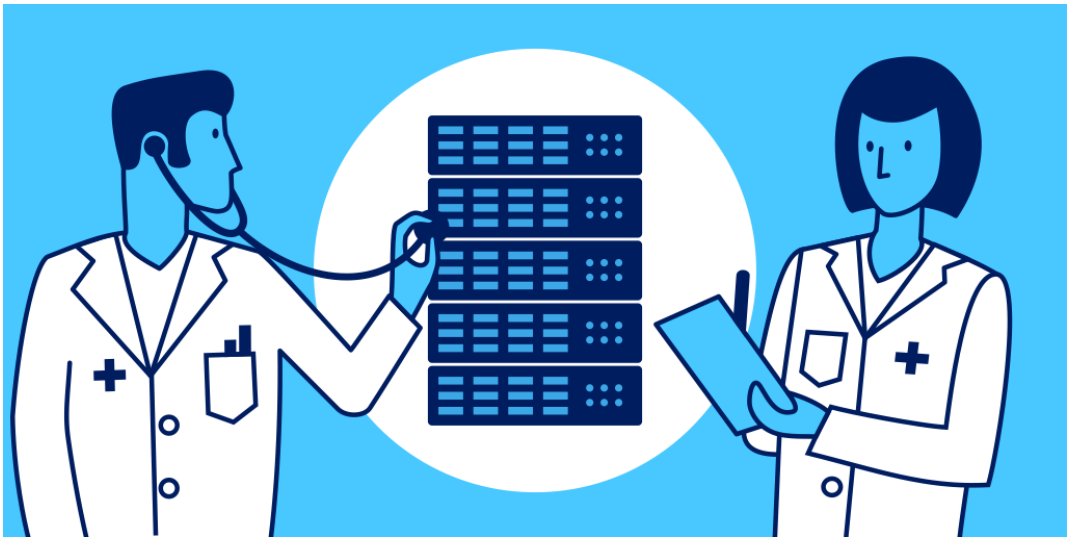
Server Health Monitoring

Made by :
Arushi Aggarwal (18103098)
Nazia Ali (181013102)
Mathew Pius (18103105)
Nikhil Shaji (18103109)
Submission of Report: 18th December 2020

Under the supervision of:
Dr. Divya Bansal.

Server Health Monitor



Department of Computer Science Engineering
Punjab Engineering College(Deemed to be University)

TABLE OF CONTENTS

LIST OF FIGURES…….ii
ACKNOWLEDGMENT…….……iii
ABSTRACT ……....iv
1.0 INTRODUCTION……..1
1.1 PROBLEM STATEMENT……...1
1.2 PROPOSED SOLUTION..…….....1
1.3 BENEFITS OF MONITORING YOUR REMOTE SERVERS……..3
1.4 FEATURES OF SERVER HEALTH MONITOR……3
2.0 SERVER HEALTH MONITOR ARCHITECTURE…….....4
3.0 TECHNOLOGY USED……..5
3.1 NODE.JS……..5
3.2 REACT.JS……6
3.3 D3.JS……7
3.4 MONGODB……8
3.6 AUTOMATED SSH……...……....9
3.6 HEALTH MONITORING USING WEBSOCKETS……....9
3.7 FLASK MICROSERVICE……...10
4.0 PREDICTIVE ANALYSIS USING TIME SERIES……...……...12
CONCLUSION……16
BIBLIOGRAPHY…….17

LIST OF FIGURES AND TABLES

FIGURES

Figure 1 UML Sequence Diagram…….…….2
Figure 2 Server Health Monitor Architecture Diagram…….4
Figure 3 Flask Architecture Diagram……..11

TABLES

Table 1 : Model Summary……..13

ACKNOWLEDGMENT

We have taken a lot of deliberations in this venture. But it wouldn't have been possible without the help and backing of numerous people. We would like to thank everyone who has helped us complete this project.

We would like to take this opportunity to thank and extend our profound gratitude towards Dr. Divya Bansal for her guidance and support.

