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 their 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

* 100 word outline of subject matter/findings

Table of Contents

[Acknowledgements 2](#_Toc479009671)

[Abstract 2](#_Toc479009672)

[Introduction and Problem Specification 4](#_Toc479009673)

[System Requirements Specification 4](#_Toc479009674)

[Design 5](#_Toc479009675)

[Architectural Description 5](#_Toc479009676)

[User Interface Design 6](#_Toc479009677)

[Software System Design 8](#_Toc479009678)

[Client 8](#_Toc479009679)

[Edge 9](#_Toc479009680)

[Data Centre 11](#_Toc479009681)

[Implementation and Testing 11](#_Toc479009682)

[Choice of implementation languages and development environments 12](#_Toc479009683)

[How each component was implemented 13](#_Toc479009684)

[Client 13](#_Toc479009685)

[Edge 14](#_Toc479009686)

[Data Centre 15](#_Toc479009687)

[Use of software libraries 15](#_Toc479009688)

[Key implementation details 15](#_Toc479009689)

[Discussion of test approach 16](#_Toc479009690)

[Caching System Tests 16](#_Toc479009691)

[Client Application 16](#_Toc479009692)

[Edge Node Application 17](#_Toc479009693)

[Voice Recognition System Tests 17](#_Toc479009694)

[Client Application 17](#_Toc479009695)

[Edge Node Application 19](#_Toc479009696)

[Data Centre Application 20](#_Toc479009697)

[Machine Learning System Tests 20](#_Toc479009698)

[Client Application 20](#_Toc479009699)

[Edge Node Application 21](#_Toc479009700)

[Data Centre Application 22](#_Toc479009701)

[Load Balancing System Tests 23](#_Toc479009702)

[Client Application 23](#_Toc479009703)

[System Evaluation 24](#_Toc479009704)

[Experimentation 24](#_Toc479009705)

[Caching Experimentation 24](#_Toc479009706)

[Does utilising Edge Computing reduce the latency of requests for the Client Device? 24](#_Toc479009707)

[Voice Recognition Experimentation 25](#_Toc479009708)

[Does utilising Edge Computing reduce the computational load on the Client Device? 25](#_Toc479009709)

[Does utilising Edge Computing increase the latency of the request for the Client Device? 27](#_Toc479009710)

[Machine Learning Experimentation 28](#_Toc479009711)

[Does utilising Edge Computing reduce the Network Utilisation? (not sure what to do with this one so far) 28](#_Toc479009712)

[Load Balancing Experimentation 29](#_Toc479009713)

[Do the benefits of utilising Edge Computing deteriorate when multiple requests are made concurrently? Can this be improved with a custom load balancing aspect? 29](#_Toc479009714)

[Conclusion 30](#_Toc479009715)

[References 31](#_Toc479009716)

[Copyright 31](#_Toc479009717)

[Appendices 31](#_Toc479009718)

[User manual 31](#_Toc479009719)

[Test Results 31](#_Toc479009720)

[Minutes of meetings 31](#_Toc479009721)

# Introduction and Problem Specification

There is currently a growing demand for edge computing[2] as the strain being placed on data centres is growing with the increase of Client and IoT devices. This project will investigate the effects of edge computing in a 3-tier system consisting of a client device, the edge, and the data centre when handling requests in different scenarios. 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[3] is when some data produced by or being requested by the client is pre-processed on an edge device, this affects the computational strain on both the data centre and the client and also the network traffic between the systems[4]. This has the potential to reduce bandwidth[5] which is becoming more important as the number of connected devices increases rapidly. Edge computing could deal with problems relating to latency, bandwidth [http://ieeexplore.ieee.org/abstract/document/7469991/], and computational strain.

The system being implemented at the edge could be required to last as long as double the expected life of a server at a data centre[6] 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 administrators 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.

Edge computing is not required or useful in every scenario [4] [more needed] as occasionally the stronger computational processing capabilities of the data centre or data that is not available at the edge is required.

