

Plug-and-Play Load Balancer

CS3543: Computer Networks-II

Ganesh Vernekar
CS15BTECH11018

Sukrut Rao
CS15BTECH11036

1 OBJECTIVE

In this project, we created a plug-and-play load balancer. It consists of a master node, that receives requests to perform a particular task, and then allocates the task to one out of a set of slave nodes. Load balancing is implemented using a few simple load balancing algorithms. The entire system is plug-and-play, such that a slave node can join the pool of slaves any time and leave any time without knowing any configuration about the master. We also have a monitoring which can plug itself into the network just like slave and monitor the metrics of the slave.

The source code can be found at:

<https://github.com/GoodDeeds/load-balancer>

2 DESCRIPTION

2.1 Overview

The goal of this project is to build a plug-and-play load balancer. A load balancer is used to efficiently distribute network requests among a set of servers. This helps in improving performance and efficient management of server resources to process requests.

The load being handled by a server could be measured in many ways. A few simple metrics include the number of requests being processed, the amount of CPU usage, or the memory usage. This project performs load balancing using a metric defined using a combination of these parameters. These are detailed in Section 2.2. Our load balancer software consists of three modules:

(1) Master

This receives HTTP requests to perform a certain task. It then runs a load balancing algorithm to decide which slave node to assign the task to. It then receives the result and handles it. There is a single master node in the network. Details can be found in Section 2.3.

(2) Slave

This receives task requests from the master node. It executes the task if the load it is handling is within its load limit, and then returns the result and the status back to the master node. There are multiple slave nodes reporting to a single master node. Details can be found in Section 2.4.

(3) Monitor

This is responsible for monitoring the load on the slaves. This has been used only to evaluate the effectiveness of the load balancer and to visualize the load balancing. Details can be found in Section 2.6.

There are a wide array of load balancing algorithms in use today. In this project, we implement a few such load balancing algorithms. A description of these algorithms is given in Section 2.7.

The project handles requests restricted to a set of predefined tasks. These involve tasks to compute certain functions, such as finding the n^{th} Fibonacci number. The predefined tasks are meant to

be proof-of-concept, and it would be easy to extend the functionality to more complicated tasks. In fact, the code is structured in such a way that it would be very easy to define new tasks and plug it into the rest of the software. The list of tasks and their descriptions can be found in Section 2.8.

2.2 Load Metric

The following are used to define the load metric:

- (1) **Load of a task:** The load of tasks currently being handled by a slave. Different tasks are given different weights based on the time complexity of their algorithm. The task to find the n^{th} Fibonacci number is given a load n , and that to count upto the n^{th} prime is given a load n^2 .
- (2) **CPU Usage:** The average CPU usage per core of the slave over the past one minute is used. This is found using the shell command `uptime`.

Using the above, the load metric is defined as:

$$\text{Load} = \text{Load of task} \times \text{Fraction of average CPU Usage of slave} \quad (1)$$

2.3 Master

These are the main functionalities of the master:

(1) Create a HTTP server and listen for requests

An HTTP server is started, that listens for GET requests for performing a certain task. The request contains the type of task, which is chosen from the set of predefined tasks. It also contains the input for the task, for example, the value of n for the task of finding the n^{th} Fibonacci number. On receiving a valid request, the task is created and assigned. The result of the task is sent as the response.

As a result, requests for tasks can be sent by requesting a particular URL from the browser, and the result after completion is displayed in the web page in the response.

(2) Maintain a Slave Pool

The master node maintains a pool of available slave nodes for performing tasks. As slaves may arrive and leave any time, extra care is taken to ensure validity of operations on the slave pool, such as ensuring that the slave is still present to which task is being assigned. The master node stores certain information about each slave, that includes the slave's IP address, an ID (as multiple slaves can run on same machine), the current load metric of the slave, the maximum load the slave can handle, and all the tasks that have been assigned to it.

(3) Connect to slaves

The master node listens for broadcast requests from slave nodes and adds them to the slave pool. This allows the system to be plug-and-play, as the slaves join the slave pool

automatically without external intervention. The mechanism is elaborated in Section 2.5. The master maintains three tcp connections with each slave - one for constantly updating current load of a slave, and other two for sending tasks and receiving back the results. Two connections are maintained for tasks to reduce contention on a single socket for sending and receiving.

(4) **Manage load information**

The master node periodically requests every slave in the slave pool for its current load, and updates the corresponding data fields in the master for that slave. The load is used by the load balancing algorithms when deciding which slave a task is to be assigned to.

(5) **Handle tasks**

When a task request is received, the master node parses the request and creates a packet that stores the type of task, the input parameters, and the estimated load of the task. It also maintains a list of all tasks. It assigns each task to a particular slave based on load-balancing algorithm, and gives the result back to the http handler.

(6) **Run algorithms**

The load balancing algorithms are defined in the master node. When a new task is created, master runs the selected algorithm on the slave pool to determine which slave the task is to be assigned to, and then assigns the task.

(7) **Interface with the monitor**

Master provides the monitoring node with the IP of all the slave nodes connected to it upon request from the monitoring node.

2.4 Slave

These are the main functionalities of the slave:

(1) **Starting Prometheus service**

Before starting the slave, we start Prometheus service. We need to do it beforehand so that the slave can tell Prometheus to scrape its metrics when slave starts.

(2) **Create sockets**

Even before connecting to master, slave creates 3 tcp sockets for 3 connections described in point 3 of section 2.3. This is done beforehand to share the ports with the master during connection.