ABSTRACT

Server health is a very important component when it comes to running and executing servers. However, the current methods used to monitor the health of multiple servers are fairly inefficient and time-consuming in terms of user accessibility.

Server Health Monitor aims to solve the problem of vendor dependent server monitoring where users have to log in to the respective vendor website to monitor/analyze their remote server's health.

Our goal is to create a user-friendly website that allows users to monitor important information about their remote servers and hence prevent any catastrophic failures.

INTRODUCTION

1.1 Problem Statement :

Most companies and organizations rely on servers to conduct their day to day business. As such, a server is a critical piece of infrastructure that requires constant care and maintenance. Servers are often distributed across several physical locations, operating in hybrid setups making it difficult for system administrators to maintain an overview of server performance.

Predominantly, the tools most widely used for monitoring remote servers have been limited to vendor dependent software. This invokes dependence risk, wherein if there are any problems in the environment used, the software terminates and stops working.

Being able to remotely monitor and manage network performance is a necessity for modern companies. Many network monitoring solutions work on agents that may not be locally installed so the system administrator has to be present on-site to fix the problem. Such problems are fixed by the added capability of allowing remote reconfiguration which can be accessed from anywhere.

The use of software in existing technologies makes the system less portable. To be able to monitor and check logs, one has to be near a laptop with the software installed in the right configuration and environment needed.

1.2 Proposed Solution :

We measure critical metrics of the user’s remote server that helps them troubleshoot and take necessary actions needed to prevent failures.

We monitor the following metrics:

CPU usage: Allows users to monitor their CPU to determine how much load is being placed on your server's processor. You can load balance apps or upgrade/replace the hardware to boost performance.

Memory Usage: Monitor how much memory is free at an instance in time. You can delete or update files or applications based on this to boost performance.

Uptime: Monitor how long the server has been up since the first deployment. Allows users to take necessary precautions on hardware.

Bytes Read and Written by Disk: Number of bytes written and read by disk at a particular instance of time throughout the computer. Allows users to monitor the disk write and read speed as memory grows. This gives further insight into the users as to whether they have to replace or upgrade the disk.

Bytes Sent and Received(Net I/O Counters): Number of bytes sent and received by the remote servers from the internet for various tasks and applications throughout the computer. This allows users to restrict or allow communication of the remote server with the internet by updating the respective security configurations.