The goal of this project is to produce a system that can handle…

# 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 are as follows;

1. A user interface 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. A low latency connection between the client and edge node
3. The requests from the client to be pre-processed quickly on the edge node before being passed on to the data centre or returned to the client
4. A data centre application capable of processing requests from the edge or from the client
5. The computational strain placed on the client should be lessened by the introduction of edge computing
6. The development and implementation of applications capable of performing edge computing tasks
7. The ability to run multiple applications on the same edge node
8. Edge node applications can be easily deployed, controlled, and updated

# Design

## Architectural Description



The entry point for the system will be the Electron Application which runs on the Client Raspberry Pi. There will be 4 pages that the user can access from the homepage. Traffic from each application is colour coded.

The Edge will be comprised of 3 Raspberry Pi’s which form a Docker swarm. The applications run in containers that can be running on any of the Raspberry Pi worker nodes and have multiple instances depending on the deployment.

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

## User Interface Design



The purpose of the graphical user interface (GUI) was to allow interaction with the Edge Nodes and Data Centre to perform tests and demonstrate functionality as per the requirement stated above.

The caching application was 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.



The voice recognition application required more interaction from the user to execute a test. This meant that the GUI contained more elements. There was a home button that was consistent throughout the entire application as a means of navigating back to the homepage. There was a button for the user to record their voice for the system to process. The two buttons below the recording button would allow the user to choose whether the voice recording was executed locally or remotely. The checkbox below the remote button was a modifier that would indicate if the request 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.



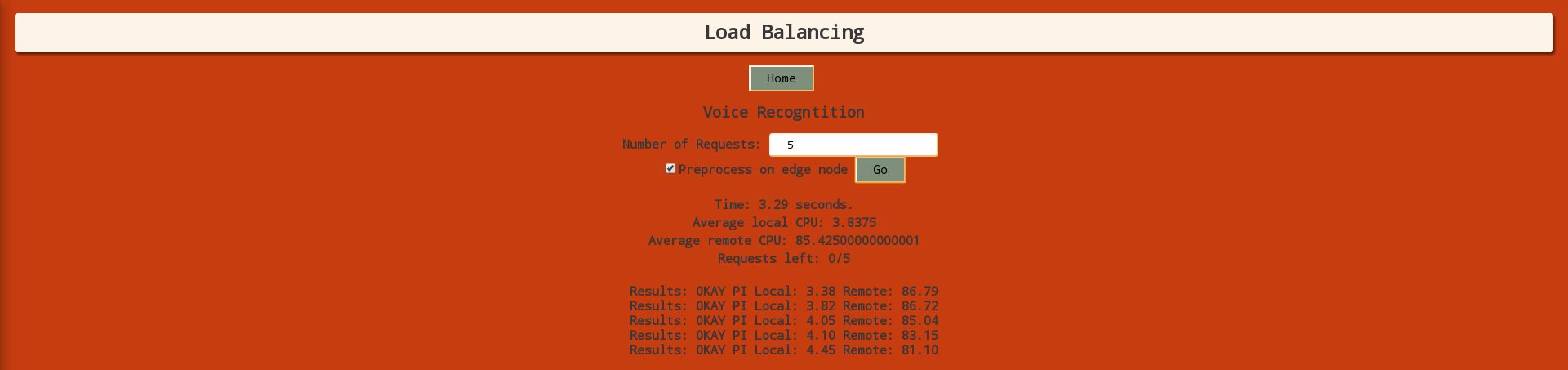
When the machine learning application is first launched, the user is asked to enter their username;



After the user enters their username the screen is populated with the average of their previous results and a recommendation. 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.



The final link from the homepage brings the user to a load balancing application. This application has the standard home button, it them had a space for a user to enter a numerical value. This value will dictate the number of lines displayed to the user in the results section. 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 then starts 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 as a whole 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 what the applications were trying to achieve.