(3) **Connect to master**

On startup, slave broadcasts messages to discover the master. More about the mechanism is described in 2.5.

(4) **Listen for requests from master**

After connecting to the master, slave waits for master to connect to its sockets on the ports specified. Once connected, slave listens on these sockets for incoming requests from master.

(5) **Providing load information**

Slave receives requests from master asking for its load status in small equal intervals. Load being a real time metric, slaves responds immediately with its load status.

(6) **Handling tasks**

Upon request from master to perform a task, an ACK is sent to the master to confirm that slave is ready to take up

the task. Depending on the task type mentioned in the task packet, slave extracts appropriate input from the packet and performs the specified task. After completion, slave sends back the result to the master to the same socket to which it sent the ACK.

(7) **Expose metrics for monitoring**

We expose the following metrics for Prometheus to scrape:

- (a) **tasks_requested**: Number of tasks that the master requested to slave.
- (b) **tasks_completed**: Number of tasks slave completed processing.
- (c) **current_load**: Current load metric of the slave.

2.5 Plug and Play Mechanism

Here we explain about the mechanism between master and slave, which is very similar to what we do even for monitoring node.

(1) **Setting and Required conditions**

- (a) We use UDP for handshake as we cant make TCP connection without knowing the master IP.
- (b) The slave and the master should be in the same subnet for the broadcast messages to work. The broadcast IP is of the subnet.
- (c) We know the port number on which the master will be listening for the broadcast. Hence based on this, we do the entire handshake.

(2) **Handshake**

These steps are in the order in which they occur.

(a) **Slave Broadcasting**

The slave broadcasts the broadcast packet by sending a packet to the obtained broadcast IP from the subnet and the predefined port. We do multiple retries till we get any response from master. The port from which the slave is currently sending the packet is taken as the ID of the slave. (Slave IP, Slave ID) is used to uniquely identify a slave.

(b) **ACK/NAK from master**

Once the master receives the broadcast packet for connection, it checks against its existing slaves to avoid multiple connection requests from same slave. We also have a limit on number of slaves a master can have. If all checks are positive, then master sends back an ACK to the IP and the port mentioned in the packet. NACK otherwise.

(c) **Best effort ACK from slave**

Once slave receives ACK, it sends 10 ACKs in burst as a best effort. These ACKs also contain the IP of slave and 3 ports on which master can make TCP connections.

(d) **Final connections**

After master receives the final ACK from the slave, it opens 3 TCP connections (as described in point 3 of section 2.3) on the specified ports in the packet.

2.6 Monitoring

(1) **Starting Grafana**

Grafana is started before starting monitoring node. Grafana is used to visualize all the metrics exposed by the slaves.

(2) **Connect to master**

Connection to master is performed similar to slave. It follows

plug and play principle. Master identifies that it is a monitoring node based on the packet types that the monitoring node sends.

(3) **Updating slave IPs**

Monitoring node sends requests to master time to time to know the IP of the current slaves present in the subnet. After every request, if there is any change in the slave IPs, it updates the configuration of Grafana accordingly.

(4) **Metrics visualized in Grafana**

- (a) CPU usage pattern of every slave (not the exact % values, but visualizing the ups and downs via graph)
- (b) Metrics exposed by every slave

2.7 Load Balancing Algorithms

The following algorithms have been implemented:

(1) **First Available**

This algorithm allocates the task to the first available slave (in the order of the slaves in the slave pool) that can accommodate the task.

(2) **Round Robin**

This algorithm maintains the list of available slaves in a particular order. When a new task is received, it allocates it to the next slave as per the order in the list. On reaching the end of the list, it moves back to the beginning.

This algorithm does not take into account the current load of the slave. However, it does ensure not to allocate a task to a slave whose maximum permissible load is more than the load of the task.

(3) **Least Load**

This algorithm allocates a task to the slave that currently has the least load among all slaves. Only those slaves whose maximum permissible load is at least the load of the task are considered.

This is similar to the Least Connection algorithm in literature, where nodes with the least number of connections are chosen. When using the number of connections as the load metric, this becomes the Least Connection Algorithm.

(4) **Least Difference**

Let the load availability of a slave be defined as the difference of the maximum permissible load and the current load of the slave. This algorithm allocates a task to the slave that has the least load availability among the set of slaves that can currently accommodate the task. The idea behind this algorithm is that if a slave s_1 has a large load availability, it can be used to service tasks that have a large load. If there is a task that has a small load, and there is a slave s_2 with a smaller load that can accommodate it, then it is preferable to assign it to s_2 , as it then avoids fragmenting the load availability of s_1 and leaves it free to service tasks with larger load that s_2 cannot handle. This improves the efficiency of the allocation of resources.

2.8 Tasks

There are a set of predefined tasks that can be requested from this software. They are predefined for demonstration purposes, however, it is easy to extend to other, more complicated tasks.

The tasks defined mainly compute mathematical functions on a single integer, and return an integer. This is done so that it becomes easier to send requests and receive responses in a single HTTP GET request. The core program can easily work with other types, such as strings, arrays, structures, etc.

The following tasks have been defined:

(1) **Finding the n^{th} Fibonacci number**

This task accepts a positive integer input n , and returns the n^{th} number in the Fibonacci series. (Specify size limits for validity).

(2) **Counting number of primes upto a number**

This task accepts a positive integer n and returns the number of primes in the range $[1, n]$. A naïve approach has intentionally been used in implementing this task so that it takes more time to complete it, which helps in demonstrating the effectiveness of the algorithms, and of the usage of load balancing.