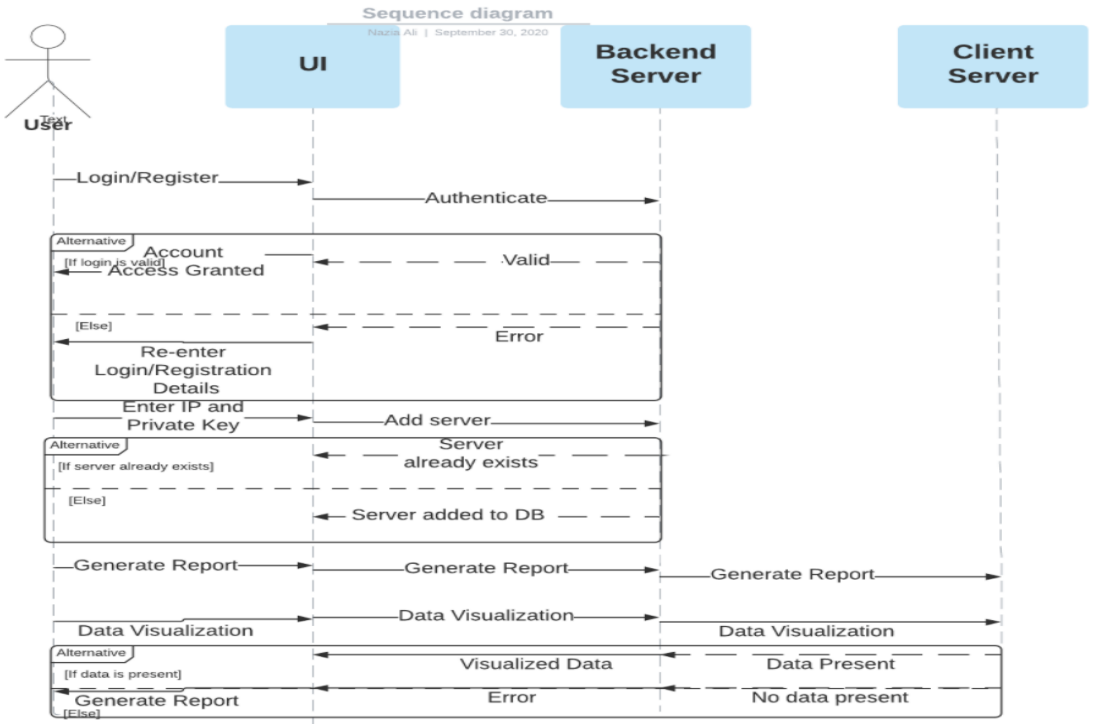


Figure 1 : UML Sequence Diagram detailing how operations are carried out

1.3 Benefits of monitoring your remote servers :

The main benefit of monitoring your remote server environment is being proactively notified of performance issues before end-users notice there is a problem. With remote server monitoring software, you can:

- Identify and troubleshoot issues related to server hardware health
- Monitor the overall performance and availability of your remote servers
- Identify other performance issues related to response time, resource utilization, app downtime, etc.
- Remotely remediate performance issues, including rebooting servers, restarting websites, and more.

1.4 Features of the Server Health Monitor :

The Server Health Monitor aims to provide the user with useful information in an easily readable and understandable form.

A few features the health monitor has are:

- Real-time data monitoring using WebSockets.
- Automated SSH into the user's remote server.
- Predictive analysis using Time Series prediction that predicts data after a specific time frame.
- Graphical representation of data for easy interpretation of data.

These features are all independent for each remote server.

Server Health Monitor Architecture:

