | |
| 2 | 28.1 | | | | 0.15 | | | 1.542 | | | 245804 | |
| 3 | 27.2 | | | | 0.15 | | | 1.564 | | | 245804 | |
| 4 | 27.96 | | | | 0.15 | | | 1.569 | | | 245804 | |
| 5 | 26.92 | | | | 0.15 | | | 1.548 | | | 245804 | |
| 6 | 26.9 | | | | 0.15 | | | 1.527 | | | 245804 | |
| 7 | 27 | | | | 0.15 | | | 1.535 | | | 245804 | |
| 8 | 28.01 | | | | 0.15 | | | 1.54 | | | 245804 | |
| 9 | 27.29 | | | | 0.15 | | | 1.538 | | | 245804 | |
| 10 | 27.9 | | | | 0.15 | | | 1.539 | | | 245804 | |
| Avg | 27.507 | | | | 0.15 | | | 1.5521 | | | 245804 | |
| SD | 0.489423 | | | | 0 | | | 0.026793241 | | | 0 | |
| Run 5 |  | | | |  | | |  | | |  | |
| Iteration | Local Execution | | | | | | | | | | | |
|  | Local CPU | | | | Local RAM | | | Execution Time | | | File Size | |
| 1 | 28.42 | | | | 0.15 | | | 1.622 | | | 245804 | |
| 2 | 27.72 | | | | 0.15 | | | 1.565 | | | 245804 | |
| 3 | 28.11 | | | | 0.15 | | | 1.55 | | | 245804 | |
| 4 | 28.04 | | | | 0.15 | | | 1.561 | | | 245804 | |
| 5 | 28.41 | | | | 0.15 | | | 1.543 | | | 245804 | |
| 6 | 26.64 | | | | 0.15 | | | 1.537 | | | 245804 | |
| 7 | 27.42 | | | | 0.15 | | | 1.553 | | | 245804 | |
| 8 | 28.02 | | | | 0.15 | | | 1.559 | | | 245804 | |
| 9 | 28.18 | | | | 0.15 | | | 1.553 | | | 245804 | |
| 10 | 27.8 | | | | 0.15 | | | 1.55 | | | 245804 | |
| Avg | 27.876 | | | | 0.15 | | | 1.5593 | | | 245804 | |
| SD | 0.531417 | | | | 0 | | | 0.023527998 | | | 0 | |
| Remote request without pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | Local CPU | | | | Edge CPU | | | Data Centre CPU | Local RAM | | | Edge RAM | | | Data Centre RAM | Data Received At DC | | Data Sent From DC | | | File Size |
| 0.904 | | 10.28 | | | | 1.74 | | | 33.42 | 0.15 | | | 0.54 | | | 6.22 | 246051 | | 7 | | | 245804 |
| 0.793 | | 4.72 | | | | 6.03 | | | 34.05 | 0.15 | | | 0.54 | | | 6.38 | 246051 | | 7 | | | 245804 |
| 0.854 | | 4.12 | | | | 0.95 | | | 31.17 | 0.15 | | | 0.54 | | | 6.41 | 246051 | | 7 | | | 245804 |
| 0.847 | | 2.41 | | | | 0.95 | | | 28.24 | 0.15 | | | 0.54 | | | 6.41 | 246051 | | 7 | | | 245804 |
| 0.838 | | 3.92 | | | | 2.88 | | | 29.45 | 0.15 | | | 0.54 | | | 6.36 | 246051 | | 7 | | | 245804 |
| 0.903 | | 3.92 | | | | 2.4 | | | 22.6 | 0.15 | | | 0.54 | | | 6.35 | 246051 | | 7 | | | 245804 |
| 0.913 | | 4.97 | | | | 2.59 | | | 37.27 | 0.15 | | | 0.54 | | | 6.35 | 246051 | | 7 | | | 245804 |
| 0.892 | | 5.31 | | | | 2.1 | | | 35.95 | 0.15 | | | 0.54 | | | 6.36 | 246051 | | 7 | | | 245804 |
| 0.828 | | 4.55 | | | | 3.49 | | | 28.89 | 0.14 | | | 0.54 | | | 6.34 | 246051 | | 7 | | | 245804 |
| 0.817 | | 3.43 | | | | 3.61 | | | 34.65 | 0.14 | | | 0.54 | | | 6.29 | 246051 | | 7 | | | 245804 |
| 0.8589 | | 4.763 | | | | 2.674 | | | 31.569 | 0.148 | | | 0.54 | | | 6.347 | 246051 | | 7 | | | 245804 |
| 0.041709178 | | 2.107521 | | | | 1.493521 | | | 4.39848825 | 0.0042164 | | | 0 | | | 0.056578363 | 0 | | 0 | | | 0 |
| Remote request without pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | Local CPU | | | | Edge CPU | | | Data Centre CPU | Local RAM | | | Edge RAM | | | Data Centre RAM | Data Received At DC | | Data Sent From DC | | | File Size |
| 0.955 | | 8.95 | | | | 1.66 | | | 25.76 | 0.15 | | | 0.54 | | | 6.37 | 246051 | | 7 | | | 245804 |
| 0.808 | | 4.7 | | | | 1.81 | | | 22.48 | 0.15 | | | 0.54 | | | 6.37 | 246051 | | 7 | | | 245804 |
| 0.828 | | 4.26 | | | | 2.06 | | | 25.14 | 0.15 | | | 0.54 | | | 6.33 | 246051 | | 7 | | | 245804 |
| 0.754 | | 5.94 | | | | 5.81 | | | 28.19 | 0.15 | | | 0.54 | | | 6.33 | 246051 | | 7 | | | 245804 |
| 0.728 | | 5.15 | | | | 2.42 | | | 28.59 | 0.15 | | | 0.54 | | | 6.33 | 246051 | | 7 | | | 245804 |
| 0.699 | | 5.04 | | | | 4.38 | | | 22.33 | 0.15 | | | 0.54 | | | 6.32 | 246051 | | 7 | | | 245804 |
| 0.831 | | 4.53 | | | | 1.71 | | | 27.61 | 0.14 | | | 0.54 | | | 6.34 | 246051 | | 7 | | | 245804 |
| 0.91 | | 3.06 | | | | 3.33 | | | 33.3 | 0.14 | | | 0.54 | | | 6.3 | 246051 | | 7 | | | 245804 |
| 0.785 | | 5.1 | | | | 2.73 | | | 27.18 | 0.14 | | | 0.54 | | | 6.33 | 246051 | | 7 | | | 245804 |
| 0.691 | | 5.11 | | | | 2.34 | | | 28.71 | 0.14 | | | 0.54 | | | 6.32 | 246051 | | 7 | | | 245804 |
| 0.7989 | | 5.184 | | | | 2.825 | | | 26.929 | 0.146 | | | 0.54 | | | 6.334 | 246051 | | 7 | | | 245804 |
| 0.086621078 | | 1.520813 | | | | 1.341949 | | | 3.234345031 | 0.005164 | | | 0 | | | 0.021705094 | 0 | | 0 | | | 0 |
| Remote request without pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | Local CPU | | | | Edge CPU | | | Data Centre CPU | Local RAM | | | Edge RAM | | | Data Centre RAM | Data Received At DC | | Data Sent From DC | | | File Size |
| 2.905 | | 5.87 | | | | 2.37 | | | 26.13 | 0.14 | | | 0.54 | | | 6.33 | 246051 | | 7 | | | 245804 |
| 2.295 | | 3.26 | | | | 0.46 | | | 15.45 | 0.14 | | | 0.54 | | | 6.33 | 246051 | | 7 | | | 245804 |
| 2.944 | | 2.21 | | | | 0.44 | | | 16.91 | 0.14 | | | 0.54 | | | 5.78 | 246051 | | 7 | | | 245804 |
| 0.88 | | 5.16 | | | | 3.4 | | | 24.87 | 0.14 | | | 0.54 | | | 5.77 | 246051 | | 7 | | | 245804 |
| 1.037 | | 5.81 | | | | 3.28 | | | 33.79 | 0.14 | | | 0.54 | | | 5.77 | 246051 | | 7 | | | 245804 |
| 3.374 | | 1.71 | | | | 1.23 | | | 19.29 | 0.14 | | | 0.54 | | | 5.64 | 246051 | | 7 | | | 245804 |
| 2.55 | | 2.36 | | | | 0.52 | | | 17.44 | 0.14 | | | 0.54 | | | 5.74 | 246051 | | 7 | | | 245804 |
| 2.478 | | 2.92 | | | | 0.53 | | | 16.54 | 0.14 | | | 0.54 | | | 5.73 | 246051 | | 7 | | | 245804 |
| 1.344 | | 3.18 | | | | 0.8 | | | 18.38 | 0.14 | | | 0.54 | | | 5.82 | 246051 | | 7 | | | 245804 |
| 1.833 | | 2.6 | | | | 0.99 | | | 17.03 | 0.14 | | | 0.54 | | | 6.06 | 246051 | | 7 | | | 245804 |
| 2.164 | | 3.508 | | | | 1.402 | | | 20.583 | 0.14 | | | 0.54 | | | 5.897 | 246051 | | 7 | | | 245804 |
| 0.855510244 | | 1.534136 | | | | 1.172308 | | | 5.857418942 | 2.926E-17 | | | 0 | | | 0.252104476 | 0 | | 0 | | | 0 |
| Remote request without pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | Local CPU | | | | Edge CPU | | | Data Centre CPU | Local RAM | | | Edge RAM | | | Data Centre RAM | Data Received At DC | | Data Sent From DC | | | File Size |
| 1.075 | | 14.86 | | | | 1.58 | | | 30.19 | 0.14 | | | 0.54 | | | 5.71 | 246051 | | 7 | | | 245804 |
| 0.819 | | 8.21 | | | | 1.75 | | | 35.57 | 0.14 | | | 0.54 | | | 5.7 | 246051 | | 7 | | | 245804 |
| 0.779 | | 6.17 | | | | 1.05 | | | 47.24 | 0.14 | | | 0.54 | | | 5.72 | 246051 | | 7 | | | 245804 |
| 0.843 | | 6.55 | | | | 1.63 | | | 34.34 | 0.14 | | | 0.54 | | | 5.73 | 246051 | | 7 | | | 245804 |
| 0.859 | | 5.56 | | | | 1.26 | | | 29.23 | 0.14 | | | 0.54 | | | 5.73 | 246051 | | 7 | | | 245804 |
| 0.82 | | 5.59 | | | | 0.99 | | | 34.49 | 0.14 | | | 0.54 | | | 5.73 | 246051 | | 7 | | | 245804 |
| 0.928 | | 4.89 | | | | 1.98 | | | 32.76 | 0.14 | | | 0.54 | | | 5.78 | 246051 | | 7 | | | 245804 |
| 0.883 | | 4.3 | | | | 0.6 | | | 37.74 | 0.14 | | | 0.54 | | | 5.77 | 246051 | | 7 | | | 245804 |
| 0.81 | | 5.64 | | | | 0.66 | | | 26.74 | 0.14 | | | 0.54 | | | 5.77 | 246051 | | 7 | | | 245804 |
| 0.881 | | 4.6 | | | | 0.93 | | | 32.92 | 0.14 | | | 0.54 | | | 5.75 | 246051 | | 7 | | | 245804 |
| 0.8697 | | 6.637 | | | | 1.243 | | | 34.122 | 0.14 | | | 0.54 | | | 5.739 | 246051 | | 7 | | | 245804 |
| 0.084004034 | | 3.095352 | | | | 0.473335 | | | 5.623735018 | 0 | | | 0 | | | 0.02726414 | 0 | | 0 | | | 0 |
| Remote request without pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | Local CPU | | | | Edge CPU | | | Data Centre CPU | Local RAM | | | Edge RAM | | | Data Centre RAM | Data Received At DC | | Data Sent From DC | | | File Size |