3 EXTERNAL MODULES USED

The following external software/modules have been used:

- (1) **Prometheus:** An open-source monitoring solution with its own time series database. We used Prometheus's modules to expose the CPU metrics and exposed our own metrics to Prometheus for monitoring. More about this can be found here <https://prometheus.io/>
- (2) **Grafana:** Used for visualizing the data gathered by Prometheus. More about this can be found here <https://grafana.com/>

4 CODE STRUCTURE

4.1 Overview

The code is organized in the following directories:

- (1) `master_src` - contains the source files for the master module
- (2) `slave_src` - contains the source files for the slave module
- (3) `monitoring_src` - contains the source files for the monitoring module
- (4) `common` - contains utilities, types and functions common to all the modules
- (5) `cmd` - contains the entry points to run each module

The root directory contains a Makefile, for compiling the project using GNU make. Details have been provided in Section 6.

The organization for each is described in this section.

4.2 Master

The files are in the `master_src` directory.

4.2.1 `master.go`. This defines the data types for the data stored by the master node, the data stored for each task, and functions for basic operations on the master node.

4.2.2 `slave.go`. This defines data types to store information for each slave in the master module. This also includes functions for the master to communicate with the slaves.

4.2.3 `slavePool.go`. This defines types and functions for maintaining the slave pool in the master module.

4.2.4 connect.go. This defines types and functions for storing the connection information with slaves and to connect to new slaves in the master node.

4.2.5 algorithm.go. This implements the load balancing algorithms used by master. It consists of types to store information needed by the algorithms and functions to determine the slave to assign a task to.

4.2.6 http.go. This defines functions to create a front end HTTP server for the master node. Using the server, end users can send requests to master directly from the browser.

4.2.7 task.go. This defines functions for task creation and handling by master. It includes functions to assign tasks, create task request packets, and handle responses.

4.2.8 monitoring.go. Contains handling of requests from monitoring node.

4.3 Slave

The files are in the `slave_src` directory.

4.3.1 slave.go. This defines types to store data known by the slave, such as the list of tasks and the current load. It also defines types to store information about tasks and metrics, and functions to perform basic operations on the slave.

4.3.2 connect.go. This consists of functions for the slave to connect to master, and for communication of load and task requests and responses with master.

4.3.3 task.go. This defines functions for the slave to handle received tasks and to create and send appropriate responses to master.

4.3.4 task_list.go. This contains the predefined tasks to be run by the slave.

4.3.5 metric.go. This contains exposing of slave metrics to prometheus.

4.4 Monitor

The files are in `monitoring_src` directory.

4.4.1 monitoring.go. This defines the basic types needed and contains start point for the monitor.

4.4.2 connect.go. This consists of functions for the monitor to connect to master, and for communication of slave IPs and responses with master.

4.4.3 templates.go. This contains the JSON templates used for updating Grafana.

4.5 Common files and Utilities

The files are in `common` directory.

4.5.1 constants/constants.go. This defines constants used in the program. They include fixed port numbers, addresses, and timeouts.

4.5.2 logger/logger.go. This defines functions for displaying effective log messages.

4.5.3 packets/packets.go. This defines types for packets used for different tasks.

4.5.4 utility/utility.go. This defines common utility functions used by the program.

5 MEMBER CONTRIBUTIONS

- Ganesh Vernekar (CS15BTECH11018)

- (1) Implemented initial backbone structure of the codebase from which all the components were implemented - which included defining packet and packet handling mechanism, and the directory structure.
- (2) Implemented entire plug and play mechanism of master and slave as explained in 2.5.
- (3) Implemented slave pool in the master to manage the slaves and also added garbage collection to remove dead slaves.
- (4) Implemented the HTTP server in the master to receive requests from outside.
- (5) Implemented entire monitoring which included - pluggable monitor node, constantly getting slave IPs from master and updating Grafana from time to time, exposing of metrics in slave to Prometheus.
- (6) Added proper logging mechanism using golang logging mechanism.

- Sukrut Rao (CS15BTECH11036)

- (1) Implemented all the four load balancing algorithms - First Available, Round Robin, Least Load, and Least Difference.
- (2) Defined all packets, types, and implemented the mechanism to send tasks from the master to the slave, and to send task responses from the slave to the master.
- (3) Implemented all functions to create tasks, and to handle tasks both in master and slave.
- (4) Implemented all the predefined tasks used in the project - Fibonacci and Counting Primes.

6 USAGE INSTRUCTIONS

6.1 Prerequisites

- (1) Golang v1.10
- (2) git
- (3) \$GOPATH/bin in \$PATH

6.2 Installing

```
go get github.com/GoodDeeds/load-balancer
cd $GOPATH/src/github.com/GoodDeeds/load-balancer
make
```

6.3 Running the program

To run master

```
./master <algorithm_name (optional)>
```

List of algorithm names can be seen in `master_src/master.go`.

To run slave

```
make slave_preproc
```

Plug-and-Play Load Balancer

```
./slave
```

To run monitor

```
make monitor_preproc
```

Then create an API key in grafana

```
./monitoring <Grafana API key>
```

6.4 Screenshots

These are included after the references.

7 LIMITATIONS AND POSSIBLE IMPROVEMENTS

- (1) There could be a comparison of the load balancing algorithms using our project program.
- (2) The load metric used in the project could be improved.
- (3) Tests could be conducted using more complicated tasks.
- (4) More algorithms, incorporating other features, such as memory usage, response latency, etc. can be added as an extension to this project.

