Major Project Report

On

Distributed Computing Over User Agents

In partial fulfilment of requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY:

Anamika(14507)

Sheweta(14509)

Sahil Singh(14544)

Utkarsh Raj(14565)

Shantanu Shukla(14566)

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF NATIONAL INSTITUTE TECHNOLOGY

HAMIRPUR-177005

MAY 2018

CERTIFICATE

We, hereby certify that the work which is being presented in report entitled "**Distributed Computing Over User Agents**" in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Computer Science and Engineering and submitted to the department of computer science and engineering at National Institute of Technology Hamirpur is an authentic record of our own work carried out under the supervision of Dr. Pardeep Singh, Assistant Professor, Department Computer Science and Engineering, National Institute of Technology Hamirpur .

To the best of our knowledge, the matter presented in this report has not been submitted to any other University/ Institute for the award of any Degree or Diploma.

Signature *Signature*

Anamika(14507) **Sheweta(14509)**

Signature *Signature*

Sahil Singh(14544) **Utkarsh Raj(14565)**

\

Signature

Shantanu Shukla(14566)

This is to certify that the above statement made by the candidates is correct to the best of our knowledge.

Date:

Signature of Supervisor

Dr.Pardeep Singh

Head of Department

Computer Science and Engineering Department

National Institute of Technology Hamirpur(HP)

ACKNOWLEDGEMENTS

We consider our self privileged to express gratitude and respect towards all those who guided us through this project. It is with our hearty gratitude that we acknowledge their contributions to this project. We would like to express our sincere gratitude and heart full thanks to Dr. Pardeep Singh for his unflinching support and guidance, valuable suggestions and expert advice. His words of wisdom and expertise in subject matter were of immense help throughout the duration of this project.We also take the opportunity to thank all the faculty of Department of Computer Science and Engineering, NIT Hamirpur for helping us by providing necessary knowledge base and resources. Last but not the least we thank our parents and all our mates whose words of encouragement and criticism have molded this report in its present form.

Anamika(14507)
Sheweta(14509)
Sahil Rangra(14544)
Utkarsh Raj(14565)
Shantanu Shukla(14566)

ABSTRACT

The world runs on Internet. The Internet has become the fuel for all the tasks around the globe. Internet can be accessed with the help of special applications called Web Browsers.Consequently the Web browsers have become one of the most used softwares in the world.Web browsers allow us to access web pages which can be written in a number of languages like HTML, CSS, JavaScript. A site written only in JavaScript, HTML, CSS can be accessed by anyone with a Web Browser irrespective of which platform they are using it on. JavaScript is used by 94.9% of all the websites. This cross-platform capability provided by Web Browsers can be used for distributed computing. This is also reinforced by the study on performances of languages which proved that JavaScript is very capable language for a lot of computation tasks. To process exceptionally large computation tasks,the task can be converted into a JavaScript oriented tasks (if possible) , it can then be sent over to the client where their browser will take the code and the data, perform computations on it. After computation is over on all smaller fragments of the task, the results from each fragment can be sent back, merged and then compiled. This project takes into consideration the working implementation for distributed computing over web browsers.

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
DOM	Document Object Model
HTTP	Hypertext markup language
JSON	JavaScript Object Notation
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
XML	Extensible markup language
XHR	XMLHttpRequest

CHAPTER -1

INTRODUCTION

Web Browser is one of the most commonly used softwares in the world. After all, it allows us to access the Internet. In the recent years, as of June 2017, 51% of the world's population has internet access [1]. Along with the increase in the users, the infrastructure to access it has also grown at a fervent pace. In a recent study by speedtest.net where it studied over 133 countries where each country had at least 3,333 unique tests. It reported that as of April, 2018 the global average download speed was 45.07 Mbps and average global upload speed was 21.93 Mbps [3]. This allows us to use the Internet for distributed computing over the Internet which was previously infeasible due to speed constraints.

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997. There have been a lot of improvements in JavaScript in the recent years, which has improved its performance considerably. There also have been new features added in recent years which has allowed for JavaScript to be used as a language suitable for distributed computing.