| 0.853 | | 14.84 | | | | 2.4 | | | 25.88 | 0.14 | | | 0.54 | | | 6.01 | 246051 | | 7 | | | 245804 |
| 0.767 | | 6.58 | | | | 1.36 | | | 31.35 | 0.14 | | | 0.54 | | | 6.02 | 246051 | | 7 | | | 245804 |
| 0.625 | | 6.88 | | | | 2.61 | | | 28.67 | 0.14 | | | 0.54 | | | 6.11 | 246051 | | 7 | | | 245804 |
| 0.653 | | 8.11 | | | | 2.26 | | | 25.76 | 0.14 | | | 0.54 | | | 6.07 | 246051 | | 7 | | | 245804 |
| 0.714 | | 4.64 | | | | 2.3 | | | 34.51 | 0.14 | | | 0.54 | | | 6.07 | 246051 | | 7 | | | 245804 |
| 0.697 | | 4.03 | | | | 2.75 | | | 27.52 | 0.14 | | | 0.54 | | | 6.06 | 246051 | | 7 | | | 245804 |
| 0.695 | | 6.2 | | | | 1.25 | | | 32.03 | 0.14 | | | 0.54 | | | 6.05 | 246051 | | 7 | | | 245804 |
| 0.733 | | 4.83 | | | | 3.89 | | | 34.84 | 0.14 | | | 0.54 | | | 6.05 | 246051 | | 7 | | | 245804 |
| 0.582 | | 7.86 | | | | 1.5 | | | 38.23 | 0.14 | | | 0.54 | | | 6.03 | 246051 | | 7 | | | 245804 |
| 0.551 | | 7.87 | | | | 4.85 | | | 18.62 | 0.14 | | | 0.54 | | | 6.02 | 246051 | | 7 | | | 245804 |
| 0.687 | | 7.184 | | | | 2.517 | | | 29.741 | 0.14 | | | 0.54 | | | 6.049 | 246051 | | 7 | | | 245804 |
| 0.089122886 | | 3.051154 | | | | 1.130192 | | | 5.668106582 | 0 | | | 0 | | | 0.030349812 | 0 | | 0 | | | 0 |
| Remote request with pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | | Local CPU | Edge CPU | | | Data Centre CPU | | | | Local RAM | | | Edge RAM | Data Centre RAM | | | Data Received At DC | | Data Sent From DC | File Size | |
| 2.058 | | | 4.88 | 18.83 | | | 86.086 | | | | 0.14 | | | 0.54 | 6.26 | | | 8 | | 7 | 245804 | |
| 1.957 | | | 2.7 | 20.16 | | | 34.38 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 1.978 | | | 1.66 | 20.73 | | | 10.07 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 1.982 | | | 1.53 | 20.18 | | | 33.24 | | | | 0.14 | | | 0.54 | 6.26 | | | 8 | | 7 | 245804 | |
| 1.995 | | | 3.4 | 19.92 | | | 59.54 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 2.024 | | | 2.97 | 18.83 | | | 77.08 | | | | 0.14 | | | 0.54 | 6.26 | | | 8 | | 7 | 245804 | |
| 2.062 | | | 2.93 | 18.9 | | | 46.51 | | | | 0.14 | | | 0.54 | 6.26 | | | 8 | | 7 | 245804 | |
| 2.082 | | | 1.09 | 19.24 | | | 53.78 | | | | 0.14 | | | 0.54 | 6.26 | | | 8 | | 7 | 245804 | |
| 2.004 | | | 2.14 | 21.32 | | | 23.38 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 2.042 | | | 2.57 | 19.59 | | | 83.06 | | | | 0.14 | | | 0.54 | 6.26 | | | 8 | | 7 | 245804 | |
| 2.0184 | | | 2.587 | 19.77 | | | 50.7126 | | | | 0.14 | | | 0.54 | 6.264 | | | 8 | | 7 | 245804 | |
| 0.041612498 | | | 1.085951 | 0.850111 | | | 25.96750706 | | | | 0 | | | 0 | 0.005163978 | | | 0 | | 0 | 0 | |
| Remote request with pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | | Local CPU | Edge CPU | | | Data Centre CPU | | | | Local RAM | | | Edge RAM | Data Centre RAM | | | Data Received At DC | | Data Sent From DC | File Size | |
| 1.738 | | | 8.13 | 23.11 | | | 49.98 | | | | 0.14 | | | 0.54 | 6.3 | | | 8 | | 7 | 245804 | |
| 1.739 | | | 1.89 | 22.27 | | | 32.57 | | | | 0.14 | | | 0.54 | 6.3 | | | 8 | | 7 | 245804 | |
| 1.716 | | | 2.2 | 24.11 | | | 17.08 | | | | 0.14 | | | 0.54 | 6.29 | | | 8 | | 7 | 245804 | |
| 1.7 | | | 3.28 | 22.61 | | | 43.86 | | | | 0.14 | | | 0.54 | 6.29 | | | 8 | | 7 | 245804 | |
| 1.728 | | | 2.91 | 22.84 | | | 28.42 | | | | 0.14 | | | 0.54 | 6.29 | | | 8 | | 7 | 245804 | |
| 1.855 | | | 2.3 | 23.96 | | | 14.91 | | | | 0.14 | | | 0.54 | 6.39 | | | 8 | | 7 | 245804 | |
| 1.66 | | | 3.77 | 23.34 | | | 46.92 | | | | 0.14 | | | 0.54 | 6.36 | | | 8 | | 7 | 245804 | |
| 1.774 | | | 2.83 | 21.59 | | | 29.76 | | | | 0.14 | | | 0.54 | 6.37 | | | 8 | | 7 | 245804 | |
| 1.672 | | | 2.11 | 23.11 | | | 57.1 | | | | 0.14 | | | 0.54 | 6.35 | | | 8 | | 7 | 245804 | |
| 1.763 | | | 3.29 | 21.91 | | | 35.03 | | | | 0.14 | | | 0.54 | 6.37 | | | 8 | | 7 | 245804 | |
| 1.7345 | | | 3.271 | 22.885 | | | 35.563 | | | | 0.14 | | | 0.54 | 6.331 | | | 8 | | 7 | 245804 | |
| 0.055733991 | | | 1.811356 | 0.819759 | | | 13.87091686 | | | | 0 | | | 0 | 0.04040077 | | | 0 | | 0 | 0 | |
| Remote request with pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | | Local CPU | Edge CPU | | | Data Centre CPU | | | | Local RAM | | | Edge RAM | Data Centre RAM | | | Data Received At DC | | Data Sent From DC | File Size | |
| 2.022 | | | 9.75 | 20.41 | | | 31.26 | | | | 0.14 | | | 0.54 | 6.05 | | | 8 | | 7 | 245804 | |
| 1.734 | | | 2.19 | 24.67 | | | 44.5 | | | | 0.14 | | | 0.54 | 6.04 | | | 8 | | 7 | 245804 | |
| 1.763 | | | 2.29 | 22.43 | | | 60.22 | | | | 0.14 | | | 0.54 | 6.04 | | | 8 | | 7 | 245804 | |
| 1.768 | | | 2.43 | 21.96 | | | 44.37 | | | | 0.14 | | | 0.54 | 6.03 | | | 8 | | 7 | 245804 | |
| 1.774 | | | 2.42 | 22.6 | | | 22.9 | | | | 0.14 | | | 0.54 | 6.04 | | | 8 | | 7 | 245804 | |
| 1.878 | | | 2.27 | 20.69 | | | 44.08 | | | | 0.14 | | | 0.54 | 6.04 | | | 8 | | 7 | 245804 | |
| 1.751 | | | 2.45 | 22.29 | | | 42.83 | | | | 0.14 | | | 0.54 | 6.04 | | | 8 | | 7 | 245804 | |
| 1.717 | | | 2.49 | 22.66 | | | 26.96 | | | | 0.14 | | | 0.54 | 6.04 | | | 8 | | 7 | 245804 | |
| 1.764 | | | 2.98 | 22.06 | | | 39.7 | | | | 0.14 | | | 0.54 | 6.03 | | | 8 | | 7 | 245804 | |
| 1.771 | | | 2.83 | 21.82 | | | 62.98 | | | | 0.14 | | | 0.54 | 6.02 | | | 8 | | 7 | 245804 | |
| 1.7942 | | | 3.21 | 22.159 | | | 41.98 | | | | 0.14 | | | 0.54 | 6.037 | | | 8 | | 7 | 245804 | |
| 0.090605126 | | | 2.311032 | 1.165356 | | | 12.91276801 | | | | 0 | | | 0 | 0.008232726 | | | 0 | | 0 | 0 | |
| Remote request with pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | | Local CPU | Edge CPU | | | Data Centre CPU | | | | Local RAM | | | Edge RAM | Data Centre RAM | | | Data Received At DC | | Data Sent From DC | File Size | |
| 1.814 | | | 9.21 | 22.59 | | | 25.36 | | | | 0.14 | | | 0.54 | 5.74 | | | 8 | | 7 | 245804 | |
| 1.768 | | | 2.01 | 22.01 | | | 11.79 | | | | 0.14 | | | 0.54 | 5.84 | | | 8 | | 7 | 245804 | |
| 1.76 | | | 2.87 | 21.99 | | | 59.06 | | | | 0.14 | | | 0.54 | 5.8 | | | 8 | | 7 | 245804 | |
| 1.882 | | | 3.31 | 22.28 | | | 36.38 | | | | 0.14 | | | 0.54 | 5.79 | | | 8 | | 7 | 245804 | |
| 1.75 | | | 3.02 | 21.94 | | | 48.6 | | | | 0.14 | | | 0.54 | 5.8 | | | 8 | | 7 | 245804 | |
| 1.712 | | | 3.23 | 23.02 | | | 33.7 | | | | 0.14 | | | 0.54 | 5.8 | | | 8 | | 7 | 245804 | |
| 1.745 | | | 3.16 | 22.39 | | | 32.82 | | | | 0.14 | | | 0.54 | 5.8 | | | 8 | | 7 | 245804 | |
| 1.674 | | | 2.56 | 23.35 | | | 19.96 | | | | 0.14 | | | 0.54 | 5.62 | | | 8 | | 7 | 245804 | |
| 1.747 | | | 2.86 | 21.73 | | | 44.96 | | | | 0.14 | | | 0.54 | 5.63 | | | 8 | | 7 | 245804 | |
| 1.921 | | | 1.97 | 20.24 | | | 32.59 | | | | 0.14 | | | 0.54 | 5.64 | | | 8 | | 7 | 245804 | |
| 1.7773 | | | 3.42 | 22.154 | | | 34.522 | | | | 0.14 | | | 0.54 | 5.746 | | | 8 | | 7 | 245804 | |
| 0.075267006 | | | 2.088226 | 0.841866 | | | 13.87544017 | | | | 0 | | | 0 | 0.083692559 | | | 0 | | 0 | 0 | |
| Remote request with pre-processing | | | | | | | | | | | | | | | | | | | | | | |
| Request Time | | | Local CPU | Edge CPU | | | Data Centre CPU | | | | Local RAM | | | Edge RAM | Data Centre RAM | | | Data Received At DC | | Data Sent From DC | File Size | |
| 1.775 | | | 4.56 | 23.31 | | | 46.01 | | | | 0.14 | | | 0.54 | 6.25 | | | 8 | | 7 | 245804 | |
| 1.748 | | | 3 | 22.49 | | | 20.84 | | | | 0.14 | | | 0.54 | 6.26 | | | 8 | | 7 | 245804 | |
| 1.755 | | | 2.16 | 24.08 | | | 46.54 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 1.741 | | | 2.32 | 22.22 | | | 46.35 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 1.746 | | | 2.72 | 22.73 | | | 18.84 | | | | 0.14 | | | 0.54 | 6.25 | | | 8 | | 7 | 245804 | |