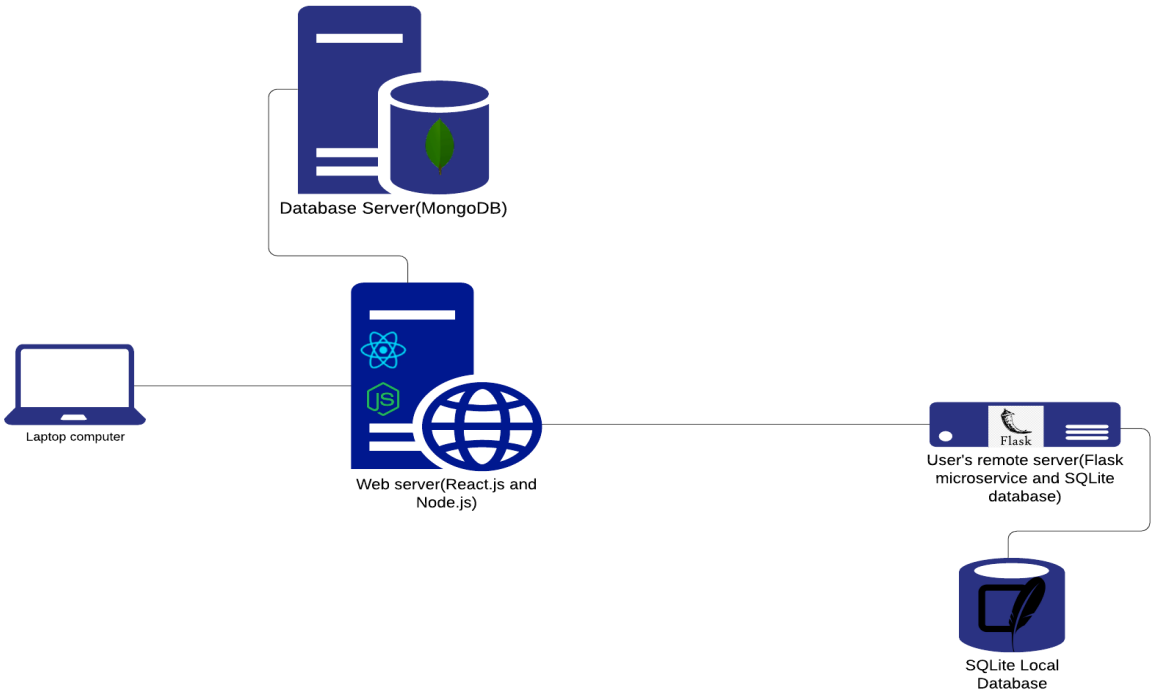


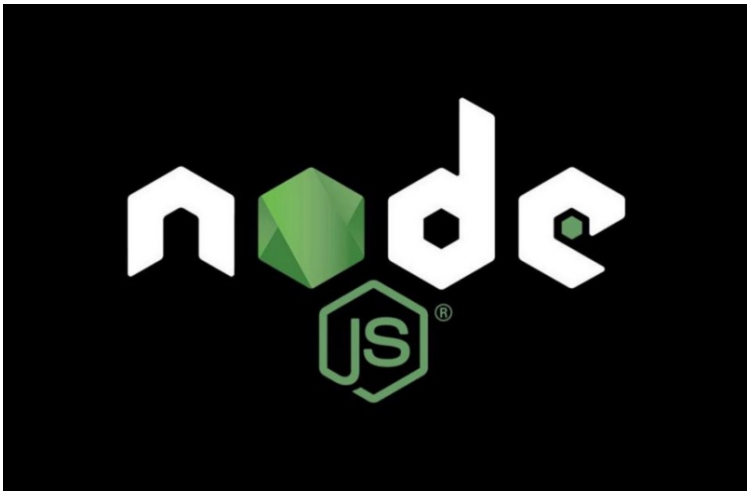
Figure 2 : Server Health Monitor Architecture Diagram

The Server Health Monitor is meant to run on the webserver with it serving pages to the user acting as the user interface.

When the user goes to the server health monitoring website, they are served with HTML, CSS, and Javascript files on their web browser.

After the user registers their server on the website, the web server automatically ssh's into the user's remote server and downloads the Flask microservice.
This microservice sends and receives requests to and from the web server. The data received by the webserver is then displayed to the user in a well-organized way.

Technology Used:

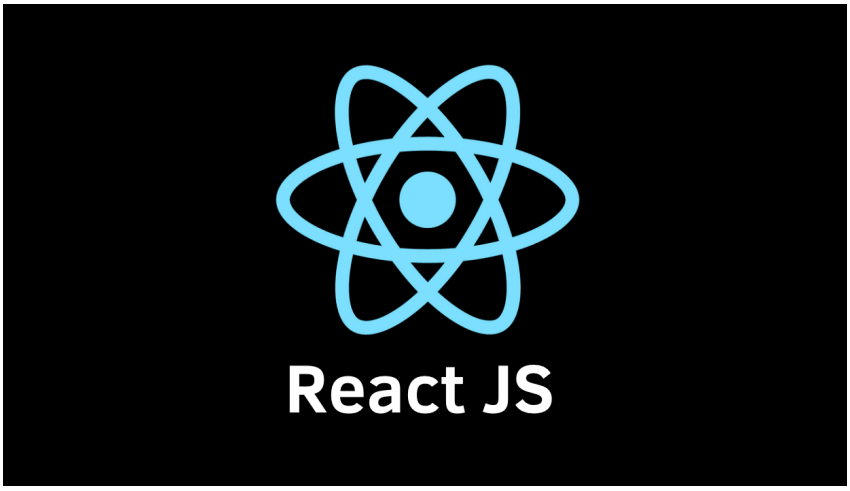


3.1 Node.js:

Node.js is an open-source, cross-platform, back-end, JavaScript runtime environment that executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command-line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server-side and client-side scripts.
[https://en.wikipedia.org/wiki/Node.js]
As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.

There were two reasons why we use chose Node.js for writing the application's backend logic:

- Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time web applications like the Server Health Monitor.
- It gives us as developers the convenience of only having to work with one language throughout the application without having to switch between different coding patterns.