REFERENCES

- (1) <https://github.com/cloudfoundry/gosigar>
- (2) <https://kemptechnologies.com/in/load-balancer/load-balancing-algorithms-techniques/>
- (3) <https://github.com/prometheus/prometheus>
- (4) https://github.com/prometheus/node_exporter
- (5) <https://github.com/grafana/grafana>
- (6) <https://golang.org/>

The image shows two terminal windows side-by-side. The left window shows the output of the 'master' script, which starts the load balancer and connects four slave nodes. The right window shows the output of the 'monitoring' script, which starts the monitoring node and updates the Grafana dashboard.

```

codesome@codesome: ~/gopath/src/github.com/GoodDeeds/load-balancer
codesome@codesome:~/gopath/src/github.com/GoodDeeds/load-balancer$ ./master
level=INFO time=23:05:15.440 file=http.go:39 ▶ msg="Starting the server"
level=INFO time=23:05:15.440 file=master.go:98 ▶ msg="Master running"
level=INFO time=23:05:15.440 file=master.go:188 ▶ msg="Garbage collection routine started"
level=INFO time=23:05:16.505 file=connect.go:98 ▶ msg="Connection request" ip="192.168.0.107" port="46953"
level=INFO time=23:05:16.507 file=connect.go:158 ▶ msg="Connection request granted" ip="192.168.0.107" port="46953"
level=INFO time=23:05:18.603 file=connect.go:98 ▶ msg="Connection request" ip="192.168.0.100" port="34089"
level=INFO time=23:05:18.603 file=connect.go:158 ▶ msg="Connection request granted" ip="192.168.0.100" port="34089"
level=INFO time=23:05:20.204 file=connect.go:98 ▶ msg="Connection request" ip="192.168.0.100" port="47506"
level=INFO time=23:05:20.205 file=connect.go:158 ▶ msg="Connection request granted" ip="192.168.0.100" port="47506"
level=INFO time=23:05:23.708 file=connect.go:174 ▶ msg="Monitor connection request" ip="192.168.0.100" port="58884"
level=INFO time=23:05:23.709 file=connect.go:225 ▶ msg="Monitor connection request granted" ip="192.168.0.100" port="58884"
level=INFO time=23:05:23.709 file=monitoring.go:49 ▶ msg="Monitor request"
level=INFO time=23:05:23.709 file=monitoring.go:93 ▶ msg="Sending slave ip"
-

codesome@codesome:~/gopath/src/github.com/GoodDeeds/load-balancer$ ./monitoring eyJrIj0tWEZnaVh0S1hYcG9sMWMtMDZlbnV5bDNGU0t1NGdsZS5EULCjU1j0lQWRTaW41LCZpZCI6MX0=
level=INFO time=23:05:23.708 file=monitoring.go:47 ▶ msg="Monitor running"
level=INFO time=23:05:23.709 file=connect.go:104 ▶ msg="Connection response" ack="true" server_ip="192.168.0.100"
level=INFO time=23:05:23.709 file=monitoring.go:134 ▶ msg="Adding datasource" ip="192.168.0.107"
level=INFO time=23:05:23.874 file=monitoring.go:134 ▶ msg="Adding datasource" ip="192.168.0.100"
level=INFO time=23:05:24.041 file=monitoring.go:134 ▶ msg="Adding datasource" ip="192.168.0.100"
level=INFO time=23:05:24.091 file=monitoring.go:152 ▶ msg="Add datasource failed" status="409" ip="192.168.0.100"
level=INFO time=23:05:24.091 file=monitoring.go:176 ▶ msg="Updating dashboard"
-

```

Figure 1: Left: Master started and 4 slaves are connected, Right: Monitoring node started and updated grafana