1. Web sockets
2. Web workers

1.1 Web workers

Web Workers has made this possible to run operation of the script in background thread which is separated from the main execution thread of web application .The aftermath of this is that time taking processing can be done in a distinct thread. It allows the running of thread without being blocked and minimize the slowing down of the thread.
It is basically an object which is created using a constructor(i.e Worker()) and runs a named JavaScript file- and this file has code which will run in the worker thread. Worker used to run in the additional global context that is distinct from the present window.
So, this factor is shown by either a DedicatedWorkerGlobalScope object or can be represented by the Shared Worker Global Scope. And you can run the code whatever code you like to see inside the worker thread which should be with some of the exceptions.

You can use varied number of the items which are available under the window, which includes WebSockets, and some of the mechanisms of data storage like for instance IndexedDB and the Firefox OS-only Data Store API. For the deep details have a look at the functions and class which are available to the worker.

Data is transferred in between the main thread and the workers with the help of system of messages - and postMessage() method is used to send the messages and for the reply of message onmessage event handler is used for the both side of communication. (the event's data property of the message contain the message). The data is replicated and sharing of data is not allowed.

In turn workers can create new workers, until the workers are introduced within the same agent as the source page. Additionally , workers can use the XMLHttpRequest for the network I/O. Including the exception, addition to the dedicated workers , there are some other types of worker:

(i)Share workers are the workers that can be employed by the different scripts running in distinct windows, IFrames , etc., until they are in the same domain. They are just complicated than the dedicated workers- and the scripts must communicate through an active port. Read the SharedWorker for the deeper details.

(ii)Service Workers factually act as a proxy servers that can sit + essentially act as proxy servers that sit between web applications, the browser, and the network (when available). They are intended, among other things, to enable the creation of effective offline experiences, intercept network requests and take appropriate action based on whether the network is available, and update assets residing on the server. They will also allow access to push notifications and background sync APIs.

(iii)Chrome Workers are a Firefox-only type of worker that you can use if you are developing add-ons and want to use workers in extensions and have access to js-ctypes in your worker. See ChromeWorker for more details.

(iv)Audio Workers provide the ability for direct scripted audio processing to be done inside a web worker context.

n that the responseXML and channel attributes on XMLHttpRequest always return null.

1.2 Web Sockets

The Web Socket simply defines a socket connection between a client and a server . In a simple language: Websockets establish a continuous connection between the client and the server and both clients and servers can start sending data anytime.

In today's era of advancing technology, WebSockets play an active role, wherein their use makes it possible to initiate an interactive communication session among various user's browser and all the servers.. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.

WebSocket protocol brings real time communication in web browsers to a new level. Daily, new products are designed to stay permanently connected to the web. WebSocket is the technology enabling this revolution. WebSockets are supported by all current browsers, but it is still a new technology in constant evolution.

WebSockets are slowly replacing older client-server communication technologies. As opposed to comet-like technologies WebSockets' remarkable performances is a result of the protocol's fully duplex nature and because it doesn't rely on HTTP communications.

WebSockets allow applications to have a bidirectional channel to a URI endpoint. Sockets can send and receive messages and respond to opening or closing a WebSocket. Although not part of the specification, two-way communication can be achieved in several other ways, including Comet (AJAX with long polling), Bayeux, and BOSH.

Listing 1-1 shows the code to create a WebSocket that talks to the echo server endpoint. After creating the socket, we set up the functions to be executed when the socket is opened, closed, receives a message, or throws an error. Next, a "Hello World!" message is sent, and the browser displays "Hello World!" upon receipt of the return message.

Listing 1-1. WebSocket Code for Echoing a Message

```
var socket = new WebSocket(ws://websockets.org:8787/echo);
socket.onopen = function(evt) { console.log("Socket opened");};
socket.onclose = function(evt) {console.log("Socket closed");};
socket.onmessage = function(evt){console.log(evt.data);};
socket.onerror = function(evt) {console.log("Error: "+evt.data);};
socket.send("Hello World!");
```

Web socket is a tcp protocol which is not like HTTP. Both protocols are located at layer 7 in the OSI model and, as such, depend on TCP at layer 4. Although they are different, RFC 6455 states that WebSocket "is designed to work over HTTP ports 80 and 443 as well as to support HTTP proxies and intermediaries" thus making it compatible with the HTTP protocol. To achieve compatibility, the WebSocket handshake uses the HTTP Upgrade header to change from the HTTP protocol to the WebSocket protocol.