3.2 React.js:

React (also known as React.js or ReactJS) is an open-source, front end, JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with rendering data to the DOM, and so creating React applications usually requires the use of additional libraries for state management and routing. Redux and React Router are respective examples of such libraries. [https://en.wikipedia.org/wiki/React_(web_framework)]
A few reasons we have used React.js for the development of the project are:

- React allows developers to create large web applications that can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple.
- React uses a virtual DOM which is a copy of the actual DOM. Therefore any changes made are first made on the virtual DOM and later the changes are incorporated on the actual DOM instead of overwriting the entire DOM each time a change is made, hence increasing the performance of the application significantly.
- Another reason is instead of designing various HTML pages we can design React components that are reusable and easy to move around. Therefore making the application easy to develop and maintain.

3.3 D3.js:

Data-Driven Documents is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3.js is written in JavaScript and uses a functional style which means that one can reuse the code. This means that we can make it as powerful as required. Choosing the style, manipulation, and making the data interactive is up to the developer.
[https://d3js.org/]
A few reasons as to why we have used this particular library in our project:

- Since D3 is a JavaScript library, it can be used with any framework like Angular.JS, React.JS, or even Ember.JS. In our project, we have implemented D3 along with React.JS
- D3 is an open-source that mainly focuses on the data. So we can work with the source code and add our features, thereby making it the most appropriate tool for data visualization. It works well with large datasets and gives us complete control over visualization to customize it according to our needs. In this project, we have used D3 to show the Server Health data in form of line charts.
- D3 works with HTML, CSS, and SVG, thus there is no additional learning or debugging tool required to work on D3. We have used HTML5, CSS3, and JS entirely on the Front-End of our project. Therefore, using D3 was another major advantage.D3 is lightweight, works directly with web standards, and is extremely fast.

3.4 MongoDB(Database):

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. Every record in a database is a JSON (Javascript object notation) object. This is ideal since there is no interaction between users on Server Health Monitor and hence the need for the ability to perform joins is not needed and the JSON format of records makes updating of schema and transformation of data obtained from records very intuitive from the perspective of a Javascript developer. [https://en.wikipedia.org/wiki/MongoDB]

We have used MongoDB in our project because:

- NoSQL structure of the database allows developers to manipulate and access data from the database easily as they are in a JSON format.
- As MongoDB does not have a specific schema it is easy to update or modify data as the application is further developed.

As you can infer from above, the Server Health Monitor uses the MERN(MongoDB, Express, React, and Node) Stack.

3.5 Automated SSH

The Health Monitor uses a library in JavaScript for automated SSH that allows the website to gain access to the user's remote server.

SSH stands for secure shell, which is a cryptographic network protocol that is used for sending or receiving data over an unrestricted network.

Typical applications include remote command-line, login, and remote command execution, but any network service can be secured with SSH.
After SSH is done into the user's remote server, the website uses the concept of File Transfer Protocol(FTP) to download and deploy files on the remote server.

FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.

3.6 Health Monitoring using Websockets:

WebSockets is a next-generation bidirectional communication technology for web applications that operates over a single socket and is exposed via a JavaScript interface in HTML5 compliant browsers.
[https://www.tutorialspoint.com/html5/html5_websocket.htm]

In the project, we have developed a web socket that communicates with the frontend of the server health monitor and subsequently fetches data from the remote server every second. This allows the user to view health data in real-time.

3.7 Flask MicroService:

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies, and several common framework related tools. [https://en.wikipedia.org/wiki/Flask_(web_framework)]
The Flask application acts as a microservice on the user's remote server where it receives and sends requests to and from the main web server.

We have used a microservice architecture as it communicates with three separate entities on the remote server.

- The local database uses SQLite for storing data.
- The report file is written in python using psutil library.
- The web server which requests the flask application for data.

The report file is a python program that collects data from the server(bytes read and written by disk, bytes sent and received from the net) using the psutil library(https://psutil.readthedocs.io/en/latest/). The data that is collected is sent to the flask application as a post request every five seconds.