| 1.846 | | | 2.19 | 22.44 | | | 77.08 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 1.757 | | | 2.15 | 22.12 | | | 20.58 | | | | 0.14 | | | 0.54 | 6.26 | | | 8 | | 7 | 245804 | |
| 1.717 | | | 2.2 | 23.53 | | | 35.8 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 1.809 | | | 2.9 | 21.62 | | | 52.03 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 1.726 | | | 2.91 | 22.41 | | | 48.04 | | | | 0.14 | | | 0.54 | 6.27 | | | 8 | | 7 | 245804 | |
| 1.762 | | | 2.711 | 22.695 | | | 41.211 | | | | 0.14 | | | 0.54 | 6.264 | | | 8 | | 7 | 245804 | |
| 0.03904698 | | | 0.73647 | 0.737492 | | | 17.94579954 | | | | 0 | | | 0 | 0.00843274 | | | 0 | | 0 | 0 | |

### Load Balancing Results

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run 1 |  | |  | | |  | | | |  | | |  | | |  | | |  | |  | |
| Iteration | Load Balance 10 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | Avg Request Time | | | Avg Local CPU | | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | Avg Data Centre RAM | |
| 1 | 2.521 | | 2.092 | | | 5.46 | | | | 6.57 | | | 57.91 | | | 0.11 | | | 0.43 | | 10.08 | |
| 2 | 2.393 | | 1.939 | | | 7.29 | | | | 6.43 | | | 57.44 | | | 0.1 | | | 0.43 | | 10.05 | |
| 3 | 2.443 | | 1.81 | | | 6.83 | | | | 8.52 | | | 56.58 | | | 0.1 | | | 0.43 | | 10.07 | |
| 4 | 2.469 | | 2.102 | | | 6.42 | | | | 5.3 | | | 64.64 | | | 0.1 | | | 0.43 | | 10.03 | |
| 5 | 2.429 | | 1.839 | | | 6.79 | | | | 5.85 | | | 54.38 | | | 0.1 | | | 0.43 | | 10.05 | |
| 6 | 2.392 | | 1.95 | | | 7.44 | | | | 4.28 | | | 60.19 | | | 0.11 | | | 0.43 | | 10.05 | |
| 7 | 2.946 | | 2.24 | | | 6.61 | | | | 5.17 | | | 42.83 | | | 0.11 | | | 0.43 | | 10.06 | |
| 8 | 2.684 | | 2.036 | | | 7.06 | | | | 4.32 | | | 58.78 | | | 0.1 | | | 0.43 | | 10.04 | |
| 9 | 2.465 | | 1.912 | | | 6.97 | | | | 5.32 | | | 59.72 | | | 0.1 | | | 0.43 | | 10.03 | |
| 10 | 2.842 | | 2.086 | | | 6.84 | | | | 6.45 | | | 46.97 | | | 0.1 | | | 0.43 | | 10.03 | |
| Avg | 2.5584 | | 2.0006 | | | 6.771 | | | | 5.821 | | | 55.944 | | | 0.103 | | | 0.43 | | 10.049 | |
| SD | 0.197107 | | 0.133808487 | | | 0.549392594 | | | | 1.255516273 | | | 6.475710514 | | | 0.004830459 | | | 0 | | 0.017288403 | |
| Run 2 |  | |  | | |  | | | |  | | |  | | |  | | |  | |  | |
| Iteration | Load Balance 10 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | Avg Request Time | | | Avg Local CPU | | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | Avg Data Centre RAM | |
| 1 | 3.827 | | 3.495 | | | 4.71 | | | | 2.89 | | | 49.48 | | | 0.1 | | | 0.43 | | 9.92 | |
| 2 | 2.741 | | 2.172 | | | 6.62 | | | | 4.66 | | | 49.56 | | | 0.1 | | | 0.43 | | 9.91 | |
| 3 | 2.605 | | 1.943 | | | 7.54 | | | | 5.46 | | | 48.34 | | | 0.1 | | | 0.43 | | 9.84 | |
| 4 | 2.541 | | 1.94 | | | 7.11 | | | | 5.57 | | | 50.95 | | | 0.09 | | | 0.43 | | 9.88 | |
| 5 | 2.795 | | 2.098 | | | 7.77 | | | | 4.36 | | | 45.05 | | | 0.11 | | | 0.43 | | 9.9 | |
| 6 | 2.597 | | 1.887 | | | 7.74 | | | | 5.5 | | | 48.46 | | | 0.1 | | | 0.43 | | 9.92 | |
| 7 | 2.671 | | 2.017 | | | 6.31 | | | | 5.13 | | | 42.1 | | | 0.1 | | | 0.43 | | 9.91 | |
| 8 | 2.817 | | 2.03 | | | 7.62 | | | | 5.78 | | | 48.73 | | | 0.1 | | | 0.43 | | 9.92 | |
| 9 | 2.672 | | 2.374 | | | 4.93 | | | | 4.1 | | | 54.86 | | | 0.1 | | | 0.42 | | 9.89 | |
| 10 | 2.611 | | 1.899 | | | 7.39 | | | | 5.4 | | | 44.47 | | | 0.1 | | | 0.42 | | 9.91 | |
| Avg | 2.7877 | | 2.1855 | | | 6.774 | | | | 4.885 | | | 48.2 | | | 0.1 | | | 0.428 | | 9.9 | |
| SD | 0.3759539 | | 0.483160256 | | | 1.136243323 | | | | 0.896093373 | | | 3.602770539 | | | 0.004714045 | | | 0.00421637 | | 0.024944383 | |
| Run 3 |  | |  | | |  | | | |  | | |  | | |  | | |  | |  | |
| Iteration | Load Balance 10 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | Avg Request Time | | | Avg Local CPU | | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | Avg Data Centre RAM | |
| 1 | 2.618 | | 1.971 | | | 7.85 | | | | 5.51 | | | 49.14 | | | 0.1 | | | 0.43 | | 9.89 | |
| 2 | 2.704 | | 2.202 | | | 6.32 | | | | 4.75 | | | 51.47 | | | 0.1 | | | 0.43 | | 9.88 | |
| 3 | 2.433 | | 1.784 | | | 7.2 | | | | 6.61 | | | 58.38 | | | 0.1 | | | 0.43 | | 9.91 | |
| 4 | 2.666 | | 2.247 | | | 6.71 | | | | 4.88 | | | 52.12 | | | 0.1 | | | 0.43 | | 9.83 | |
| 5 | 3.931 | | 3.459 | | | 4.22 | | | | 3.23 | | | 34.24 | | | 0.1 | | | 0.43 | | 9.87 | |
| 6 | 3.141 | | 2.057 | | | 5.79 | | | | 6.61 | | | 59.53 | | | 0.1 | | | 0.43 | | 9.87 | |
| 7 | 2.59 | | 1.977 | | | 6.28 | | | | 5.89 | | | 54.64 | | | 0.1 | | | 0.43 | | 9.89 | |
| 8 | 3.432 | | 2.351 | | | 5.93 | | | | 4.18 | | | 55.6 | | | 0.1 | | | 0.43 | | 9.9 | |
| 9 | 2.446 | | 1.928 | | | 8.2 | | | | 5.07 | | | 52.38 | | | 0.1 | | | 0.43 | | 9.91 | |
| 10 | 2.579 | | 2.057 | | | 6.1 | | | | 4.75 | | | 49.23 | | | 0.1 | | | 0.43 | | 9.91 | |
| Avg | 2.854 | | 2.2033 | | | 6.46 | | | | 5.148 | | | 51.673 | | | 0.1 | | | 0.43 | | 9.886 | |
| SD | 0.492042 | | 0.47141561 | | | 1.130329352 | | | | 1.052349752 | | | 7.055143199 | | | 0 | | | 0 | | 0.025033311 | |
| Run 4 |  | |  | | |  | | | |  | | |  | | |  | | |  | |  | |
| Iteration | Load Balance 10 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | Avg Request Time | | | Avg Local CPU | | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | Avg Data Centre RAM | |
| 1 | 2.609 | | 2.179 | | | 6.1 | | | | 5.01 | | | 69.09 | | | 0.1 | | | 0.43 | | 9.89 | |
| 2 | 10.259 | | 9.728 | | | 1.86 | | | | 1.55 | | | 47.51 | | | 0.09 | | | 0.43 | | 9.89 | |
| 3 | 2.687 | | 2.687 | | | 6.03 | | | | 5.48 | | | 47.63 | | | 0.1 | | | 0.43 | | 9.91 | |
| 4 | 3.037 | | 3.037 | | | 5.82 | | | | 4.06 | | | 42.47 | | | 0.11 | | | 0.43 | | 9.9 | |
| 5 | 2.547 | | 2.547 | | | 8.25 | | | | 5.53 | | | 45.7 | | | 0.1 | | | 0.43 | | 9.91 | |
| 6 | 2.823 | | 2.823 | | | 4.68 | | | | 4.08 | | | 50.64 | | | 0.1 | | | 0.43 | | 9.91 | |
| 7 | 2.56 | | 2.56 | | | 6.22 | | | | 5.33 | | | 59.12 | | | 0.1 | | | 0.43 | | 9.91 | |
| 8 | 2.405 | | 2.405 | | | 6.9 | | | | 5.07 | | | 49.72 | | | 0.09 | | | 0.43 | | 9.9 | |
| 9 | 2.828 | | 2.828 | | | 5.73 | | | | 3.84 | | | 49.78 | | | 0.09 | | | 0.43 | | 9.83 | |
| 10 | 2.71 | | 2.71 | | | 6.66 | | | | 5.39 | | | 48.92 | | | 0.1 | | | 0.43 | | 9.9 | |
| Avg | 3.4465 | | 3.3504 | | | 5.825 | | | | 4.534 | | | 51.058 | | | 0.098 | | | 0.43 | | 9.895 | |
| SD | 2.4002205 | | 2.253624449 | | | 1.668280885 | | | | 1.228542045 | | | 7.63849578 | | | 0 | | | 0 | | 0.024152295 | |
| Run 5 |  | |  | | |  | | | |  | | |  | | |  | | |  | |  | |
| Iteration | Load Balance 10 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | Avg Request Time | | | Avg Local CPU | | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | Avg Data Centre RAM | |
| 1 | 2.768 | | 2.239 | | | 4.81 | | | | 5.65 | | | 57.34 | | | 0.1 | | | 0.43 | | 9.86 | |
| 2 | 2.378 | | 1.828 | | | 6.74 | | | | 7.27 | | | 52.7 | | | 0.1 | | | 0.43 | | 9.86 | |
| 3 | 3.433 | | 2.901 | | | 4.45 | | | | 4.25 | | | 52.87 | | | 0.1 | | | 0.43 | | 9.86 | |
| 4 | 2.626 | | 2.11 | | | 6.28 | | | | 5.23 | | | 46.78 | | | 0.09 | | | 0.43 | | 9.86 | |
| 5 | 3.983 | | 3.235 | | | 3.68 | | | | 3.78 | | | 47.09 | | | 0.09 | | | 0.43 | | 9.86 | |
| 6 | 2.7 | | 2.073 | | | 6.09 | | | | 5.67 | | | 43.79 | | | 0.1 | | | 0.43 | | 9.88 | |
| 7 | 3.464 | | 2.606 | | | 5.06 | | | | 4.13 | | | 42.08 | | | 0.1 | | | 0.43 | | 9.88 | |
| 8 | 2.691 | | 2.108 | | | 5.7 | | | | 5.48 | | | 46.04 | | | 0.1 | | | 0.43 | | 9.88 | |
| 9 | 5.162 | | 3.813 | | | 3.94 | | | | 2.8 | | | 35.82 | | | 0.1 | | | 0.43 | | 9.88 | |
| 10 | 2.64 | | 2.146 | | | 5.96 | | | | 5.33 | | | 50.32 | | | 0.1 | | | 0.43 | | 9.86 | |