CHAPTER-2

LITERATURE REVIEW

F. Boldrin in his paper proposed a new approach for distributed computing. The main novelty consists in the exploitation of web browsers as clients, thanks to the availability of Javascript and AJAX. The described solution has two main advantages: it is client- free, so no additional programs have to be installed to perform the computation, and it requires low CPU usage, so clientside computation is no invasive for users [4]. The solution is developed using AJAX technology embedding a pseudoclient into a web page that hosts the computation. While users browse the hosting web page, computation takes place resolving single subproblems and sending the solution to the serverside part of the system. The new architecture has been tested through various performance metrics by implementing two examples of distributed computing, the RSA crack and the correlation index between genetic data sets. Results have shown good feasibility of this approach.

1. **Web Sockets** has removed the need to use AJAX, it provides connection with much less overhead. Since it allows to send messages both ways we can use it to send signals from the server, eliminating need for polling from client-side.
2. **Web Workers** allows certain processes to run in the background without affecting the user experience adversely. The computation which is taken in by the client’s browser does not compete with the resources for web browsing, which could be a big reason for non-adoption of our scheme.
3. **Network Information API** allows us to distinguish the medium with which device has been connected, based on this information we can block certain devices from accessing the code
4. **Network Timing API** allows to benchmark the connection based on page load time, network latency, etc. This is very useful to prevent clients from fetching the sub-jobs if they do not follow certain minimum requirements.

CHAPTER - 3

INSPIRATION

3.1 The Great Internet Mersenne Prime Search:

In mathematics, a **Mersenne prime** is a prime number that is one less than a power of two. That is, it is a prime number of the form $M_n = 2^n - 1$ for some integer n. They are named after Marin Mersenne, a French Minim friar, who studied them in the early 17th century.

The exponents n which give Mersenne primes are 2, 3, 5, 7, 13, 17, 19, 31, ... (sequence A000043 in the OEIS) and the resulting Mersenne primes are 3, 7, 31, 127, 8191, 131071, 524287, 2147483647, ... (sequence A000668 in the OEIS).

If n is a composite number then so is $2^n - 1$. ($2^{ab} - 1$ is divisible by both $2^a - 1$ and $2^b - 1$.) This definition is therefore equivalent to a definition as a prime number of the form $M_p = 2^p - 1$ for some prime p.

Since 1997, all newly found Mersenne primes have been discovered by the Great Internet Mersenne Prime Search (GIMPS), a **distributed computing** project on the Internet.

With the advent of computers to perform number-crunching tasks formerly done by humans, ever-larger Mersenne primes (and primes in general) have been found. The quest to find prime numbers is akin to other numerical searches done by computers.

It takes the most powerful computer a long time to check a large number to determine if it is prime, and an even longer time to determine if it is a Mersenne prime. For this reason, Mersenne primes are of particular interest to developers of strong encryption methods.

It is a part of the distributed computing project which has created inspiration to do more work in the field of distributed computing.

3.2 Allowing background Cryptominer to run instead of Ads

Salon.com has a new, cryptocurrency-driven strategy for making money when readers block ads. If you want to read Salon without seeing ads, you can do so—as long as you let the website use your spare computing power to mine some coins.

If you visit Salon with an ad blocker enabled, you might see a pop-up that asks you to disable the ad blocker or "Block ads by allowing Salon to use your unused computing power."

Salon explains what's going on in a new FAQ. "How does Salon make money by using my processing power?" the FAQ says. "We intend to use a small percentage of your spare processing power to contribute to the advancement of technological discovery, evolution, and innovation. For our beta program, we'll start by applying your processing power to help support the evolution and growth of blockchain technology and cryptocurrencies."