The data that is received from the report file by the Flask application is stored in a local database file made in SQLite for later query and use

Flask Architecture:

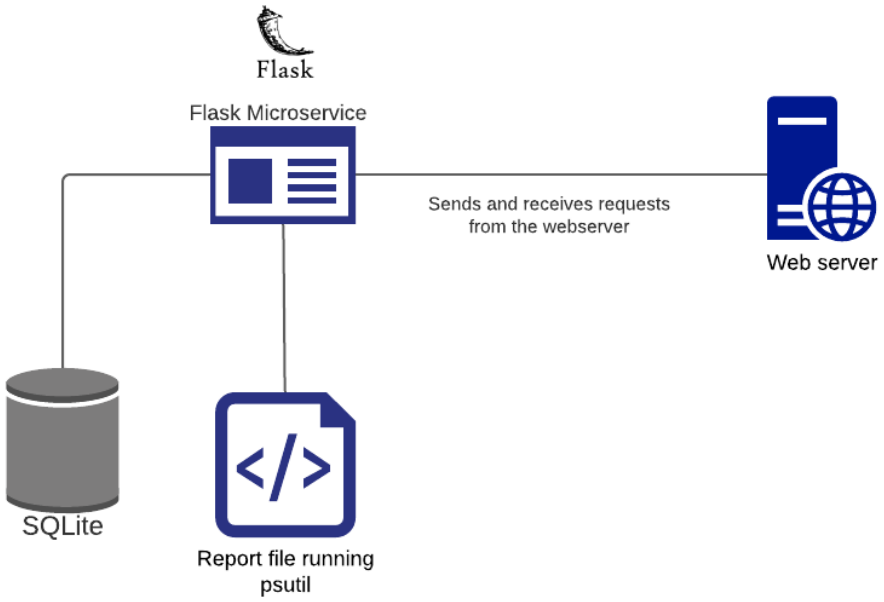


Figure 3 : Flask Architecture Diagram

The Flask application also uses predictive analysis of data collected and stored in the database. The predictive analysis is done using time series prediction.

Predictive Analysis using Time Series

Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. Time series forecasting is the use of a model to predict future values based on previously observed values.

Time series data can consist of series of observations and a model is required to learn from the series of past observations to predict the next value in the sequence.

For the use of our project, the specific type of Time Series Forecasting done was Multivariate Time Series using Long Short-Term Memory(LSTM).

Multivariate time series data means data where there is more than one observation for each time step which has to be remembered while making the prediction for the next timestep.

Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) capable of learning order dependence in sequence prediction problems. Popular Deep Learning Algorithms such as Simple Recurrent Neural Network suffer from short term memory where if a sequence is long enough, the information from earlier timesteps is not relayed to later ones. During back propagation, RNN suffers from the vanishing gradient problem where the gradient may shrink too much as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute to learning. This loss of information can drastically affect the predicted values in time series data,

LSTM Functionality:

LSTMS were created as the solution to short term memory by use of gates which can regulate the flow of information. These gates can learn which data in a sequence is important to keep or thrown away. By doing so, it can pass relevant information down the chain of sequences to make predictions. The core concept of LSTMS is the cell state and its various gates. The forget gate decides what information should be thrown away using the values from the previous hidden state and information from current input is passed through a sigmoid function wherein, a value close to 0 means it must be forgotten. To update the cell state, we have the input gate which transforms values in between 0 and 1 on a metric of relevant information. The output gate decides what the next hidden state should be which is used for predictions.