| Avg | 3.1845 | | 2.5059 | | | 5.271 | | | | 4.959 | | | 47.483 | | | 0.098 | | | 0.43 | | 9.868 | |
| SD | 0.8562567 | | 0.629726299 | | | 1.040271866 | | | | 1.249332711 | | | 6.173716781 | | | 0.00421637 | | | 0 | | 0.010327956 | |
| Load Balance 10 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 5.837 | | 5.502 | | | 3.44 | | | 91.86 | | | 70.82 | | | 0.1 | | | 0.4 | | | 9.9 | |
| 6.251 | | 5.455 | | | 4.52 | | | 90.79 | | | 56.51 | | | 0.1 | | | 0.41 | | | 9.91 | |
| 5.764 | | 5.463 | | | 3.2 | | | 91.95 | | | 70.29 | | | 0.1 | | | 0.41 | | | 9.91 | |
| 5.755 | | 5.371 | | | 3.46 | | | 91.35 | | | 46.72 | | | 0.1 | | | 0.4 | | | 9.85 | |
| 5.741 | | 5.433 | | | 4.09 | | | 93.74 | | | 70.39 | | | 0.1 | | | 0.41 | | | 9.91 | |
| 5.796 | | 5.503 | | | 3.45 | | | 91.94 | | | 45.56 | | | 0.1 | | | 0.41 | | | 9.91 | |
| 5.766 | | 5.383 | | | 3.2 | | | 93.04 | | | 58.86 | | | 0.1 | | | 0.4 | | | 9.91 | |
| 5.781 | | 5.508 | | | 4.46 | | | 91.56 | | | 61.28 | | | 0.1 | | | 0.41 | | | 9.9 | |
| 5.828 | | 5.566 | | | 3.8 | | | 89.33 | | | 59.33 | | | 0.1 | | | 0.41 | | | 9.89 | |
| 5.73 | | 5.463 | | | 3.01 | | | 91.52 | | | 63.91 | | | 0.1 | | | 0.41 | | | 9.89 | |
| 5.8249 | | 5.4647 | | | 3.663 | | | 91.708 | | | 60.367 | | | 0.1 | | | 0.407 | | | 9.898 | |
| 0.1536912 | | 0.059211579 | | | 0.533563492 | | | 1.18891921 | | | 9.099992735 | | | 0 | | | 0.004830459 | | | 0.018737959 | |
| Load Balance 10 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 5.654 | | 5.346 | | | 3.11 | | | 93.25 | | | 62.07 | | | 0.1 | | | 0.4 | | | 9.76 | |
| 6.585 | | 5.709 | | | 2.65 | | | 88.23 | | | 63.75 | | | 0.1 | | | 0.4 | | | 9.61 | |
| 5.618 | | 5.314 | | | 3.08 | | | 93.24 | | | 47.63 | | | 0.1 | | | 0.41 | | | 9.69 | |
| 5.87 | | 5.521 | | | 4.35 | | | 91.09 | | | 66.81 | | | 0.1 | | | 0.4 | | | 9.69 | |
| 5.703 | | 5.354 | | | 3.06 | | | 91.84 | | | 53.26 | | | 0.1 | | | 0.41 | | | 9.7 | |
| 5.701 | | 5.385 | | | 2.9 | | | 91.57 | | | 53.46 | | | 0.1 | | | 0.41 | | | 9.69 | |
| 5.86 | | 5.544 | | | 3.13 | | | 88.49 | | | 70.73 | | | 0.1 | | | 0.41 | | | 9.7 | |
| 5.634 | | 5.302 | | | 2.87 | | | 92.95 | | | 58.74 | | | 0.1 | | | 0.4 | | | 9.69 | |
| 5.721 | | 5.335 | | | 3.64 | | | 92.62 | | | 38.61 | | | 0.1 | | | 0.41 | | | 9.7 | |
| 5.697 | | 5.39 | | | 2.88 | | | 92.62 | | | 64.67 | | | 0.1 | | | 0.4 | | | 9.71 | |
| 5.8043 | | 5.42 | | | 3.167 | | | 91.59 | | | 57.973 | | | 0.1 | | | 0.405 | | | 9.694 | |
| 0.2872467 | | 0.130571394 | | | 0.489445264 | | | 1.845775958 | | | 9.780444264 | | | 0 | | | 0.005270463 | | | 0.036270588 | |
| Load Balance 10 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 5.826 | | 5.395 | | | 3.47 | | | 91.89 | | | 52.21 | | | 0.1 | | | 0.4 | | | 9.67 | |
| 5.8 | | 5.543 | | | 3.3 | | | 89.56 | | | 65.66 | | | 0.1 | | | 0.41 | | | 9.68 | |
| 5.696 | | 5.381 | | | 3.49 | | | 94.18 | | | 52.68 | | | 0.1 | | | 0.4 | | | 9.7 | |
| 5.771 | | 5.465 | | | 5.59 | | | 91.63 | | | 63.3 | | | 0.09 | | | 0.41 | | | 9.69 | |
| 5.581 | | 5.298 | | | 2.89 | | | 93.7 | | | 48.51 | | | 0.09 | | | 0.4 | | | 9.73 | |
| 5.608 | | 5.152 | | | 3.92 | | | 94.61 | | | 50.2 | | | 0.1 | | | 0.4 | | | 9.73 | |
| 5.653 | | 5.329 | | | 3.2 | | | 93.62 | | | 43.65 | | | 0.1 | | | 0.4 | | | 9.73 | |
| 5.684 | | 5.324 | | | 3.15 | | | 91.76 | | | 58.03 | | | 0.1 | | | 0.4 | | | 9.73 | |
| 5.698 | | 5.392 | | | 3.44 | | | 92.16 | | | 50.53 | | | 0.1 | | | 0.4 | | | 9.73 | |
| 5.659 | | 5.372 | | | 3.03 | | | 92.1 | | | 59.16 | | | 0.1 | | | 0.41 | | | 9.73 | |
| 5.6976 | | 5.3651 | | | 3.548 | | | 92.521 | | | 54.393 | | | 0.098 | | | 0.403 | | | 9.712 | |
| 0.080035 | | 0.103522891 | | | 0.772554205 | | | 1.511448533 | | | 6.931871721 | | | 0.00421637 | | | 0.004830459 | | | 0.024404007 | |
| Load Balance 10 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 5.664 | | 5.289 | | | 3.9 | | | 93.18 | | | 51.18 | | | 0.1 | | | 0.41 | | | 9.7 | |
| 5.78 | | 5.436 | | | 3.12 | | | 91.95 | | | 81.84 | | | 0.1 | | | 0.42 | | | 9.72 | |
| 5.588 | | 5.235 | | | 3.54 | | | 93.08 | | | 53.67 | | | 0.1 | | | 0.41 | | | 9.74 | |
| 5.617 | | 5.34 | | | 3.16 | | | 92.81 | | | 63.73 | | | 0.1 | | | 0.41 | | | 9.72 | |
| 5.58 | | 5.308 | | | 3.16 | | | 93.58 | | | 44.46 | | | 0.1 | | | 0.42 | | | 9.72 | |
| 5.646 | | 5.314 | | | 3.35 | | | 92.7 | | | 68.11 | | | 0.1 | | | 0.42 | | | 9.72 | |
| 5.555 | | 5.187 | | | 3.73 | | | 93.52 | | | 78.58 | | | 0.1 | | | 0.41 | | | 9.71 | |
| 5.676 | | 5.374 | | | 3.08 | | | 92.9 | | | 51.2 | | | 0.1 | | | 0.42 | | | 9.72 | |
| 5.659 | | 5.387 | | | 3.22 | | | 92.36 | | | 50.45 | | | 0.1 | | | 0.42 | | | 9.71 | |
| 5.624 | | 5.257 | | | 3.39 | | | 93.9 | | | 40 | | | 0.09 | | | 0.41 | | | 9.71 | |
| 5.6389 | | 5.3127 | | | 3.365 | | | 92.998 | | | 58.322 | | | 0.099 | | | 0.415 | | | 9.717 | |
| 0.0633955 | | 0.075094089 | | | 0.278338323 | | | 0.58724971 | | | 14.14891264 | | | 0.003162278 | | | 0.005270463 | | | 0.010593499 | |
| Load Balance 10 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 5.61 | | 5.274 | | | 3.24 | | | 93.09 | | | 51.1 | | | 0.09 | | | 0.41 | | | 9.69 | |
| 5.705 | | 5.492 | | | 3.35 | | | 90.64 | | | 74.35 | | | 0.1 | | | 0.42 | | | 9.71 | |
| 5.572 | | 5.262 | | | 3.42 | | | 93.32 | | | 62.33 | | | 0.1 | | | 0.41 | | | 9.71 | |
| 5.578 | | 5.253 | | | 3.67 | | | 93.94 | | | 52.27 | | | 0.1 | | | 0.42 | | | 9.71 | |
| 5.699 | | 5.439 | | | 3.55 | | | 93.35 | | | 66.91 | | | 0.1 | | | 0.42 | | | 9.71 | |
| 6.029 | | 5.79 | | | 3.16 | | | 86.7 | | | 64.61 | | | 0.09 | | | 0.42 | | | 9.71 | |
| 5.553 | | 5.273 | | | 2.82 | | | 93.97 | | | 47.25 | | | 0.09 | | | 0.42 | | | 9.72 | |
| 5.721 | | 5.331 | | | 2.79 | | | 93.09 | | | 61.25 | | | 0.09 | | | 0.41 | | | 9.7 | |
| 5.935 | | 5.488 | | | 3.22 | | | 90.4 | | | 71.79 | | | 0.1 | | | 0.42 | | | 9.71 | |
| 5.677 | | 5.31 | | | 3.47 | | | 92.69 | | | 47.12 | | | 0.1 | | | 0.41 | | | 9.71 | |
| 5.7079 | | 5.3912 | | | 3.269 | | | 92.119 | | | 59.898 | | | 0.096 | | | 0.416 | | | 9.708 | |
| 0.1580488 | | 0.16818429 | | | 0.289768107 | | | 2.271066563 | | | 9.934418062 | | | 0.005163978 | | | 0.005163978 | | | 0.007888106 | |
| Load Balance 10 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 6.428 | | 5.387 | | | 3.99 | | | 91.94 | | | 32.79 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 5.657 | | 5.331 | | | 3.6 | | | 93.68 | | | 45.54 | | | 0.11 | | | 0.43 | | | 9.93 | |
| 5.614 | | 5.242 | | | 4.21 | | | 93.8 | | | 43.78 | | | 0.11 | | | 0.42 | | | 9.93 | |
| 5.629 | | 5.138 | | | 2.98 | | | 93.85 | | | 54.91 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 5.774 | | 5.404 | | | 2.97 | | | 91.81 | | | 43.33 | | | 0.1 | | | 0.42 | | | 9.91 | |
| 5.623 | | 5.23 | | | 2.73 | | | 93.5 | | | 67.41 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 5.785 | | 5.413 | | | 3.94 | | | 93.47 | | | 61.78 | | | 0.1 | | | 0.42 | | | 9.94 | |
| 5.609 | | 5.254 | | | 3.74 | | | 94.29 | | | 39.39 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 5.652 | | 5.211 | | | 4.13 | | | 93.43 | | | 51.36 | | | 0.11 | | | 0.42 | | | 9.93 | |
| 5.685 | | 5.289 | | | 3.71 | | | 94.43 | | | 35.3 | | | 0.11 | | | 0.43 | | | 9.93 | |
| 5.7456 | | 5.2899 | | | 3.6 | | | 93.42 | | | 47.559 | | | 0.104 | | | 0.422 | | | 9.929 | |
| 0.2478612 | | 0.091748993 | | | 0.526054497 | | | 0.879734809 | | | 11.23567973 | | | 0.005163978 | | | 0.00421637 | | | 0.007378648 | |
| Load Balance 10 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 5.577 | | 5.259 | | | 3.55 | | | 93.65 | | | 61.43 | | | 0.1 | | | 0.42 | | | 9.87 | |