Coinhive, as a tool, is a JavaScript library that website owners can load on their site. When users access the site, the Coinhive JavaScript code library executes and mines for Monero for the site owner, but using the user's CPU resources.Coinhive claims that webmasters can remove ads from their sites, and load the Coinhive library and mine for Monero using a small portion of the user's CPU while the user is navigating the site. Site owners can make money and support their business, but without peppering their visitors with annoying ads.

Coinhive, as a tool, is a JavaScript library that website owners can load on their site. When users access the site, the Coinhive JavaScript code library executes and mines for Monero for the site owner, but using the user's CPU resources.Coinhive claims that webmasters can remove ads from their sites, and load the Coinhive library and mine for Monero using a small portion of the user's CPU while the user is navigating the site. Site owners can make money and support their business, but without peppering their visitors with annoying ads.

CHAPTER – 4

DESIGN, CONCEPTS AND SETUP

4.1 Network Information API

The Network Information API provides access to the connection type the system is using to communicate with the network, cellular, Wi-Fi, Bluetooth, etc. It also provides a means for scripts to be notified if the connection type changes. This is to allow developers to make dynamic changes to the DOM and/or inform the user that the network connection type has changed.

The current version of the Network Information API exposes a connection object that belongs to the window.navigator object. The connection object contains one property, type, which returns the user agent's connection type. The type property can have one of the following values:

- bluetooth
- cellular
- ethernet
- none
- wifi
- other
- unknown

Some of these values, such as bluetooth and wifi, are self-explanatory while others need a bit more explaining. The cellular type refers to a mobile connection, such as EDGE, 3G, 4G, etc. The other type means that the current connection type is not unknown, but it isn't any of the other types either. The unknown type means that the user agent has established a network connection, but it is unable to determine what the connection type is.

In addition to type, the Network Information API exposes the on type change event. It is fired every time the type of the network connection changes.

Developers can use the Network Information API to change some features based on the current connection type. For example, we can slow down a process that takes up significant bandwidth if we detect the device is using a mobile connection. The API also lets us assign a specific class to the html element, for example high-bandwidth, in the same way Modernize does.

4.2 Network Timing API

The Navigation Timing API provides data that can be used to measure the performance of a website. Unlike JavaScript-based libraries that have historically been used to collect similar information, the Navigation Timing API can be much more accurate and reliable. Navigation Timing API can be used to gather performance data on the client side which you can then transmit to a server using XMLHttpRequest or other techniques. Also, the API lets you measure data that was previously difficult to obtain, such as the amount of time needed to unload the previous page, how long domain lookups take, the total time spent executing the window's load handler, and so forth. The Navigation Timing API provides data that can be used to measure the performance of a website. Unlike JavaScript-based libraries that have historically been used to collect similar information, the Navigation Timing API can be much more accurate and reliable.

4.3 Concepts and usage

One can use the Navigation Timing API to gather performance data on the client side which you can then transmit to a server using XMLHttpRequest or other techniques. Also, the API lets us measure data that was previously difficult to obtain, such as the amount of time needed to unload the previous page, how long domain lookups take, the total time spent executing the window's load handler, and so forth.

4.4 Interfaces

Performance

The window performance property returns a Performance object. While this interface is defined by the High Resolution Time API, the Navigation Timing API adds two properties: timing and navigation, of the types below.

PerformanceTiming

Used as the type for the value of timing, objects of this type contain timing information that can provide insight into web page performance.

Performance Navigation

The type used to return the value of navigation, which contains information explaining the context of the load operation described by this Performance instance.

Performance Navigation Timing

Provides methods and properties to store and retrieve metrics regarding the browser's document navigation events. For example, this interface can be used to determine how much time it takes to load or unload a document.

The Navigation Timing API can be used to gather performance data on the client side to be sent to a server via XHR as well as measure data that was very difficult to measure by other means such as time to unload a previous page, domain lookup time, window.onload total time, etc.

Examples

Calculate the total page load time

To compute the total amount of time it took to load the page, you can use the following code:

```
var perfData = window.performance.timing;
var pageLoadTime = perfData.loadEventEnd - perfData.navigationStart;
```

This subtracts the time at which navigation began (navigationStart) from the time at which the load event handler returns (loadEventEnd). This gives you the perceived page load time.