The Server Health Monitoring System makes use of LSTM as follows. The values to be used within prediction are taken from the table in the existing database. The quantity of the data needed for an LSTM depends on the complexity of the network. For our model, we made use of a Sequential model which is a linear stack of layers. The first layer within the Sequential model was the LSTM layer that received the input parameters.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 64)                16896
_________________________________________________________________
dense (Dense)                (None, 4)                 260
=================================================================
Total params: 17,156
Trainable params: 17,156
Non-trainable params: 0
_________________________________________________________________
```

Table 1 : Model Summary

The LSTM layer has 64 units, i.e, the number of hidden states within a cell. Typically, this can be any number since the hidden state is only fed back to the RNN cell at every timestep. The length of the hidden state affects the capability of the RNN cell capturing the structural and semantic features of the input cell. The shape of the output LSTM layer is (None, 64) indicating None as the batch dimension for the number of samples, 64 as the length of state vector and 1 as the time step. We do not explicitly assign the number of timesteps in the definition of the LSTM layer but it knows how many times it should reiterate over the same state once the input is applied since it contains timestep in its shape. The default activation function for a LSTM cell is tanh, however, we used Rectified Linear Activation Function (or ReLU for short) to allow the neural network to learn nonlinear dependencies. ReLU will return input directly if the value is greater than 0 but if less than 0, a 0.0 is simply learned. The idea is to allow the network to approximate a linear function when necessary, with the flexibility to also account for nonlinearity. ReLU can flatten out volatility in time series data, hence, removing the degree of variation of series over time - a component necessary to maintain the stationarity of data. A problem with the ReLU activation function is that values must strictly be >= 0, due to which gradients may become extremely large and explode leading to NaN values. Hence, to combat this, we make use of Adam Optimizer to clip gradient values that exceed a preferred range.

Additionally, we apply Dropout to the input connection within the LSTM nodes. A dropout on the input means that for a given probability, the data on the input connection to each LSTM block will be excluded from node activation and weight updates. The dropout value is a percentage between 0(no dropout) and 1(no connection).

For example, setting a Dropout of 0.4 means that there will be a dropout rate of 0.6(set 60% of inputs to zero). The outputs of the first layer have dropout applied to them which are then taken as input to the net layer.

The use of Dropout is mainly to stimulate a large number of different network architectures by randomly dropping nodes during training. This is an effective regularization method to reduce overfitting and improve generalization error.

Since our model trains on a relatively small dataset at the initial stages of adding a server to the system, the model may overfit the training data wherein, it learns the statistical noise in the training data leading to poor performance when evaluated on new values. One approach to reduce overfitting is to fit all possible different neural networks on the same dataset and to average the predictions from each model. In practice, this is not feasible and can instead be approximated with a small collection of different models that require less computational power (an important metric to consider, provided that servers added by a user may be small and cannot withstand excessive computations)

The last layer is a Dense layer which takes the 64 tensors as units and outputs 4 units which are the units that are to be forecasted. The predicted values are appended to a dictionary along with the values from the database which are sent from the flask microservice to the server in form of a JSON object. At the frontend, D3.JS is used to visualize the values in form of a line chart, showcasing the predicted trend in values given current performance.

Conclusion

Server Health Monitor aims to solve the problem of vendor dependent server monitoring where users have to log in to the respective vendor website to monitor/analyze their remote server's health.

Our goal is to create a user-friendly website that allows users to monitor important information about their linux remote servers and hence prevent any catastrophic failures.

We have created a website that allows the user to register their remote server. After registration the website can automatically SSH and set up the health monitoring functionalities. The website then presents data in a tabular and graphical representation to the user.

All the features mentioned have been implemented and now scalability issues will be looked over for a production ready website.

Bibliography and References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript
- https://devdocs.io/css/
- https://nodejs.org/api/
- https://reactjs.org/docs/getting-started.html
- https://github.com/d3/d3/blob/master/API.md
- https://flask.palletsprojects.com/en/1.1.x/

Libraries

- WebSockets (https://www.npmjs.com/package/socket.io)
- SSH2 (https://developer.aliyun.com/mirror/npm/package/ssh2#server-events)
- Psutil (https://psutil.readthedocs.io/en/latest/)
- OS-Utils (https://www.npmjs.com/package/os-utils)
- Mongoose MongoDB (https://mongoosejs.com/docs/)
- SQLite (https://docs.python.org/3/library/sqlite3.html)
- TensorFlow(https://www.tensorflow.org/guide)
- Keras (https://faroit.com/keras-docs/1.2.0/)
- Pandas (https://pandas.pydata.org/docs/)

Project Links

- https://github.com/mathewpius19/Server-Health-Monitor-v2
- Flask Microservice and Websockets (https://github.com/mathewpius19/Health-Monitoring)