| 5.611 | | 5.222 | | | 2.83 | | | 94.56 | | | 49.24 | | | 0.1 | | | 0.42 | | | 9.82 | |
| 5.711 | | 5.335 | | | 3.42 | | | 93.24 | | | 44.95 | | | 0.1 | | | 0.42 | | | 9.82 | |
| 5.621 | | 5.314 | | | 2.4 | | | 92.99 | | | 61.59 | | | 0.1 | | | 0.43 | | | 9.75 | |
| 5.585 | | 5.26 | | | 3.4 | | | 94.26 | | | 49.19 | | | 0.1 | | | 0.42 | | | 9.83 | |
| 5.621 | | 5.229 | | | 2.89 | | | 94.87 | | | 55.21 | | | 0.1 | | | 0.42 | | | 9.85 | |
| 5.534 | | 5.158 | | | 2.71 | | | 94.6 | | | 58.75 | | | 0.1 | | | 0.42 | | | 9.85 | |
| 5.686 | | 5.322 | | | 3.54 | | | 92.76 | | | 71.68 | | | 0.1 | | | 0.43 | | | 9.84 | |
| 5.634 | | 5.285 | | | 3.45 | | | 93.31 | | | 53.52 | | | 0.1 | | | 0.42 | | | 9.84 | |
| 5.646 | | 5.198 | | | 2.64 | | | 93.85 | | | 57.24 | | | 0.1 | | | 0.42 | | | 9.84 | |
| 5.6226 | | 5.2582 | | | 3.083 | | | 93.809 | | | 56.28 | | | 0.1 | | | 0.422 | | | 9.831 | |
| 0.0516338 | | 0.057379439 | | | 0.431896335 | | | 0.737088115 | | | 7.716232529 | | | 0 | | | 0.00421637 | | | 0.032128215 | |
| Load Balance 10 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 5.752 | | 5.32 | | | 3.12 | | | 91.78 | | | 50.08 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 5.663 | | 5.335 | | | 3.29 | | | 92.82 | | | 62.98 | | | 0.1 | | | 0.42 | | | 9.94 | |
| 5.597 | | 5.156 | | | 3.37 | | | 93.97 | | | 30.55 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 7.447 | | 6.167 | | | 2.7 | | | 81.32 | | | 48.05 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 7.527 | | 6.257 | | | 3.55 | | | 79.93 | | | 43.4 | | | 0.1 | | | 0.43 | | | 9.93 | |
| 5.69 | | 5.29 | | | 3.56 | | | 93.74 | | | 38.63 | | | 0.1 | | | 0.42 | | | 9.94 | |
| 5.509 | | 5.165 | | | 2.98 | | | 95.75 | | | 48.74 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 5.643 | | 5.282 | | | 3.31 | | | 93.5 | | | 57.37 | | | 0.1 | | | 0.42 | | | 9.92 | |
| 5.677 | | 5.324 | | | 3.47 | | | 93.85 | | | 56.98 | | | 0.1 | | | 0.43 | | | 9.92 | |
| 5.643 | | 5.292 | | | 2.58 | | | 93.1 | | | 56.97 | | | 0.1 | | | 0.42 | | | 9.94 | |
| 6.0148 | | 5.4588 | | | 3.193 | | | 90.976 | | | 49.375 | | | 0.1 | | | 0.422 | | | 9.931 | |
| 0.778691 | | 0.402356061 | | | 0.343642256 | | | 5.556136147 | | | 9.847324792 | | | 0 | | | 0.00421637 | | | 0.007378648 | |
| Load Balance 10 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 6.152 | | 5.794 | | | 3.21 | | | 86.92 | | | 56.01 | | | 0.1 | | | 0.43 | | | 9.9 | |
| 7.706 | | 6.489 | | | 3.01 | | | 76.82 | | | 33.75 | | | 0.1 | | | 0.43 | | | 9.91 | |
| 6.236 | | 5.822 | | | 3.1 | | | 58.72 | | | 62.91 | | | 0.1 | | | 0.43 | | | 9.92 | |
| 7.153 | | 5.982 | | | 3.29 | | | 83.4 | | | 49.57 | | | 0.1 | | | 0.42 | | | 9.93 | |
| 6 | | 5.38 | | | 3.02 | | | 89.52 | | | 45.87 | | | 0.1 | | | 0.42 | | | 9.91 | |
| 5.789 | | 5.341 | | | 3.71 | | | 92.29 | | | 72.67 | | | 0.1 | | | 0.42 | | | 9.92 | |
| 6.144 | | 5.785 | | | 2.69 | | | 86.66 | | | 58.2 | | | 0.1 | | | 0.43 | | | 9.91 | |
| 5.621 | | 5.173 | | | 2.9 | | | 93.78 | | | 34.21 | | | 0.09 | | | 0.42 | | | 9.92 | |
| 6.13 | | 5.724 | | | 2.99 | | | 86.25 | | | 46.05 | | | 0.1 | | | 0.43 | | | 9.92 | |
| 5.656 | | 5.282 | | | 3.26 | | | 93.53 | | | 43.02 | | | 0.1 | | | 0.42 | | | 9.92 | |
| 6.2587 | | 5.6772 | | | 3.118 | | | 84.789 | | | 50.226 | | | 0.099 | | | 0.425 | | | 9.916 | |
| 0.6661266 | | 0.396106271 | | | 0.274339611 | | | 10.49388123 | | | 12.37559265 | | | 0.003162278 | | | 0.005270463 | | | 0.00843274 | |
| Load Balance 10 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 5.653 | | 5.309 | | | 2.67 | | | 93.77 | | | 45.88 | | | 0.1 | | | 0.43 | | | 9.86 | |
| 6.026 | | 5.663 | | | 2.72 | | | 88.18 | | | 63.37 | | | 0.1 | | | 0.43 | | | 9.89 | |
| 5.538 | | 5.121 | | | 3.38 | | | 94.1 | | | 40.79 | | | 0.1 | | | 0.42 | | | 9.89 | |
| 5.601 | | 5.209 | | | 3.94 | | | 93.78 | | | 67.42 | | | 0.09 | | | 0.42 | | | 9.9 | |
| 5.76 | | 5.404 | | | 3.13 | | | 91.99 | | | 39.32 | | | 0.11 | | | 0.43 | | | 9.9 | |
| 5.814 | | 5.398 | | | 3.15 | | | 92.29 | | | 46.11 | | | 0.1 | | | 0.43 | | | 9.89 | |
| 5.467 | | 5.147 | | | 3.12 | | | 94.94 | | | 50.31 | | | 0.1 | | | 0.42 | | | 9.9 | |
| 5.595 | | 5.268 | | | 3.56 | | | 93.56 | | | 43.05 | | | 0.1 | | | 0.42 | | | 9.84 | |
| 5.559 | | 5.237 | | | 4.08 | | | 93.7 | | | 46.05 | | | 0.1 | | | 0.42 | | | 9.84 | |
| 5.605 | | 5.26 | | | 2.8 | | | 93.63 | | | 60.78 | | | 0.1 | | | 0.42 | | | 9.87 | |
| 5.6618 | | 5.3016 | | | 3.255 | | | 92.994 | | | 50.308 | | | 0.1 | | | 0.424 | | | 9.878 | |
| 0.1635555 | | 0.157257185 | | | 0.48749359 | | | 1.889492583 | | | 9.958062953 | | | 0.004714045 | | | 0.005163978 | | | 0.02394438 | |
| Run 1 |  | | |  | | |  | |  | | |  | | |  | | |  | | |  | |
| Iteration | Load Balance 30 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | | Avg Request Time | | | Avg Local CPU | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 1 | 8.884 | | | 6.164 | | | 6.74 | | 6.06 | | | 43.74 | | | 0.11 | | | 0.17 | | | 8.96 | |
| 2 | 8.079 | | | 5.951 | | | 6.76 | | 6.45 | | | 43.03 | | | 0.11 | | | 0.17 | | | 8.97 | |
| 3 | 7.677 | | | 5.396 | | | 6.97 | | 6.71 | | | 44.14 | | | 0.11 | | | 0.17 | | | 8.97 | |
| 4 | 10.336 | | | 7.736 | | | 5.01 | | 4.52 | | | 38.03 | | | 0.11 | | | 0.17 | | | 8.97 | |
| 5 | 6.954 | | | 5.089 | | | 7.72 | | 6.12 | | | 49.79 | | | 0.11 | | | 0.17 | | | 8.96 | |
| 6 | 7.284 | | | 5.337 | | | 7.31 | | 5.71 | | | 47.35 | | | 0.11 | | | 0.17 | | | 8.94 | |
| 7 | 6.777 | | | 5.006 | | | 7.02 | | 5.88 | | | 48.45 | | | 0.1 | | | 0.17 | | | 9.01 | |
| 8 | 7.133 | | | 5.413 | | | 6.61 | | 5.19 | | | 49.58 | | | 0.11 | | | 0.17 | | | 9.02 | |
| 9 | 8.456 | | | 6.304 | | | 6.13 | | 5.49 | | | 39.96 | | | 0.1 | | | 0.17 | | | 9.01 | |
| 10 | 6.712 | | | 4.965 | | | 7 | | 5.98 | | | 53.22 | | | 0.11 | | | 0.17 | | | 9.03 | |
| Avg | 7.8292 | | | 5.7361 | | | 6.727 | | 5.811 | | | 45.729 | | | 0.108 | | | 0.17 | | | 8.984 | |
| SD | 1.1439826 | | | 0.84833909 | | | 0.735210476 | | 0.630069661 | | | 4.758502215 | | | 0.00421637 | | | 0 | | | 0.030623158 | |
| Run 2 |  | | |  | | |  | |  | | |  | | |  | | |  | | |  | |
| Iteration | Load Balance 30 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | | Avg Request Time | | | Avg Local CPU | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 1 | 7.519 | | | 5.36 | | | 9.3 | | 6.92 | | | 46.05 | | | 0.11 | | | 0.17 | | | 8.8 | |
| 2 | 7.43 | | | 5.438 | | | 9.31 | | 6.73 | | | 48.05 | | | 0.11 | | | 0.17 | | | 8.86 | |
| 3 | 8.041 | | | 6.115 | | | 5.46 | | 5.41 | | | 40.79 | | | 0.11 | | | 0.17 | | | 8.83 | |
| 4 | 8.62 | | | 6.239 | | | 6.35 | | 5.73 | | | 46.15 | | | 0.11 | | | 0.17 | | | 8.86 | |
| 5 | 8.663 | | | 6.25 | | | 6.57 | | 5.9 | | | 45.64 | | | 0.11 | | | 0.17 | | | 8.85 | |
| 6 | 7.947 | | | 5.681 | | | 7.49 | | 5.67 | | | 43.5 | | | 0.11 | | | 0.17 | | | 8.83 | |
| 7 | 7.335 | | | 5.133 | | | 7.81 | | 6.08 | | | 43.35 | | | 0.1 | | | 0.16 | | | 8.83 | |
| 8 | 10.061 | | | 7.903 | | | 5.2 | | 3.73 | | | 46.44 | | | 0.11 | | | 0.17 | | | 8.82 | |
| 9 | 8.564 | | | 5.909 | | | 6.67 | | 6.28 | | | 40.9 | | | 0.11 | | | 0.17 | | | 8.8 | |
| 10 | 8.15 | | | 5.917 | | | 6.71 | | 5.59 | | | 43.65 | | | 0.11 | | | 0.17 | | | 8.8 | |
| Avg | 8.233 | | | 5.9945 | | | 7.087 | | 5.804 | | | 44.452 | | | 0.109 | | | 0.169 | | | 8.828 | |
| SD | 0.8077659 | | | 0.770859297 | | | 1.408648288 | | 0.878182466 | | | 2.41607487 | | | 0.003162278 | | | 0.003162278 | | | 0.023475756 | |
| Run 3 |  | | |  | | |  | |  | | |  | | |  | | |  | | |  | |