## Software System Design

**ENSURE CONSISTANT USE OF “WILL” AND “SHOULD” RATHER THAN “DOES”**

### 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 application that contains separate pages for each of the services running on the Edge Node 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 to be implemented. A distinct HTML and JavaScript file should exist for every individual page. A “Common” JavaScript file should exist that gets reused throughout the application. The caching application requires an input area and buttons for navigating back home, executing the request, and clearing the cache. The clear cache button should send a request to the edge application for it to clear the cache. Once the request has finished executing the result should be displayed to the user.

Voice Recognition

The voice recognition page will be next and it requires more elements, such as the ability to record your voice and use this recording for executing experiments. The results section 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.

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 these details before they can test this system. After these have been entered a request should be sent to the data centre to obtain the user’s previously watched movies. This can then be used to get a recommendation request from the edge node. This should happen without any user intervention so that they 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.

### Edge

The role of the edge 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 Node is the ability to run multiple application on the same Edge Node to allow for extensibility. This will be achieved by running the applications in Docker Swarm and hosting them on unique ports. Node.js will be 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.

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). 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 container should be available across all the edge nodes and the caching application should access each of these directly and in a fair manner. This is to utilise the resources available in the edge node swarm 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 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. 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 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

The voice recognition being performed should be the same as that being performed on the edge node. The data centre will know if the data has been pre-processed or not, 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.

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 needed to be a low powered device that was low priced as to make it accessible, this made the Raspberry Pi computer a good choice.

Electron was chosen as the framework that would be used to implement the Client Application. Electron utilises Node.js 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 was a Raspberry Pi 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 computers, 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, cheap so that they are viewed as a viable option to distribute to many different areas as the Clients will be widely distributed and the Data Centre is usually in one central location, and relatively powerful as they are equipped with a quad core ARM CPU and 1GB of RAM.

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 framework that…. **(Reference)** As there are multiple edge nodes it was specifically Docker Swarm that was used as multiple worker nodes could be added to 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 (-global) that can be used.

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 utilising WebAPI and hosted in IIS as this was readily available in a Windows development environment. WebAPI was implemented in C#... (previously known)

## How each component was implemented

### Client

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.

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

**BADLY WRITTEN FIRST PARAGRAPH**

The first issue that needed to be solved for voice recognition to occur on the client was for audio to be able to be recorded. For this to happen investigation had to be done into how audio was handled on Linux systems. Once the microphone was setup on the client device audio needed to be recorded through JavaScript, for this to happen a third part library was used **(reference)** and audio was captured. This recording had to be in a specific format so that PocketSphinx**(reference)** could use it. When the user has finished recording their voice the audio file is saved to disk, this recording is then used when the user requests it to be processed, 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. 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

This went through multiple iterations. The first UI was basic and **(screenshot?)** and then it was updated to perform more complex computations on the available data.

JSON communication between the different applications which required all previous results to be obtained and averaged.

Load Balancing

The load balancing application is used to test the voice recognition system with a multitude of requests, this is achieved creating a thread per number of requests that the user has entered, 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

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 machine can host multiple applications and Docker handles the direction of traffic to where the application is hosted within its network layer.