The image shows two terminal windows side-by-side. The left window shows the output of the 'slave' script, which starts the slave node and connects to the master. The right window shows the output of the 'slave' script, which starts the slave node and connects to the master.

```

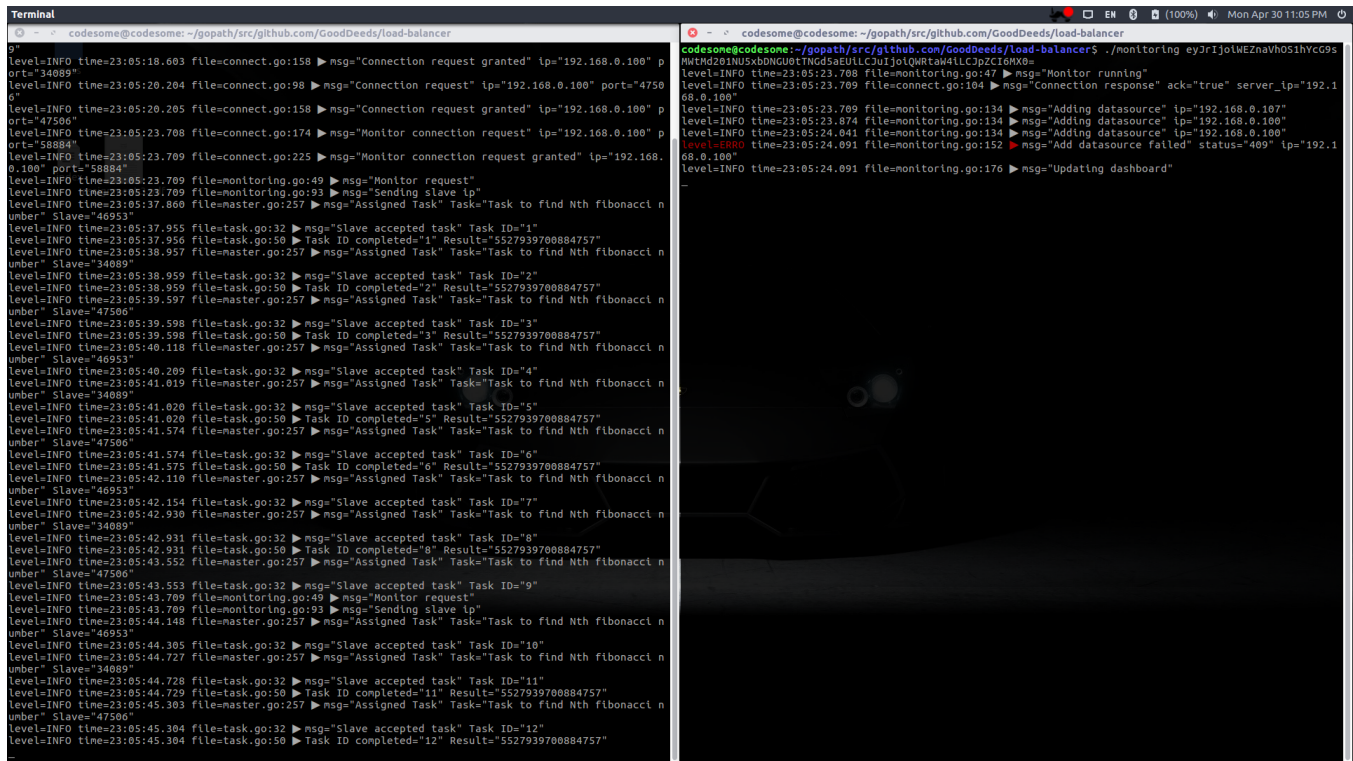
codesome@codesome: ~/gopath/src/github.com/GoodDeeds/load-balancer
codesome@codesome:~/gopath/src/github.com/GoodDeeds/load-balancer$ ./slave
level=INFO time=23:05:18.568 file=metric.go:40 ▶ msg="Starting the server"
level=INFO time=23:05:18.602 file=metric.go:92 ▶ msg="Server and metrics started"
level=INFO time=23:05:18.602 file=connect.go:130 ▶ loadReqPort="37549" reqSendPort="36545"
level=INFO time=23:05:18.603 file=connect.go:114 ▶ msg="Connection response" ack="true" server_ip="192.168.0.100"
level=INFO time=23:05:18.603 file=slave.go:82 ▶ msg="Slave running"
-

codesome@codesome:~/gopath/src/github.com/GoodDeeds/load-balancer$ ./slave
level=INFO time=23:05:20.157 file=metric.go:40 ▶ msg="Starting the server"
level=INFO time=23:05:20.204 file=metric.go:92 ▶ msg="Server and metrics started"
level=INFO time=23:05:20.204 file=connect.go:130 ▶ loadReqPort="44224" reqSendPort="44690"
level=INFO time=23:05:20.205 file=connect.go:114 ▶ msg="Connection response" ack="true" server_ip="192.168.0.100"
level=INFO time=23:05:20.205 file=slave.go:82 ▶ msg="Slave running"
-

```

Figure 2: 2 Slaves started on a system. (2 more in other system in same subnet, not in this pic)