| Iteration | Load Balance 30 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | | Avg Request Time | | | Avg Local CPU | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 1 | 8.327 | | | 5.73 | | | 6.89 | | 5.96 | | | 42.31 | | | 0.11 | | | 0.17 | | | 8.78 | |
| 2 | 8.291 | | | 5.674 | | | 6.85 | | 5.78 | | | 41.59 | | | 0.11 | | | 0.17 | | | 8.79 | |
| 3 | 9.77 | | | 7.088 | | | 6.05 | | 4.8 | | | 41.28 | | | 0.11 | | | 0.16 | | | 8.87 | |
| 4 | 9.23 | | | 6.887 | | | 6.15 | | 4.78 | | | 46.19 | | | 0.11 | | | 0.17 | | | 8.77 | |
| 5 | 7.823 | | | 5.519 | | | 7.27 | | 6.57 | | | 44.27 | | | 0.1 | | | 0.17 | | | 8.78 | |
| 6 | 8.238 | | | 5.943 | | | 6.82 | | 5.66 | | | 45.17 | | | 0.11 | | | 0.16 | | | 8.77 | |
| 7 | 10.61 | | | 8.348 | | | 5.14 | | 3.91 | | | 42.02 | | | 0.11 | | | 0.17 | | | 8.76 | |
| 8 | 8.599 | | | 6.415 | | | 6.45 | | 5.26 | | | 41.08 | | | 0.1 | | | 0.17 | | | 8.76 | |
| 9 | 7.518 | | | 5.223 | | | 7.98 | | 6.29 | | | 41.49 | | | 0.11 | | | 0.17 | | | 8.77 | |
| 10 | 8.28 | | | 6.124 | | | 5.63 | | 5.52 | | | 43.46 | | | 0.1 | | | 0.16 | | | 8.76 | |
| Avg | 8.6686 | | | 6.2951 | | | 6.523 | | 5.453 | | | 42.886 | | | 0.107 | | | 0.167 | | | 8.781 | |
| SD | 0.9390745 | | | 0.932016506 | | | 0.822530648 | | 0.792815657 | | | 1.792857434 | | | 0.004830459 | | | 0.004830459 | | | 0.032812599 | |
| Run 4 |  | | |  | | |  | |  | | |  | | |  | | |  | | |  | |
| Iteration | Load Balance 30 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | | Avg Request Time | | | Avg Local CPU | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 1 | 7.54 | | | 5.296 | | | 7.46 | | 6.95 | | | 50.22 | | | 0.11 | | | 0.17 | | | 8.68 | |
| 2 | 7.82 | | | 5.688 | | | 6.84 | | 6.34 | | | 43.43 | | | 0.11 | | | 0.16 | | | 8.7 | |
| 3 | 7.065 | | | 5.04 | | | 6.55 | | 6.18 | | | 49.73 | | | 0.11 | | | 0.16 | | | 8.69 | |
| 4 | 8.507 | | | 6.134 | | | 6.77 | | 5.47 | | | 43.54 | | | 0.11 | | | 0.16 | | | 8.68 | |
| 5 | 7.46 | | | 5.114 | | | 6.07 | | 6.27 | | | 44.05 | | | 0.1 | | | 0.16 | | | 8.67 | |
| 6 | 7.869 | | | 5.785 | | | 6.67 | | 5.79 | | | 51.22 | | | 0.11 | | | 0.17 | | | 8.66 | |
| 7 | 8.184 | | | 5.853 | | | 5.73 | | 6 | | | 45.9 | | | 0.1 | | | 0.16 | | | 8.65 | |
| 8 | 9.852 | | | 7.292 | | | 4.74 | | 4.63 | | | 40.28 | | | 0.1 | | | 0.16 | | | 8.65 | |
| 9 | 7.95 | | | 5.726 | | | 7.29 | | 6.87 | | | 47.09 | | | 0.1 | | | 0.16 | | | 8.65 | |
| 10 | 8.028 | | | 5.741 | | | 6.45 | | 6.82 | | | 48.22 | | | 0.1 | | | 0.17 | | | 8.65 | |
| Avg | 8.0275 | | | 5.7669 | | | 6.457 | | 6.132 | | | 46.368 | | | 0.105 | | | 0.163 | | | 8.668 | |
| SD | 0.7559774 | | | 0.637129927 | | | 0.790570118 | | 0.713392521 | | | 3.542117383 | | | 0.005270463 | | | 0.004830459 | | | 0.018737959 | |
| Run 5 |  | | |  | | |  | |  | | |  | | |  | | |  | | |  | |
| Iteration | Load Balance 30 requests - no pre-process | | | | | | | | | | | | | | | | | | | | | |
|  | Total Time | | | Avg Request Time | | | Avg Local CPU | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 1 | 8.037 | | | 5.586 | | | 6.63 | | 6.22 | | | 40.2 | | | 0.11 | | | 0.16 | | | 8.59 | |
| 2 | 8.702 | | | 6.357 | | | 6.55 | | 5.36 | | | 43.55 | | | 0.11 | | | 0.16 | | | 8.61 | |
| 3 | 8.763 | | | 6.351 | | | 5.79 | | 5.18 | | | 38.04 | | | 0.1 | | | 0.16 | | | 8.61 | |
| 4 | 9.805 | | | 7.281 | | | 6.09 | | 4.63 | | | 41.17 | | | 0.1 | | | 0.16 | | | 8.54 | |
| 5 | 8.663 | | | 5.915 | | | 7.21 | | 6.89 | | | 40.52 | | | 0.1 | | | 0.16 | | | 8.61 | |
| 6 | 8.968 | | | 6.36 | | | 6.03 | | 5.77 | | | 35.06 | | | 0.1 | | | 0.16 | | | 8.6 | |
| 7 | 9.846 | | | 7.022 | | | 6.08 | | 5.6 | | | 35.88 | | | 0.11 | | | 0.17 | | | 8.66 | |
| 8 | 8.93 | | | 6.224 | | | 6.81 | | 5.95 | | | 39.35 | | | 0.1 | | | 0.17 | | | 8.65 | |
| 9 | 8.079 | | | 5.511 | | | 6.72 | | 6.01 | | | 37.63 | | | 0.1 | | | 0.17 | | | 8.64 | |
| 10 | 7.772 | | | 5.512 | | | 6.78 | | 6.15 | | | 43.86 | | | 0.1 | | | 0.17 | | | 8.63 | |
| Avg | 8.7565 | | | 6.2119 | | | 6.469 | | 5.776 | | | 39.526 | | | 0.103 | | | 0.164 | | | 8.614 | |
| SD | 0.6927304 | | | 0.607612898 | | | 0.447994296 | | 0.626776586 | | | 2.947572259 | | | 0.004830459 | | | 0.005163978 | | | 0.034383459 | |
| Load Balance 30 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 16.358 | | 14.795 | | | 3.68 | | | 95.54 | | | 54.16 | | | 0.11 | | | 0.14 | | | 8.95 | |
| 16.389 | | 14.615 | | | 3.38 | | | 95.57 | | | 53.89 | | | 0.11 | | | 0.13 | | | 8.96 | |
| 16.378 | | 14.757 | | | 3.58 | | | 95.95 | | | 58.01 | | | 0.11 | | | 0.13 | | | 8.9 | |
| 16.23 | | 14.719 | | | 3.45 | | | 96.07 | | | 52.21 | | | 0.11 | | | 0.14 | | | 8.93 | |
| 16.177 | | 14.536 | | | 3.27 | | | 96.32 | | | 58.73 | | | 0.11 | | | 0.13 | | | 8.92 | |
| 16.189 | | 14.443 | | | 4.02 | | | 96.24 | | | 66.64 | | | 0.11 | | | 0.13 | | | 8.93 | |
| 16.175 | | 14.515 | | | 3.51 | | | 96.44 | | | 47.69 | | | 0.11 | | | 0.13 | | | 8.92 | |
| 17.339 | | 15.092 | | | 3.65 | | | 93.63 | | | 54.09 | | | 0.11 | | | 0.14 | | | 8.89 | |
| 16.191 | | 14.667 | | | 3.13 | | | 96.04 | | | 51.46 | | | 0.11 | | | 0.13 | | | 8.89 | |
| 16.365 | | 14.613 | | | 3.83 | | | 95.44 | | | 53.6 | | | 0.11 | | | 0.14 | | | 8.82 | |
| 16.3791 | | 14.6752 | | | 3.55 | | | 95.724 | | | 55.048 | | | 0.11 | | | 0.134 | | | 8.911 | |
| 0.3493491 | | 0.183545816 | | | 0.262974439 | | | 0.811571863 | | | 5.130704522 | | | 0 | | | 0.005163978 | | | 0.039567102 | |
| Load Balance 30 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 16.303 | | 14.686 | | | 3.57 | | | 95.95 | | | 51.72 | | | 0.11 | | | 0.13 | | | 8.89 | |
| 16.231 | | 14.757 | | | 3.17 | | | 95.99 | | | 54.48 | | | 0.11 | | | 0.13 | | | 8.88 | |
| 16.566 | | 14.81 | | | 3.57 | | | 95.59 | | | 51.4 | | | 0.11 | | | 0.13 | | | 8.78 | |
| 16.603 | | 14.683 | | | 3.28 | | | 95.49 | | | 49.88 | | | 0.11 | | | 0.13 | | | 8.81 | |
| 17.094 | | 15.07 | | | 3.03 | | | 95.15 | | | 49.22 | | | 0.11 | | | 0.14 | | | 8.85 | |
| 16.945 | | 15.078 | | | 3.59 | | | 93.39 | | | 54.99 | | | 0.11 | | | 0.15 | | | 8.84 | |
| 16.645 | | 15.064 | | | 3.87 | | | 94.73 | | | 46.96 | | | 0.11 | | | 0.14 | | | 8.84 | |
| 16.262 | | 14.62 | | | 4.5 | | | 96.23 | | | 51.79 | | | 0.11 | | | 0.13 | | | 8.83 | |
| 16.667 | | 15.001 | | | 3.07 | | | 94.89 | | | 51.77 | | | 0.11 | | | 0.14 | | | 8.83 | |
| 17.737 | | 15.177 | | | 3.1 | | | 92.44 | | | 51.79 | | | 0.11 | | | 0.14 | | | 8.83 | |
| 16.7053 | | 14.8946 | | | 3.475 | | | 94.985 | | | 51.4 | | | 0.11 | | | 0.136 | | | 8.838 | |
| 0.4584081 | | 0.203838825 | | | 0.456027533 | | | 1.212648616 | | | 2.349808503 | | | 0 | | | 0.006992059 | | | 0.031552426 | |
| Load Balance 30 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 16.382 | | 16.748 | | | 3.03 | | | 95.6 | | | 54.04 | | | 0.1 | | | 0.13 | | | 8.76 | |
| 18.757 | | 16.378 | | | 3.52 | | | 87.34 | | | 55.68 | | | 0.11 | | | 0.15 | | | 8.76 | |
| 16.832 | | 15.034 | | | 2.74 | | | 94.06 | | | 50.56 | | | 0.11 | | | 0.15 | | | 8.7 | |
| 16.295 | | 14.467 | | | 3.52 | | | 95.65 | | | 48.48 | | | 0.11 | | | 0.13 | | | 8.75 | |
| 16.385 | | 14.712 | | | 3.32 | | | 95.49 | | | 43.69 | | | 0.11 | | | 0.13 | | | 8.73 | |
| 16.503 | | 14.698 | | | 3.44 | | | 95.68 | | | 58.52 | | | 0.1 | | | 0.13 | | | 8.73 | |
| 20.538 | | 16.408 | | | 3.46 | | | 87.67 | | | 40.51 | | | 0.11 | | | 0.14 | | | 8.74 | |
| 25.465 | | 20.267 | | | 3.07 | | | 74 | | | 44.83 | | | 0.11 | | | 0.16 | | | 8.73 | |
| 16.424 | | 14.661 | | | 3.5 | | | 95.67 | | | 52.03 | | | 0.1 | | | 0.13 | | | 8.73 | |
| 16.525 | | 14.935 | | | 3.62 | | | 94.83 | | | 46.5 | | | 0.11 | | | 0.14 | | | 8.74 | |
| 18.0106 | | 15.8308 | | | 3.322 | | | 91.599 | | | 49.484 | | | 0.107 | | | 0.139 | | | 8.737 | |