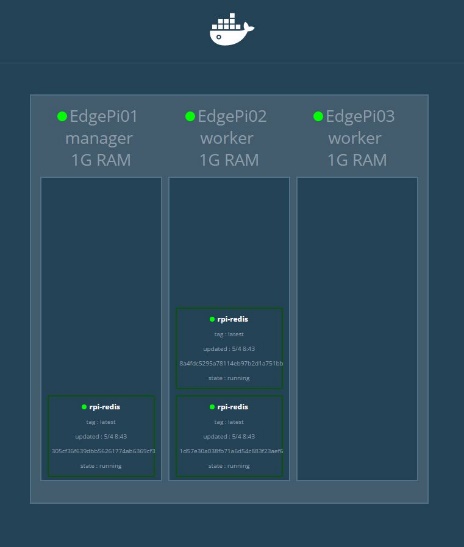
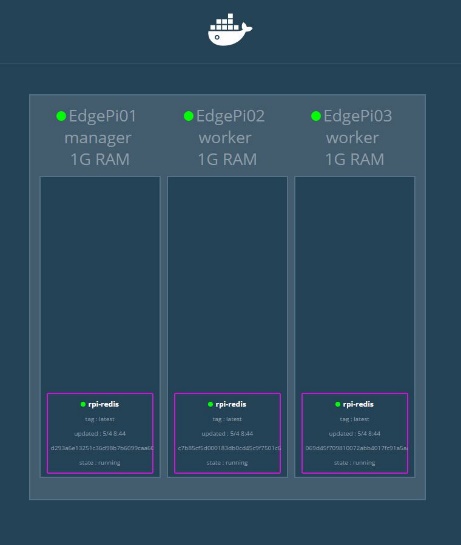
To quickly view what containers were running on what machine a visualizer**(reference)(See figure)** was used.

By default, when you want to pull an image into the Docker environment it will try to pull it from the Docker Hub**(reference)** 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**(reference)** is built to run on x86 processors and the Raspberry Pi that will be hosting the private registry would be running an ARM processor. An alternative repository image was found **(reference)** 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 master node where the image was built. If a worker node other than the master gets a request to launch the application they can search the private registry hosted on the master 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

As first iteration of the caching application passed the user requested URL as a query string the application read this query string and made a request. As this is a caching application it should not be queried directly it which meant that it needed to act as a proxy. This was solved by using the http-proxy library**(reference)** available for Node.js. What this meant from the point of view of the client is that they can enter a URL and not be aware that the caching application is present.

The caching application had some challenges. The application began by utilising a single Redis cache. There is a Redis cache container available for use with Docker so once it was fetched it the command to deploy it needed to be added to the deployment script. However, the 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 master node which would inundate the Docker Swarm master 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 **(Figure above)** 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**(reference)** and using the “—mode global” command that guarantees that there will be an instance on every node**(Figure here)**. 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 master 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**(Reference)**.

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**(reference)**. 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**(Reference)** 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. 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.

PocketSphinx required some setup… The script and voice model…

Saving the file locally on disk and making it possible to handle multiple requests concurrently was solved by appending a GUID **(Reference)** 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**(FS Reference)**. 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**(Reference)** 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

JSON for communication between all 3 systems

Need a decent sized section for refactor of the way the machine learning was going to happen

### 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 you could only build PocketSphinx from source and it was missing dependencies that it required to build in Visual Studio 2015. There was a project available from source**(Reference)** 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**(Reference)** use of the Data Centre.

Machine Learning

As the machine learning application needs to process a large quantity of movies an application was written to generate 1,000,000 lines of random test data.

## Use of software libraries **(REFERENCES)**

* PocketSphinx for voice recognition (mention that a recording was used rather than continuous listening)
* Node.js packages
  + Node.js proxy
  + NodeRedis
  + Request
* JSON.Net
* Redis

## Key implementation details

Docker… and deployment scripts. The UI that was written for ARM. How the docker file and base images work.

* Once I got the setup done.
* Deciding on redis because of docker image
* Deciding on node/webapi
* **Read notebook**
* Proxy instead of regular web server. This allows for extraction of URL’s on edge node and makes it seem more like real life

## Discussion of test approach

The nature of the client and edge node code produced meant that is proved difficult to write unit tests for as it must run in a Node.js server and make actual requests, so there was a lot of manual testing. I tried to run tests using QUnit but the test would throw an exception as soon as the first Node.js keyword, ‘require’, was used. This meant that a manual testing approach would be taken. This would be adequate for the system because it is designed to produce test results and is for research purposes. 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.