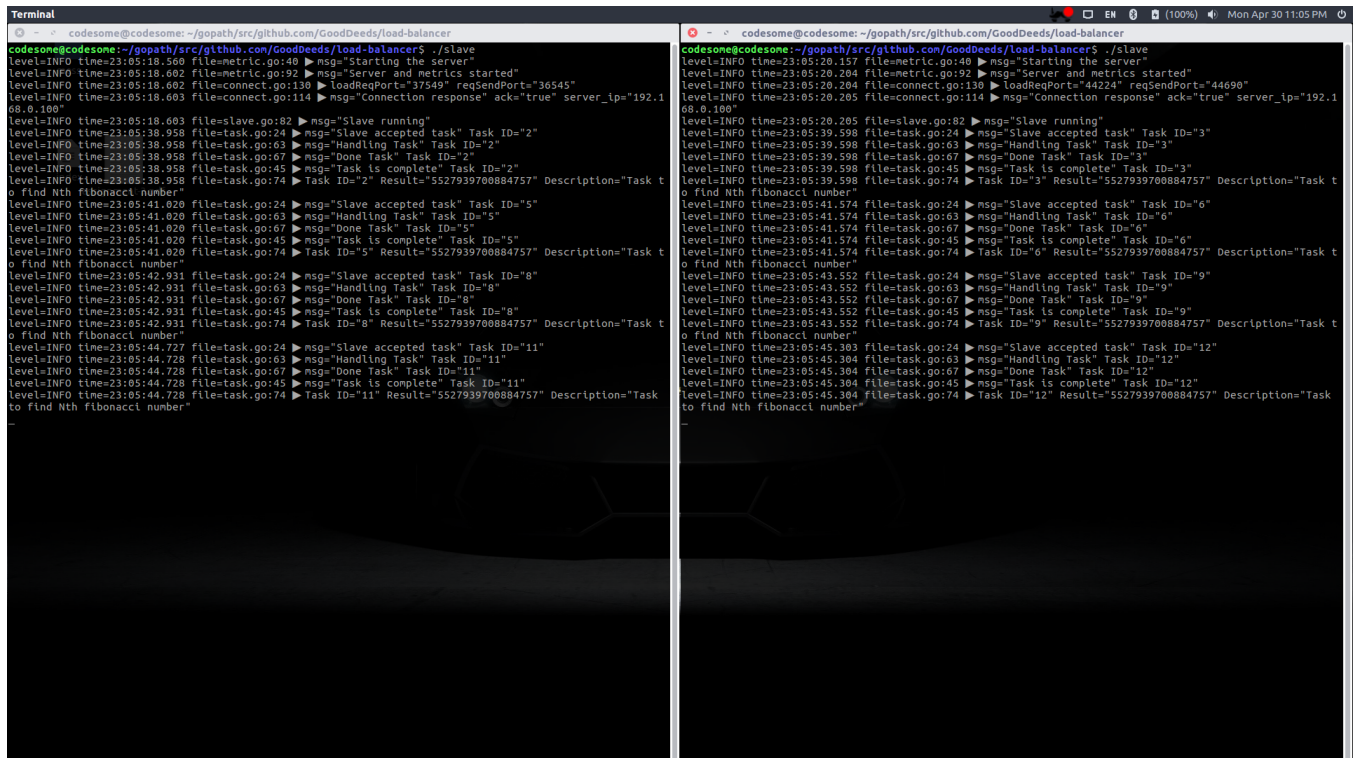
Plug-and-Play Load Balancer



```
codesome@codesome: ~/gopath/src/github.com/GoodDeeds/load-balancer
level=INFO time=23:05:18.603 file=connect.go:158 ▶ msg="Connection request granted" ip="192.168.0.100" p
ort="34089"
level=INFO time=23:05:20.204 file=connect.go:98 ▶ msg="Connection request" ip="192.168.0.100" port="4750
6"
level=INFO time=23:05:20.205 file=connect.go:158 ▶ msg="Connection request granted" ip="192.168.0.100" p
ort="47506"
level=INFO time=23:05:23.708 file=connect.go:174 ▶ msg="Monitor connection request" ip="192.168.0.100" p
ort="58884"
level=INFO time=23:05:23.709 file=connect.go:225 ▶ msg="Monitor connection request granted" ip="192.168.
0.100" port="58884"
level=INFO time=23:05:23.709 file=monitoring.go:49 ▶ msg="Monitor request"
level=INFO time=23:05:23.709 file=monitoring.go:93 ▶ msg="Sending slave ip"
level=INFO time=23:05:37.860 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="46953"
level=INFO time=23:05:37.955 file=task.go:32 ▶ msg="Slave accepted task" Task ID="1"
level=INFO time=23:05:37.956 file=task.go:50 ▶ Task ID completed="1" Result="5527939700884757"
level=INFO time=23:05:38.957 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="34089"
level=INFO time=23:05:38.959 file=task.go:32 ▶ msg="Slave accepted task" Task ID="2"
level=INFO time=23:05:38.959 file=task.go:50 ▶ Task ID completed="2" Result="5527939700884757"
level=INFO time=23:05:39.597 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="47506"
level=INFO time=23:05:39.598 file=task.go:32 ▶ msg="Slave accepted task" Task ID="3"
level=INFO time=23:05:39.598 file=task.go:50 ▶ Task ID completed="3" Result="5527939700884757"
level=INFO time=23:05:40.118 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="46953"
level=INFO time=23:05:40.209 file=task.go:32 ▶ msg="Slave accepted task" Task ID="4"
level=INFO time=23:05:41.019 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="34089"
level=INFO time=23:05:41.020 file=task.go:32 ▶ msg="Slave accepted task" Task ID="5"
level=INFO time=23:05:41.020 file=task.go:50 ▶ Task ID completed="5" Result="5527939700884757"
level=INFO time=23:05:41.574 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="47506"
level=INFO time=23:05:41.574 file=task.go:32 ▶ msg="Slave accepted task" Task ID="6"
level=INFO time=23:05:41.575 file=task.go:50 ▶ Task ID completed="6" Result="5527939700884757"
level=INFO time=23:05:42.110 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="46953"
level=INFO time=23:05:42.154 file=task.go:32 ▶ msg="Slave accepted task" Task ID="7"
level=INFO time=23:05:42.930 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="34089"
level=INFO time=23:05:42.931 file=task.go:32 ▶ msg="Slave accepted task" Task ID="8"
level=INFO time=23:05:42.931 file=task.go:50 ▶ Task ID completed="8" Result="5527939700884757"
level=INFO time=23:05:43.552 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="47506"
level=INFO time=23:05:43.553 file=task.go:32 ▶ msg="Slave accepted task" Task ID="9"
level=INFO time=23:05:43.709 file=monitoring.go:49 ▶ msg="Monitor request"
level=INFO time=23:05:43.709 file=monitoring.go:93 ▶ msg="Sending slave ip"
level=INFO time=23:05:44.148 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="46953"
level=INFO time=23:05:44.305 file=task.go:32 ▶ msg="Slave accepted task" Task ID="10"
level=INFO time=23:05:44.727 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="34089"
level=INFO time=23:05:44.728 file=task.go:32 ▶ msg="Slave accepted task" Task ID="11"
level=INFO time=23:05:44.728 file=task.go:50 ▶ Task ID completed="11" Result="5527939700884757"
level=INFO time=23:05:45.303 file=master.go:257 ▶ msg="Assigned Task" Task="Task to find Nth fibonaccl n
umber" Slave="47506"
level=INFO time=23:05:45.304 file=task.go:32 ▶ msg="Slave accepted task" Task ID="12"
level=INFO time=23:05:45.304 file=task.go:50 ▶ Task ID completed="12" Result="5527939700884757"

codesome@codesome: ~/gopath/src/github.com/GoodDeeds/load-balancer$ ./monitoring ey3rIj0LWZnaVh0S1hYcC9s
MHtM201NUSx0NGU0TNGdSaEUlLCJuij0LQWRTaW41LCJpZCI6MkX0=
level=INFO time=23:05:23.789 file=monitoring.go:134 ▶ msg="Monitor running"
level=INFO time=23:05:23.789 file=connect.go:104 ▶ msg="Connection response" ack="true" server_ip="192.1
68.0.100"
level=INFO time=23:05:23.789 file=monitoring.go:134 ▶ msg="Adding datasource" ip="192.168.0.107"
level=INFO time=23:05:23.874 file=monitoring.go:134 ▶ msg="Adding datasource" ip="192.168.0.100"
level=INFO time=23:05:24.041 file=monitoring.go:134 ▶ msg="Adding datasource" ip="192.168.0.100"
level=ERROR time=23:05:24.091 file=monitoring.go:152 ▶ msg="Add datasource failed" status="409" ip="192.1
68.0.100"
level=INFO time=23:05:24.091 file=monitoring.go:176 ▶ msg="Updating dashboard"
```