Calculate request response time

We can calculate the time elapsed between the beginning of a request and the completion of getting the response using code like this: var connectTime = perfData.responseEnd - perfData.requestStart;

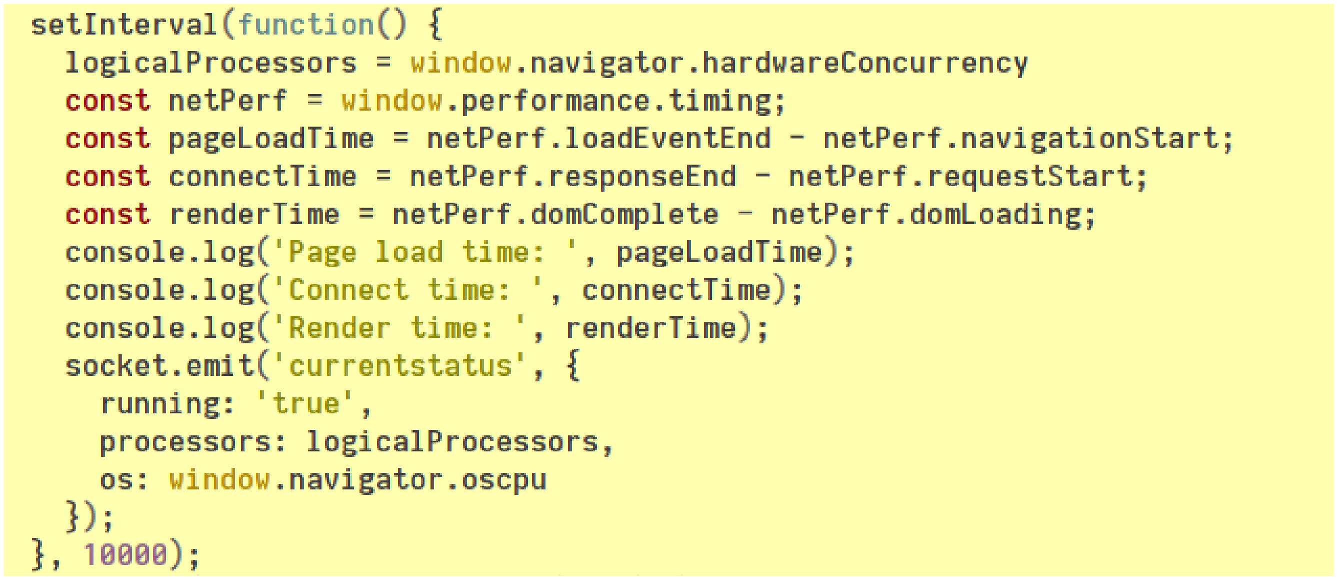
Here, the time at which the request was initiated (requestStart). from the time at which the response was finished being received (responseEnd).

Calculate page render time

As another example of an interesting piece of data you can obtain using the Navigation Timing API that you can't otherwise easily get, you can get the amount of time it took to render the page:

var renderTime = perfData.domComplete - perfData.domLoading;

This is obtained by starting with the time at which loading of the DOM and its dependencies is complete (domComplete) and subtracting from it the time at which parsing of the DOM began (domLoading).



```
setInterval(function() {
  logicalProcessors = window.navigator.hardwareConcurrency
  const netPerf = window.performance.timing;
  const pageLoadTime = netPerf.loadEventEnd - netPerf.navigationStart;
  const connectTime = netPerf.responseEnd - netPerf.requestStart;
  const renderTime = netPerf.domComplete - netPerf.domLoading;
  console.log('Page load time: ', pageLoadTime);
  console.log('Connect time: ', connectTime);
  console.log('Render time: ', renderTime);
  socket.emit('currentstatus', {
    running: 'true',
    processors: logicalProcessors,
    os: window.navigator.oscpu
  });
}, 10000);
```