| 2.9653068 | | 1.774961458 | | | 0.282716977 | | | 6.99668882 | | | 5.71656307 | | | 0.004830459 | | | 0.011005049 | | | 0.017669811 | |
| Load Balance 30 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 16.068 | | 14.37 | | | 6.17 | | | 96.5 | | | 58.67 | | | 0.1 | | | 0.13 | | | 8.65 | |
| 16.075 | | 16.616 | | | 3.51 | | | 96.38 | | | 49.35 | | | 0.11 | | | 0.13 | | | 8.65 | |
| 16.709 | | 14.955 | | | 3.88 | | | 94.18 | | | 54.2 | | | 0.11 | | | 0.14 | | | 8.64 | |
| 16.806 | | 15.075 | | | 3.63 | | | 93.55 | | | 47.27 | | | 0.11 | | | 0.14 | | | 8.64 | |
| 16.227 | | 14.761 | | | 3.31 | | | 96.11 | | | 57.01 | | | 0.1 | | | 0.13 | | | 8.63 | |
| 16.153 | | 14.624 | | | 3.22 | | | 96.3 | | | 58.59 | | | 0.11 | | | 0.12 | | | 8.63 | |
| 16.261 | | 14.723 | | | 3.69 | | | 96.06 | | | 53.28 | | | 0.1 | | | 0.13 | | | 8.56 | |
| 17.03 | | 14.881 | | | 3.54 | | | 95.58 | | | 48.18 | | | 0.11 | | | 0.13 | | | 8.63 | |
| 16.77 | | 15.134 | | | 3.55 | | | 94.04 | | | 65.71 | | | 0.1 | | | 0.14 | | | 8.61 | |
| 16.255 | | 14.757 | | | 3.63 | | | 95.72 | | | 51.89 | | | 0.11 | | | 0.13 | | | 8.62 | |
| 16.4354 | | 14.9896 | | | 3.813 | | | 95.442 | | | 54.415 | | | 0.106 | | | 0.132 | | | 8.626 | |
| 0.3541372 | | 0.612945203 | | | 0.897566154 | | | 1.095331507 | | | 5.707083805 | | | 0.005163978 | | | 0.006324555 | | | 0.026331224 | |
| Load Balance 30 requests - pre-process (w/o load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 17.334 | | 15.614 | | | 3.55 | | | 92.02 | | | 57.59 | | | 0.11 | | | 0.14 | | | 8.53 | |
| 16.694 | | 15.045 | | | 3.4 | | | 94.05 | | | 54.39 | | | 0.1 | | | 0.14 | | | 8.56 | |
| 16.169 | | 14.42 | | | 3.53 | | | 96.57 | | | 51.16 | | | 0.11 | | | 0.12 | | | 8.56 | |
| 16.7 | | 15.083 | | | 3.64 | | | 94.23 | | | 61.02 | | | 0.1 | | | 0.14 | | | 8.54 | |
| 18.202 | | 15.732 | | | 2.98 | | | 89.96 | | | 48.93 | | | 0.11 | | | 0.13 | | | 8.52 | |
| 16.743 | | 15.032 | | | 2.94 | | | 94.38 | | | 49.11 | | | 0.1 | | | 0.13 | | | 8.66 | |
| 16.087 | | 14.323 | | | 3.38 | | | 96.69 | | | 56.37 | | | 0.11 | | | 0.13 | | | 8.59 | |
| 16.627 | | 14.796 | | | 3.22 | | | 95.29 | | | 46.67 | | | 0.1 | | | 0.13 | | | 8.58 | |
| 17.055 | | 14.789 | | | 3.67 | | | 96.21 | | | 59.77 | | | 0.11 | | | 0.13 | | | 8.57 | |
| 16.967 | | 14.504 | | | 3.43 | | | 96.01 | | | 49.21 | | | 0.1 | | | 0.13 | | | 8.56 | |
| 16.8578 | | 14.9338 | | | 3.374 | | | 94.541 | | | 53.422 | | | 0.105 | | | 0.132 | | | 8.567 | |
| 0.6028829 | | 0.472089398 | | | 0.254916108 | | | 2.158813717 | | | 5.079863953 | | | 0.005270463 | | | 0.006324555 | | | 0.039171985 | |
| Load Balance 30 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 12.673 | | 8.29 | | | 5.74 | | | 89.76 | | | 57.68 | | | 0.11 | | | 0.11 | | | 8.52 | |
| 13.369 | | 10.275 | | | 4.29 | | | 92.69 | | | 52.4 | | | 0.11 | | | 0.11 | | | 8.48 | |
| 16.411 | | 14.702 | | | 3.35 | | | 94.65 | | | 52.44 | | | 0.11 | | | 0.11 | | | 8.52 | |
| 9.832 | | 6.822 | | | 6.82 | | | 94.55 | | | 51.84 | | | 0.11 | | | 0.11 | | | 8.49 | |
| 12.41 | | 7.421 | | | 5.55 | | | 90.86 | | | 48.08 | | | 0.11 | | | 0.11 | | | 8.49 | |
| 10.824 | | 6.54 | | | 5.72 | | | 90.16 | | | 49.5 | | | 0.11 | | | 0.11 | | | 8.48 | |
| 13.033 | | 9.581 | | | 4.96 | | | 92 | | | 57.81 | | | 0.11 | | | 0.11 | | | 8.47 | |
| 13.963 | | 11.065 | | | 4.6 | | | 87 | | | 49.92 | | | 0.1 | | | 0.12 | | | 8.51 | |
| 13.304 | | 9.609 | | | 5.06 | | | 93.81 | | | 52.26 | | | 0.11 | | | 0.11 | | | 8.45 | |
| 15.346 | | 11.803 | | | 4.03 | | | 89.29 | | | 43.99 | | | 0.11 | | | 0.11 | | | 8.5 | |
| 13.1165 | | 9.6108 | | | 5.012 | | | 89.29 | | | 51.592 | | | 0.109 | | | 0.111 | | | 8.491 | |
| 1.9260258 | | 2.51726844 | | | 1.000675328 | | | 2.507517808 | | | 4.148675291 | | | 0.003162278 | | | 0.003162278 | | | 0.022335821 | |
| Load Balance 30 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 12.817 | | 9.3 | | | 4.81 | | | 91.62 | | | 57.37 | | | 0.1 | | | 0.11 | | | 8.36 | |
| 8.8 | | 6.112 | | | 6.97 | | | 94.84 | | | 48.94 | | | 0.11 | | | 0.11 | | | 8.34 | |
| 13.241 | | 9.431 | | | 5.69 | | | 90.16 | | | 49 | | | 0.11 | | | 0.11 | | | 8.36 | |
| 12.474 | | 8.976 | | | 5.88 | | | 92.85 | | | 65.83 | | | 0.1 | | | 0.1 | | | 8.39 | |
| 11.806 | | 8.072 | | | 6.16 | | | 91.8 | | | 56.38 | | | 0.1 | | | 0.11 | | | 8.42 | |
| 11.345 | | 7.93 | | | 6.45 | | | 93.45 | | | 52.32 | | | 0.1 | | | 0.1 | | | 8.38 | |
| 8.956 | | 6.775 | | | 6.16 | | | 94.11 | | | 56.9 | | | 0.11 | | | 0.11 | | | 8.38 | |
| 13.861 | | 10.974 | | | 4.24 | | | 93.52 | | | 57.97 | | | 0.1 | | | 0.11 | | | 8.39 | |
| 11.689 | | 8.321 | | | 5.93 | | | 93.38 | | | 52.75 | | | 0.11 | | | 0.1 | | | 8.38 | |
| 11.091 | | 7.774 | | | 5.81 | | | 93.58 | | | 46.89 | | | 0.1 | | | 0.11 | | | 8.37 | |
| 11.608 | | 8.3665 | | | 5.81 | | | 92.931 | | | 54.435 | | | 0.104 | | | 0.107 | | | 8.377 | |
| 1.6751294 | | 1.391288791 | | | 0.781650675 | | | 1.372600533 | | | 5.625235304 | | | 0.005163978 | | | 0.004830459 | | | 0.021628171 | |
| Load Balance 30 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 8.733 | | 6.605 | | | 6.83 | | | 95.27 | | | 48.95 | | | 0.11 | | | 0.11 | | | 8.29 | |
| 13.717 | | 10.41 | | | 5 | | | 93.02 | | | 56.87 | | | 0.1 | | | 0.11 | | | 8.27 | |
| 12.292 | | 8.738 | | | 5.42 | | | 91.93 | | | 51.84 | | | 0.1 | | | 0.11 | | | 8.32 | |
| 16.811 | | 15.409 | | | 3.09 | | | 93.46 | | | 58.54 | | | 0.1 | | | 0.12 | | | 8.33 | |
| 12.308 | | 8.865 | | | 5.17 | | | 93.34 | | | 49.9 | | | 0.11 | | | 0.1 | | | 8.32 | |
| 12.917 | | 9.6 | | | 5.19 | | | 93.75 | | | 55.73 | | | 0.1 | | | 0.1 | | | 9.19 | |
| 10 | | 6.789 | | | 6.13 | | | 93.43 | | | 49.57 | | | 0.11 | | | 0.11 | | | 9.42 | |
| 16.59 | | 15.201 | | | 2.99 | | | 94.9 | | | 58.61 | | | 0.1 | | | 0.11 | | | 9.39 | |
| 11.275 | | 7.17 | | | 5.77 | | | 89.12 | | | 51.96 | | | 0.1 | | | 0.11 | | | 9.33 | |
| 11.083 | | 7.597 | | | 5.84 | | | 93 | | | 47.94 | | | 0.1 | | | 0.1 | | | 9.33 | |
| 12.5726 | | 9.6384 | | | 5.143 | | | 93.122 | | | 52.991 | | | 0.103 | | | 0.108 | | | 8.819 | |
| 2.6022218 | | 3.228845002 | | | 1.231503598 | | | 1.696020178 | | | 4.087934686 | | | 0.004830459 | | | 0.006324555 | | | 0.544210948 | |
| Load Balance 30 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 12.858 | | 9.487 | | | 4.8 | | | 93.3 | | | 51.65 | | | 0.1 | | | 0.1 | | | 9.31 | |
| 11.17 | | 7.351 | | | 6.52 | | | 93.38 | | | 51.5 | | | 0.11 | | | 0.1 | | | 9.29 | |
| 15.224 | | 10.365 | | | 4.36 | | | 90.17 | | | 49.63 | | | 0.1 | | | 0.11 | | | 9.3 | |
| 16.679 | | 15.159 | | | 3.45 | | | 93.64 | | | 52.35 | | | 0.11 | | | 0.12 | | | 9.22 | |
| 11.626 | | 7.959 | | | 6.36 | | | 92.3 | | | 52.25 | | | 0.1 | | | 0.11 | | | 9.27 | |
| 11.982 | | 8.637 | | | 6.23 | | | 93.29 | | | 54.71 | | | 0.11 | | | 0.11 | | | 9.26 | |
| 9.576 | | 6.319 | | | 6.64 | | | 93.08 | | | 51.97 | | | 0.1 | | | 0.1 | | | 9.25 | |
| 10.362 | | 7.373 | | | 5.58 | | | 94.27 | | | 50.15 | | | 0.11 | | | 0.1 | | | 9.26 | |
| 12.241 | | 8.681 | | | 6.12 | | | 92.18 | | | 46.26 | | | 0.1 | | | 0.11 | | | 9.28 | |
| 17.092 | | 15.114 | | | 3.05 | | | 94.1 | | | 68.72 | | | 0.11 | | | 0.11 | | | 9.25 | |
| 12.881 | | 9.6445 | | | 5.311 | | | 92.971 | | | 52.919 | | | 0.105 | | | 0.107 | | | 9.269 | |
| 2.5956424 | | 3.111487149 | | | 1.318277074 | | | 1.188850519 | | | 5.972178925 | | | 0.005270463 | | | 0.006749486 | | | 0.026853512 | |
| Load Balance 30 requests - pre-process (w load balancing) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 14.249 | | 10.835 | | | 4.89 | | | 91.4 | | | 57.91 | | | 0.1 | | | 0.11 | | | 9.09 | |