Figure 3: Left: Logs of master while receiving tasks



```
codesome@codesome: ~/gopath/src/github.com/GoodDeeds/load-balancer
codesome@codesome: ~/gopath/src/github.com/GoodDeeds/load-balancer$ ./slave
level=INFO time=23:05:18.568 file=metric.go:40 ▶ msg="Starting the server"
level=INFO time=23:05:18.602 file=metric.go:92 ▶ msg="Server and metrics started"
level=INFO time=23:05:18.602 file=connect.go:130 ▶ loadReqPort="37549" reqSendPort="36545"
level=INFO time=23:05:18.603 file=connect.go:114 ▶ msg="Connection response" ack="true" server_ip="192.1
68.0.100"
level=INFO time=23:05:18.603 file=slave.go:82 ▶ msg="Slave running"
level=INFO time=23:05:38.958 file=task.go:24 ▶ msg="Slave accepted task" Task ID="2"
level=INFO time=23:05:38.958 file=task.go:63 ▶ msg="Handling Task" Task ID="2"
level=INFO time=23:05:38.958 file=task.go:67 ▶ msg="Done Task" Task ID="2"
level=INFO time=23:05:38.958 file=task.go:45 ▶ msg="Task is complete" Task ID="2"
level=INFO time=23:05:38.958 file=task.go:74 ▶ Task ID="2" Result="5527939700884757" Description="Task t
o find Nth fibonaccl number"
level=INFO time=23:05:41.020 file=task.go:24 ▶ msg="Slave accepted task" Task ID="5"
level=INFO time=23:05:41.020 file=task.go:63 ▶ msg="Handling Task" Task ID="5"
level=INFO time=23:05:41.020 file=task.go:67 ▶ msg="Done Task" Task ID="5"
level=INFO time=23:05:41.020 file=task.go:45 ▶ msg="Task is complete" Task ID="5"
level=INFO time=23:05:41.020 file=task.go:74 ▶ Task ID="5" Result="5527939700884757" Description="Task t
o find Nth fibonaccl number"
level=INFO time=23:05:42.931 file=task.go:24 ▶ msg="Slave accepted task" Task ID="8"
level=INFO time=23:05:42.931 file=task.go:63 ▶ msg="Handling Task" Task ID="8"
level=INFO time=23:05:42.931 file=task.go:67 ▶ msg="Done Task" Task ID="8"
level=INFO time=23:05:42.931 file=task.go:45 ▶ msg="Task is complete" Task ID="8"
level=INFO time=23:05:42.931 file=task.go:74 ▶ Task ID="8" Result="5527939700884757" Description="Task t
o find Nth fibonaccl number"
level=INFO time=23:05:44.727 file=task.go:24 ▶ msg="Slave accepted task" Task ID="11"
level=INFO time=23:05:44.728 file=task.go:63 ▶ msg="Handling Task" Task ID="11"
level=INFO time=23:05:44.728 file=task.go:67 ▶ msg="Done Task" Task ID="11"
level=INFO time=23:05:44.728 file=task.go:45 ▶ msg="Task is complete" Task ID="11"
level=INFO time=23:05:44.728 file=task.go:74 ▶ Task ID="11" Result="5527939700884757" Description="Task
to find Nth fibonaccl number"

codesome@codesome: ~/gopath/src/github.com/GoodDeeds/load-balancer
codesome@codesome: ~/gopath/src/github.com/GoodDeeds/load-balancer$ ./slave
level=INFO time=23:05:20.157 file=metric.go:40 ▶ msg="Starting the server"
level=INFO time=23:05:20.204 file=metric.go:92 ▶ msg="Server and metrics started"
level=INFO time=23:05:20.204 file=connect.go:130 ▶ loadReqPort="44224" reqSendPort="44690"
level=INFO time=23:05:20.205 file=connect.go:114 ▶ msg="Connection response" ack="true" server_ip="192.1
68.0.100"
level=INFO time=23:05:20.205 file=slave.go:82 ▶ msg="Slave running"
level=INFO time=23:05:39.598 file=task.go:24 ▶ msg="Slave accepted task" Task ID="3"
level=INFO time=23:05:39.598 file=task.go:63 ▶ msg="Handling Task" Task ID="3"
level=INFO time=23:05:39.598 file=task.go:67 ▶ msg="Done Task" Task ID="3"
level=INFO time=23:05:39.598 file=task.go:45 ▶ msg="Task is complete" Task ID="3"
level=INFO time=23:05:39.598 file=task.go:74 ▶ Task ID="3" Result="5527939700884757" Description="Task t
o find Nth fibonaccl number"
level=INFO time=23:05:41.574 file=task.go:24 ▶ msg="Slave accepted task" Task ID="6"
level=INFO time=23:05:41.574 file=task.go:63 ▶ msg="Handling Task" Task ID="6"
level=INFO time=23:05:41.574 file=task.go:67 ▶ msg="Done Task" Task ID="6"
level=INFO time=23:05:41.574 file=task.go:45 ▶ msg="Task is complete" Task ID="6"
level=INFO time=23:05:41.574 file=task.go:74 ▶ Task ID="6" Result="5527939700884757" Description="Task t
o find Nth fibonaccl number"
level=INFO time=23:05:43.552 file=task.go:24 ▶ msg="Slave accepted task" Task ID="9"
level=INFO time=23:05:43.552 file=task.go:63 ▶ msg="Handling Task" Task ID="9"
level=INFO time=23:05:43.552 file=task.go:67 ▶ msg="Done Task" Task ID="9"
level=INFO time=23:05:43.552 file=task.go:45 ▶ msg="Task is complete" Task ID="9"
level=INFO time=23:05:43.552 file=task.go:74 ▶ Task ID="9" Result="5527939700884757" Description="Task t
o find Nth fibonaccl number"
level=INFO time=23:05:45.303 file=task.go:24 ▶ msg="Slave accepted task" Task ID="12"
level=INFO time=23:05:45.304 file=task.go:63 ▶ msg="Handling Task" Task ID="12"
level=INFO time=23:05:45.304 file=task.go:67 ▶ msg="Done Task" Task ID="12"
level=INFO time=23:05:45.304 file=task.go:45 ▶ msg="Task is complete" Task ID="12"
level=INFO time=23:05:45.304 file=task.go:74 ▶ Task ID="12" Result="5527939700884757" Description="Task
to find Nth fibonaccl number"
```