## Caching System Tests

### Client Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can reach Caching Application page | Click on the Client 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 |  |
| URL bar is present | On the caching page ensure there is a URL bar | The URL bar is present |  |
| Can type into URL bar | Click into the URL bar and type an address | The address typed will show up in the URL bar |  |
| Go button without URL typed | Click the Go button when there is no URL typed out | Nothing, there will be no error |  |
| 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 |  |
| 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 |  |
| The timer | Click the go button with a valid URL | The timer will start and when the request is finished will record the total time for the request to finish |  |
| Clear cache button | Click the clear cache button | Information will appear to indicate that the cache on the edge node was cleared successfully |  |
| Can connect to the proxy | Make a web request | Ensure that the proxy is utilised |  |

### Edge Node Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Can connect to the Redis instance | Deploy the web server | The server will automatically connect to Redis |  |
| 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 |  |
| Receive 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. This information is then returned to the user |  |
| Receive new request | Call the Edge Node with a new URL | The content is retrieved, stored in Redis and returned to the client |  |
| Receive a request for the second time | Call the Edge Node with a URL that has already been requested | The content is retrieved from Redis and returned to the user |  |

## 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 |  |
| Home button | Click home button | Should be returned to the home page |  |
| Recording status after page launch | Launch the Voice Recognition application | The recording status should be “Not Recording” |  |
| Recording status after clicking record | Click the record button | The recording status should change to “Recording” |  |
| Recording status after stopping recording | Click the stop recording button after starting a recording | The recording status should change to “Finished recording” |  |
| Recording button when not recording | Stop a recording or don’t start a recording | The recording button should read to “Start Recording” |  |
| Recording button when recording | Start a recording or don’t stop a recording | The recording button should read “Stop Recording” |  |
| Execute local recognition button without recording | Press the Execute Local Recognition button without recording a phrase | Nothing should happen, no error should be thrown |  |
| 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 |  |
| Execute remote recognition button with no recording | Press the Execute Remote Recognition button without recording a phrase | Nothing should happen, no error should be thrown |  |
| 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 |  |
| 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 |  |
| 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 |  |
| Local Results section when the local evaluation has occurred | Perform local evaluation | The local results section should show statistics |  |
| 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 |  |
| Remote Results section when no remote evaluation has occurred | Perform remote evaluation | The remote results section should show statisticss |  |

### 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 |  |
| Receive Get Request | Make a get request to the Edge Node | No error but no data returned |  |
| Receive Put Request | Make a put request to the Edge Node | No error but no data returned |  |
| Receive post request with invalid data | Make a post request with an invalid audio file or incorrect data | No error but no data is returned |  |
| 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 |  |
| 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 |  |
| 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 |  |
| 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 |  |
| Request information from the Data Centre when the WebAPI is published | Publish the WebAPI and make a request | The information should be returned correctly |  |
| Measure CPU use | Make a regular request | The CPU use on the edge node should be returned as part of the request |  |
| Measure Time of request | Make a regular request | The Time taken to process the request should be recorded |  |

### Data Centre Application

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Method | Expected Results | Passed |
| Receive invalid request to process voice | Make a valid POST request with binary voice data | The voice is processed correctly and a valid response is produced |  |
| Receive valid request to process voice | Make an invalid POST request with invalid voice data | The voice request should be handled correctly and gracefully respond |  |
| Receive invalid request to record information | Make a valid POST request with textual data from a pre-processed request | The request should be processed correctly and a response generated |  |
| Receive valid request to record information | Make an invalid POST request without valid textual data | The request should be handled gracefully and a response sent |  |

## 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 |  |
| Home button | Click home button | Should be returned to the home page |  |
| Username box is present | On the Machine Learning page ensure the username box is visible | The Username box is visible |  |
| 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 |  |
| 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 |  |
| 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 |  |
| 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 |  |
| 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 |  |
| 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 |  |
| Watch a recommended movie when there is no movie recommendation | Click the button to watch the recommended movie when there is no movie recommendation | The no movie request will be performed |  |
| 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 |  |