Fig.4.1

4.5 Architecture of Distributed Computing over User Agents

1. **Compute Client** - This is the service user which wants to do computation. It will send the job with a POST request specifying the name of the job. Code to executed in the job and the Data on which processing will be performed.
2. **Compute Server** - This will take the POST request and divide the added job into smaller sub-jobs which can easily transmitted over the network. It will also establish connection with the user agent. It will send the sub-job information to participating User Agents and also assess the device and the network connection. After it has received results for all the sub-jobs it will merge them and allow it to be downloaded by the Compute Client.
3. **Participant Server** - This is the web site which serves the code that is used to execute connection from the User Agents to the Compute Server.
4. **User Agent** - This is the browser on which connection will be established between itself and the Compute Server. This connection is used to exchange sub-job - data and code and also return the result of sub-job after it has been executed.

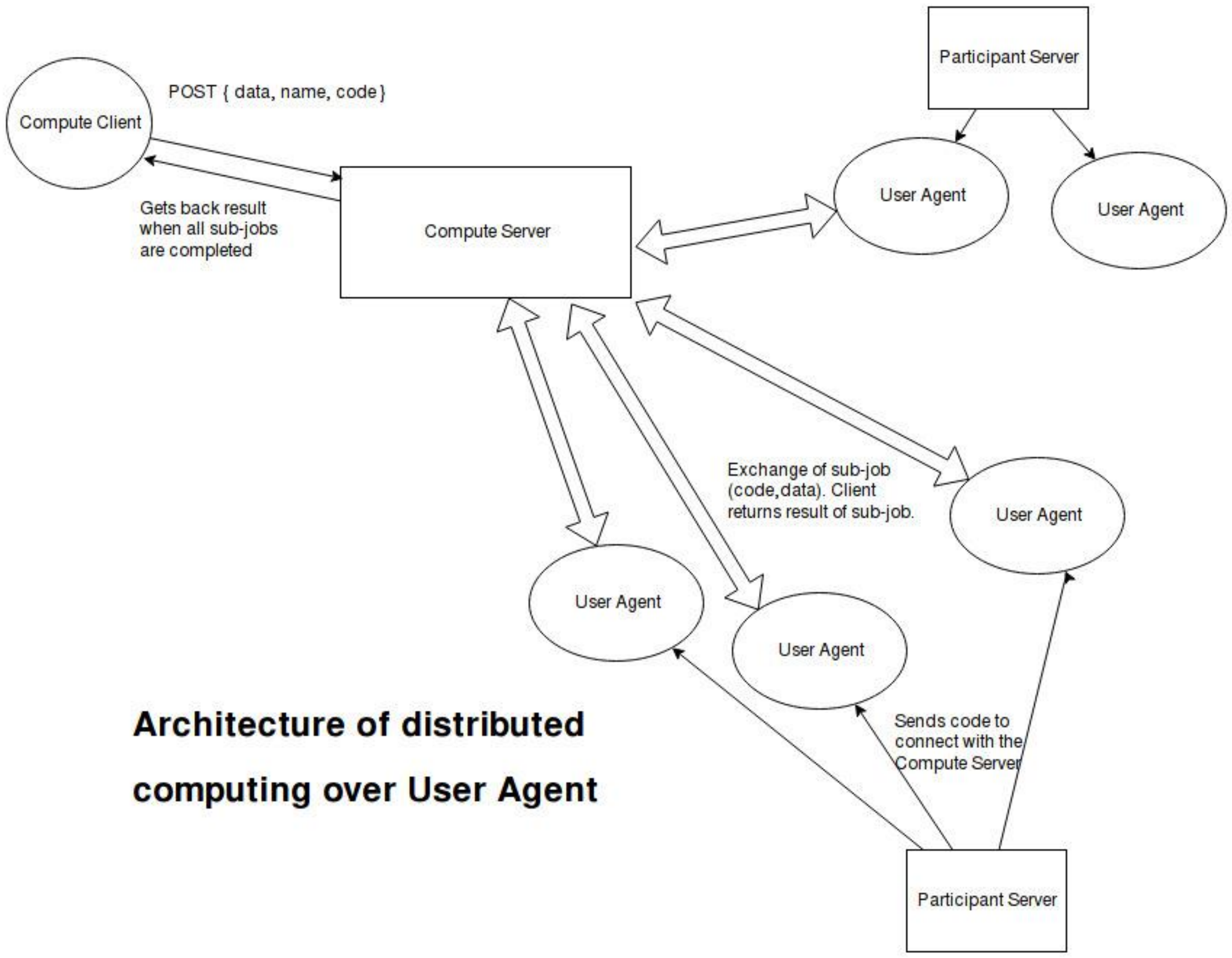

Participant Server
Compute Client
POST { data, name, code }
Compute Server
User Agent
User Agent
Gets back result when all sub-jobs are completed
Exchange of sub-job (code,data). Client returns result of sub-job.
User Agent
User Agent
User Agent
Architecture of distributed computing over User Agent
Sends code to connect with the Compute Server
Participant Server