Figure 4: Logs of slave while receiving tasks

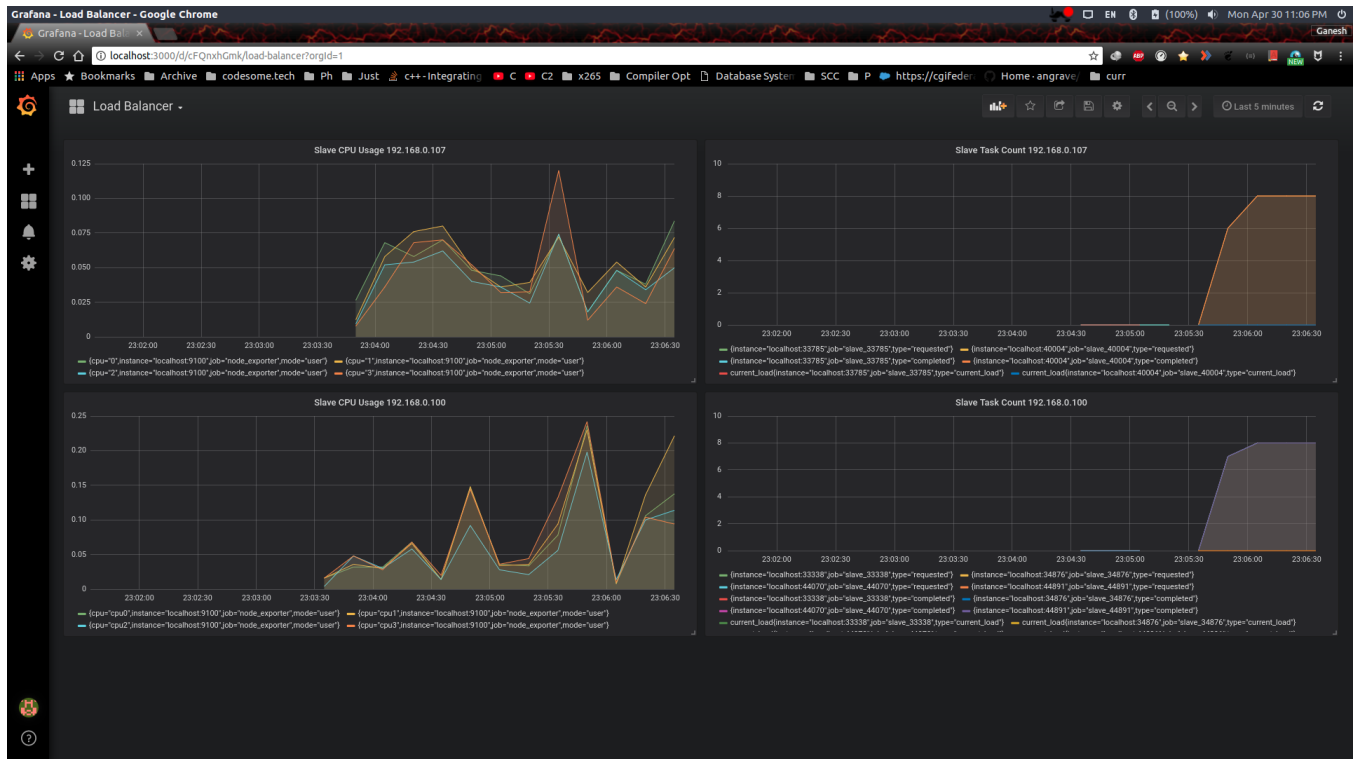


Figure 5: Monitoring in Grafana - Top 2 graphs are of one system, bottom 2 are of another system. Left panels are CPU usage pattern, Right panels are slave metrics