### 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 |  |
| Receive Get Request | Make a get request to the Edge Node | No error but no data returned |  |
| Receive Put Request | Make a put request to the Edge Node | No error but no data returned |  |
| Receive post request with invalid data | Make a post request with an invalid audio file or incorrect data | No error but no data is returned |  |
| Get movies on launch | Deploy the application | A request is performed to download movies from the Data Centre |  |
| Write received movies to disk | Deploy the application | The received movies from the Data Centre are stored on disk |  |
| Get Recommendations | Make a request to get recommendations | A movie is returned to the client as a recommendation |  |
| Watch Random Movie | Make a request to watch a random movie | The request is forwarded to the Data Centre and a recommendation produced then all the data is returned to the Client |  |
| Watch Movie | Make a request to watch a specific movie | The request is forwarded to the Data Centre and a recommendation produced then all the data is returned to the Client |  |
| Produce recommendation | Make a request that will return a recommendation | A recommendation is produced |  |

### 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 |  |
| 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 |  |
| 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 |  |
| 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 |  |
| 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. |  |

## 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 |  |
| Home button | Click home button | Should be returned to the home page |  |
| 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 |  |
| 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 |  |
| The go button is visible | Check the button for executing the requests is visible | The button for executing the requests is visible |  |
| 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” | Nothing should happen |  |
| Press go when text is entered | Type a value that is not numerical | Nothing should happen |  |
| Press go when numbers are entered | Type a value that is numerical, non negative and less than 100 | The number of requests entered should start executing and display in the results section |  |
| Press go when a number greater than 100 is entered | Type a value that is greater than 100 | An error message should appear warning you to enter a valid number |  |

# System Evaluation

Not sure what’s included here

# Experimentation

A lot of the emphasis in this project will be here.

Should setup experiment to not require human interaction.

Provide empirical results and draw conclusions.

Describe methodology (use experiment methodology like goals and hypothesis, what’s measured/controlled)

When is it good to use edge node. When is it good to use DC? Not always good to use both

Read more in notes

Setup experiment automation

## Caching Experimentation

### Does utilising Edge Computing reduce the latency of requests for the Client Device?

Setup;

* Run the Caching Application on the Client Device
* Deploy the Caching Service to one of the Edge Devices
* Deploy a Redis Instance to each of the Edge Devices

Isolate Variables;

Constants:

* Interface connecting Raspberry Pi to the router
* The web page being loaded
* The other applications deployed to the Edge Nodes

Variables

* Whether a webpage is cached or not

Method;

Measurements recorded will be the time between when the WebView element in the Client Application starts a request and ends a request, as there are many requests executed within the loading of one webpage such as images…..

After the Caching Service and Redis Service have been deployed, 10 warmup requests will performed without wiping the cache. The experiment will be performed with 10 iterations and an average will be taken.

An iteration will consist of an initial request to “http://www.bbc.co.uk” and then a subsequent request when the information is already cached on the edge node. The two times will then be recorded in seconds and the cache will be cleared ready for the next request

Hypothesis;

The hypothesis is that the cached request should take less time to execute than the initial request.

Results;

<Graph>

Analyse;

<Analysis of graph>

Conclusion;

## Voice Recognition Experimentation

### Does utilising Edge Computing reduce the computational load on the Client Device?

Setup;

* 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 Device
* 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 applications deployed to the Edge Nodes

Variables

* Where the voice processing is occurring

The variable in this experiment is whether I process the file locally on the Client Device, remotely pre-process it on the Edge Device or remotely process it on the Data Centre.

Method;

Measurements recorded will be:

* CPU load in a percentage of the total CPU on the Client Device
* CPU load in a percentage of the total CPU on the Edge Node
* Processing time in seconds
* File size
* Length of voice recording

After the Voice Recognition Service 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.

The measure of time will be recorded on the Client Device from as soon as the request starts to when the request finishes. The experiment will be repeated 10 times and an average will be taken.

The experiment will consist of a voice being recorded. The same recording will be used throughout the experiment to allow for a fair comparison of computational load.