| 13.342 | | 10.266 | | | 5.01 | | | 94.07 | | | 53.5 | | | 0.11 | | | 0.1 | | | 9.13 | |
| 13.135 | | 9.679 | | | 4.65 | | | 91.95 | | | 54.23 | | | 0.1 | | | 0.1 | | | 9.11 | |
| 12.421 | | 9.072 | | | 5.41 | | | 93.41 | | | 51.13 | | | 0.11 | | | 0.1 | | | 9.12 | |
| 16.731 | | 12.76 | | | 4.55 | | | 87.41 | | | 39.07 | | | 0.1 | | | 0.12 | | | 9.14 | |
| 11.792 | | 8.359 | | | 6.18 | | | 93.81 | | | 43.51 | | | 0.11 | | | 0.11 | | | 9.14 | |
| 7.388 | | 5.597 | | | 7.03 | | | 94.64 | | | 54.79 | | | 0.1 | | | 0.12 | | | 9.13 | |
| 14.22 | | 11.61 | | | 4.38 | | | 94.91 | | | 60.84 | | | 0.1 | | | 0.11 | | | 9.14 | |
| 10.226 | | 6.968 | | | 5.69 | | | 93.08 | | | 49.34 | | | 0.1 | | | 0.11 | | | 9.12 | |
| 12.788 | | 9.513 | | | 5.52 | | | 93.47 | | | 50.96 | | | 0.11 | | | 0.1 | | | 9.11 | |
| 12.6292 | | 9.4659 | | | 5.331 | | | 92.815 | | | 51.528 | | | 0.104 | | | 0.108 | | | 9.123 | |
| 2.5112656 | | 2.123905232 | | | 0.820155134 | | | 2.188415205 | | | 6.446635298 | | | 0.005163978 | | | 0.007888106 | | | 0.016363917 | |
| Load Balance 30 requests - pre-process (w/o load balancing) (3 instances of application) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 8.65 | | 7.15 | | | 6.13 | | | 71.87 | | | 53.83 | | | 0.12 | | | 0.11 | | | 8.04 | |
| 7.677 | | 6.362 | | | 6.58 | | | 80.8 | | | 53.21 | | | 0.12 | | | 0.11 | | | 8.06 | |
| 7.631 | | 6.311 | | | 6.45 | | | 80.9 | | | 63.29 | | | 0.12 | | | 0.11 | | | 8.06 | |
| 9.259 | | 6.514 | | | 6.59 | | | 78.7 | | | 45.99 | | | 0.12 | | | 0.11 | | | 8.11 | |
| 8.699 | | 7.354 | | | 5.39 | | | 70.18 | | | 54.94 | | | 0.12 | | | 0.11 | | | 8.08 | |
| 7.516 | | 6.231 | | | 6.73 | | | 81.91 | | | 52.44 | | | 0.12 | | | 0.11 | | | 8.08 | |
| 7.513 | | 6.171 | | | 6.62 | | | 82.54 | | | 52.39 | | | 0.11 | | | 0.11 | | | 8 | |
| 7.802 | | 6.391 | | | 6.13 | | | 80.33 | | | 60.29 | | | 0.12 | | | 0.11 | | | 8.1 | |
| 8.674 | | 7.206 | | | 4.97 | | | 71.17 | | | 52.29 | | | 0.12 | | | 0.11 | | | 8.07 | |
| 7.547 | | 6.322 | | | 6.14 | | | 80.96 | | | 63.89 | | | 0.12 | | | 0.11 | | | 8.15 | |
| 8.0968 | | 6.6012 | | | 6.173 | | | 77.936 | | | 55.256 | | | 0.119 | | | 0.11 | | | 8.075 | |
| 0.650904 | | 0.450550971 | | | 0.576523104 | | | 4.855322852 | | | 5.593164876 | | | 0.003162278 | | | 0 | | | 0.040620192 | |
| Load Balance 30 requests - pre-process (w/o load balancing) (3 instances of application) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 7.537 | | 6.534 | | | 6.7 | | | 77.94 | | | 50.37 | | | 0.12 | | | 0.11 | | | 7.93 | |
| 9.621 | | 8.8 | | | 5.43 | | | 66.94 | | | 61.58 | | | 0.12 | | | 0.11 | | | 7.96 | |
| 8.725 | | 7.722 | | | 4.93 | | | 65.7 | | | 76.34 | | | 0.11 | | | 0.11 | | | 9.94 | |
| 7.745 | | 6.695 | | | 6.16 | | | 76.34 | | | 73.17 | | | 0.12 | | | 0.11 | | | 9.97 | |
| 7.533 | | 6.579 | | | 6.18 | | | 73.17 | | | 51.26 | | | 0.12 | | | 0.11 | | | 8.02 | |
| 9.564 | | 7.566 | | | 5.26 | | | 51.26 | | | 65.03 | | | 0.11 | | | 0.11 | | | 8 | |
| 10.572 | | 8.528 | | | 5.3 | | | 65.03 | | | 61.45 | | | 0.12 | | | 0.11 | | | 8.03 | |
| 7.787 | | 6.842 | | | 6.76 | | | 61.45 | | | 58.24 | | | 0.12 | | | 0.11 | | | 8.08 | |
| 7.4 | | 6.381 | | | 6.03 | | | 58.24 | | | 65.25 | | | 0.11 | | | 0.11 | | | 8.09 | |
| 9.221 | | 6.999 | | | 5.69 | | | 65.25 | | | 65.09 | | | 0.12 | | | 0.11 | | | 8.01 | |
| 8.5705 | | 7.2646 | | | 5.844 | | | 66.132 | | | 62.778 | | | 0.117 | | | 0.11 | | | 8.403 | |
| 1.123167 | | 0.857116639 | | | 0.622418223 | | | 8.17434034 | | | 8.292326841 | | | 0.004830459 | | | 0 | | | 0.819404255 | |
| Load Balance 30 requests - pre-process (w/o load balancing) (3 instances of application) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 7.636 | | 6.629 | | | 6.36 | | | 76.8 | | | 53.36 | | | 0.11 | | | 0.11 | | | 7.88 | |
| 7.627 | | 6.603 | | | 6.59 | | | 76.82 | | | 60.27 | | | 0.11 | | | 0.1 | | | 7.96 | |
| 7.777 | | 6.84 | | | 5.97 | | | 74.72 | | | 56.25 | | | 0.12 | | | 0.11 | | | 7.79 | |
| 7.875 | | 6.772 | | | 6.06 | | | 75.07 | | | 56.43 | | | 0.11 | | | 0.11 | | | 7.85 | |
| 7.365 | | 6.375 | | | 5.81 | | | 79.52 | | | 59.91 | | | 0.11 | | | 0.11 | | | 7.82 | |
| 7.943 | | 6.96 | | | 5.94 | | | 73.74 | | | 64.96 | | | 0.11 | | | 0.11 | | | 7.82 | |
| 9.6 | | 8.319 | | | 5.08 | | | 61.67 | | | 54.52 | | | 0.11 | | | 0.11 | | | 7.91 | |
| 7.686 | | 6.615 | | | 5.67 | | | 76.85 | | | 74.03 | | | 0.11 | | | 0.1 | | | 7.92 | |
| 7.405 | | 6.463 | | | 5.83 | | | 79.15 | | | 60.28 | | | 0.11 | | | 0.1 | | | 7.9 | |
| 7.552 | | 6.484 | | | 5.5 | | | 78.15 | | | 65.75 | | | 0.11 | | | 0.1 | | | 8 | |
| 7.8466 | | 6.806 | | | 5.881 | | | 75.249 | | | 60.576 | | | 0.111 | | | 0.106 | | | 7.885 | |
| 0.6430364 | | 0.560883232 | | | 0.423541943 | | | 5.123910074 | | | 6.256321603 | | | 0.003162278 | | | 0.005163978 | | | 0.066374359 | |
| Load Balance 30 requests - pre-process (w/o load balancing) (3 instances of application) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 8.615 | | 7.617 | | | 5.44 | | | 66.84 | | | 61.53 | | | 0.12 | | | 0.11 | | | 7.86 | |
| 8.133 | | 7.063 | | | 5.65 | | | 72.29 | | | 51.86 | | | 0.11 | | | 0.1 | | | 7.84 | |
| 7.328 | | 6.323 | | | 6.01 | | | 79.88 | | | 68.19 | | | 0.12 | | | 0.1 | | | 7.84 | |
| 7.973 | | 6.927 | | | 5.38 | | | 73.41 | | | 58.44 | | | 0.12 | | | 0.1 | | | 7.75 | |
| 10.73 | | 9.082 | | | 5.17 | | | 56.74 | | | 59.11 | | | 0.12 | | | 0.11 | | | 7.92 | |
| 7.741 | | 6.663 | | | 5.73 | | | 76.56 | | | 69.88 | | | 0.11 | | | 0.1 | | | 7.92 | |
| 7.513 | | 6.484 | | | 6.31 | | | 78.14 | | | 71.52 | | | 0.12 | | | 0.1 | | | 7.92 | |
| 7.919 | | 6.827 | | | 5.67 | | | 75.37 | | | 72.63 | | | 0.11 | | | 0.1 | | | 7.9 | |
| 7.454 | | 6.428 | | | 5.91 | | | 78.43 | | | 54.98 | | | 0.11 | | | 0.1 | | | 7.9 | |
| 7.898 | | 6.794 | | | 5.69 | | | 75.76 | | | 61.79 | | | 0.11 | | | 0.11 | | | 7.9 | |
| 8.1304 | | 7.0208 | | | 5.696 | | | 73.342 | | | 62.993 | | | 0.115 | | | 0.103 | | | 7.875 | |
| 0.9860368 | | 0.813639806 | | | 0.347742721 | | | 6.920041426 | | | 7.207773658 | | | 0.005270463 | | | 0.004830459 | | | 0.054006172 | |
| Load Balance 30 requests - pre-process (w/o load balancing) (3 instances of application) | | | | | | | | | | | | | | | | | | | | | |
| Total Time | | Avg Request Time | | | Avg Local CPU | | | Avg Edge CPU | | | Avg Data Centre CPU | | | Avg Local RAM | | | Avg Edge RAM | | | Avg Data Centre RAM | |
| 7.359 | | 6.378 | | | 6.91 | | | 79.67 | | | 57.56 | | | 0.12 | | | 0.1 | | | 7.88 | |
| 8.917 | | 6.69 | | | 6.01 | | | 76.76 | | | 57.68 | | | 0.12 | | | 0.1 | | | 7.89 | |
| 7.219 | | 6.224 | | | 5.95 | | | 80.83 | | | 68.58 | | | 0.11 | | | 0.1 | | | 7.93 | |
| 7.816 | | 6.482 | | | 5.57 | | | 78.15 | | | 62.63 | | | 0.12 | | | 0.1 | | | 7.93 | |
| 7.356 | | 6.304 | | | 6.6 | | | 79.7 | | | 55.13 | | | 0.12 | | | 0.1 | | | 7.92 | |
| 7.314 | | 6.331 | | | 5.4 | | | 79.66 | | | 59.09 | | | 0.11 | | | 0.1 | | | 7.91 | |
| 7.271 | | 6.278 | | | 6.38 | | | 80.61 | | | 55.22 | | | 0.12 | | | 0.1 | | | 7.94 | |
| 7.306 | | 6.354 | | | 5.72 | | | 79.38 | | | 51.74 | | | 0.11 | | | 0.1 | | | 7.89 | |
| 7.491 | | 6.537 | | | 6.8 | | | 78.68 | | | 52.62 | | | 0.11 | | | 0.11 | | | 7.99 | |
| 7.742 | | 6.449 | | | 5.8 | | | 79.29 | | | 66.63 | | | 0.11 | | | 0.1 | | | 7.98 | |
| 7.5791 | | 6.4027 | | | 6.114 | | | 79.273 | | | 58.688 | | | 0.115 | | | 0.101 | | | 7.926 | |
| 0.5107808 | | 0.139187364 | | | 0.527935181 | | | 1.185149311 | | | 5.663706678 | | | 0.005270463 | | | 0.003162278 | | | 0.036878178 | |

## Minutes of meetings