There will be a round of 10 experiments run on the Client Device, then a round of 10 where the device is pre-processed on the Edge Node, and a final 10 where the request is processed on the Edge Device.

Hypothesis;

The hypothesis is that the Data Centre should process the request the fastest as it has the most powerful CPU but the CPU strain on the Client Device should be reduced regardless of the request being pre-processed on the Edge Device or the Data Centre.

Results;

<Graph>

Analyse;

<Analyse graph>

Conclusion;

### Does utilising Edge Computing increase the latency of the request for the Client Device?

Setup;

* 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 Device
* 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 applications deployed to the Edge Nodes

Variables

* Where the voice processing is occurring

The variable in this experiment is whether I process the file locally on the Client Device, remotely pre-process it on the Edge Device or remotely process it on the Data Centre.

Method;

Measurements recorded will be:

* CPU load in a percentage of the total CPU on the Client Device
* CPU load in a percentage of the total CPU on the Edge Node
* Processing time in seconds
* File size
* Length of voice recording

After the Voice Recognition Service 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.

The measure of time will be recorded on the Client Device from as soon as the request starts to when the request finishes. The experiment will be repeated 10 times and an average will be taken.

The experiment will consist of a voice being recorded. The same recording will be used throughout the experiment to allow for a fair comparison of computational load.

There will be a round of 10 experiments run on the Client Device, then a round of 10 where the device is pre-processed on the Edge Node, and a final 10 where the request is processed on the Edge Device.

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’s however the request should be processed much quicker on the Data Centre as it has a faster processor although this means that the Data Centre is performing more work.

Results;

<Graphs>

Analyse;

<Analysis of graphs>

Conclusion;

## Machine Learning Experimentation

### Does utilising Edge Computing reduce the Network Utilisation? (not sure what to do with this one so far)

Setup;

Isolate Variables;

Constants:

Variables

Method;

Hypothesis;

Results;

<Graph>

Analyse;

<Analysis of graph>

Conclusion;

## Load Balancing Experimentation

### Do the benefits of utilising Edge Computing deteriorate when multiple requests are made concurrently? Can this be improved with a custom load balancing aspect?

Setup;

* 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 CPU use from other applications

Variables;

* Number of requests
* Whether the deployed Edge Node service is utilising custom Load Balancing

Method;

Measurements recorded will be:

* Average CPU load in a percentage of the total CPU on the Client Device
* Average CPU load in a percentage of the total CPU on the Edge Node
* Processing time in seconds for all requests to finish

The experiment will consist of a voice being recorded. The same recording will be used throughout the experiment to allow for a fair comparison of computational load.

Deploy the Voice Recognition Service to the Edge Node without custom load balancing enabled and perform 2 sets of 10 warm up requests.

Perform the experiment 10 times with varying number of requests, from 10 to 50 in increments of 10. Repeat all 5 experiments after deploying the Service to the Edge Node with load balancing enabled and performing the warmup again.

Hypothesis;

Adding load balancing will improve response times (over 70% CPU utilisation the request gets sent to the Data Centre)

Results;

Analyse;

Conclusion

# Conclusion

General summary of the success of the project with respect to criteria identified in the intro

Discussion of significance of experimental results. Agree with others work?

Strengths and weaknesses

Useful in some cases and can be implemented in different ways (DC ease vs Client ease) but not useful in all?

If I was to continue this project…

Evaluation of hardware/software

Looking for critical appraisal and significance of contribution in the context of wider work.

# References

[1]"Final Year Project Repository", *GitLab*, 2017. [Online]. Available: https://gitlab.eeecs.qub.ac.uk/40103631/FinalYearProject. [Accessed: 21- Mar- 2017]

# Copyright

* Electron
* Node.js/ Node.js libraries
  + Node JS proxy
* Pocketsphinx
* Testing Suites
* Redis/Docker src images
* JSON
* Check licence file in Client src folder

# Appendices

## User manual

## Test Results

Excel results?

## Minutes of meetings