Fig. 4.2

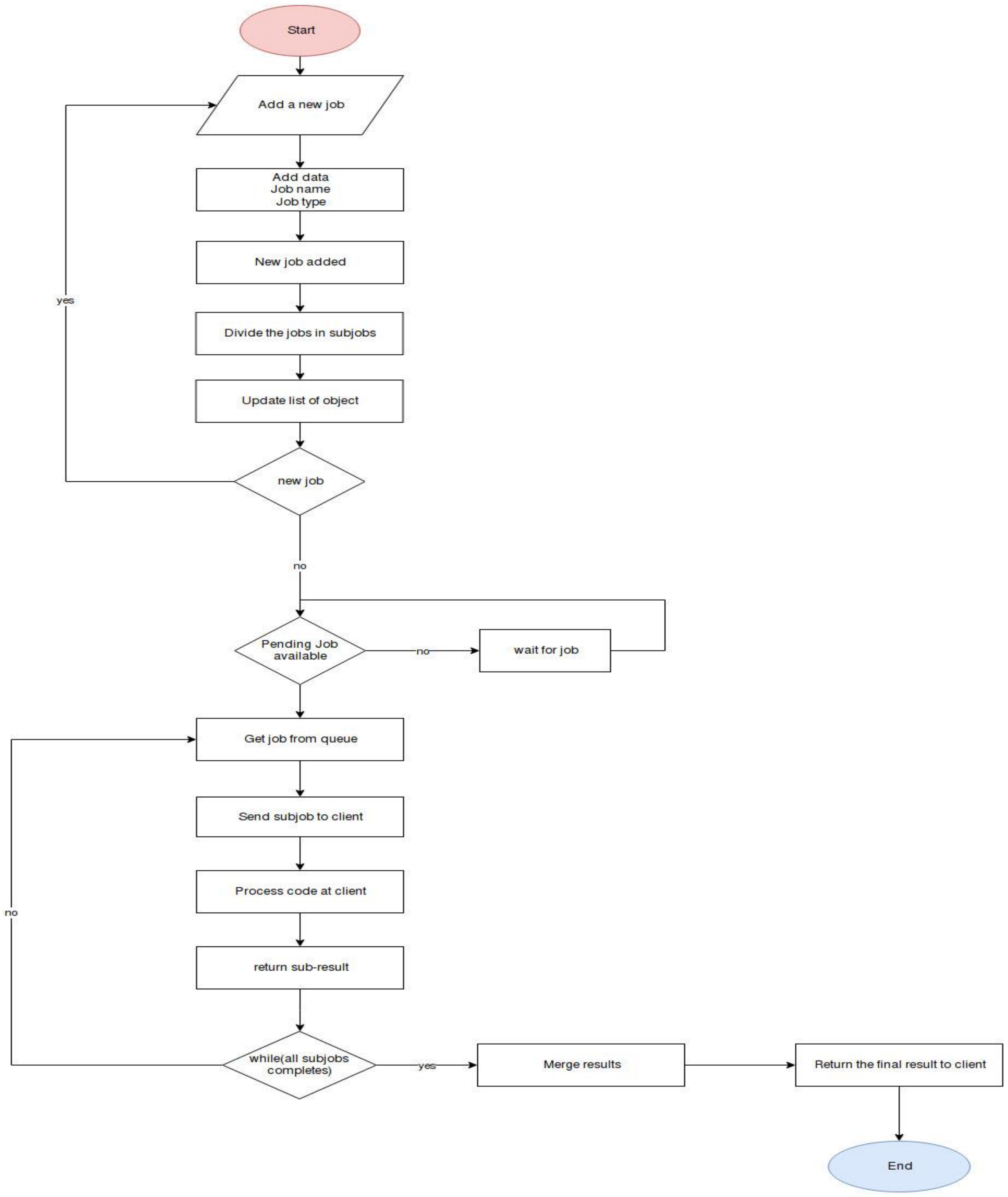


Fig.4.3 Flow chart

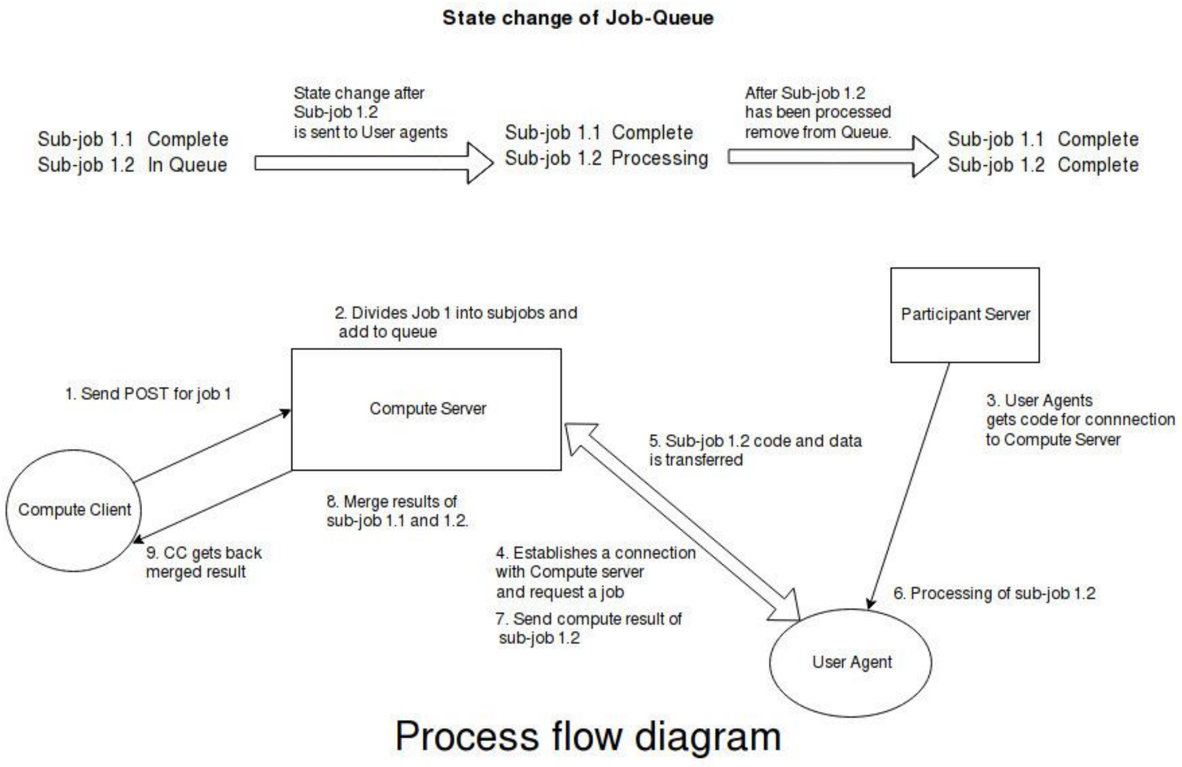

State change of Job-Queue
Sub-job 1.1 Complete
Sub-job 1.2 In Queue
State change after Sub-job 1.2 is sent to User agents
Sub-job 1.1 Complete
Sub-job 1.2 Processing
After Sub-job 1.2 has been processed remove from Queue.
Sub-job 1.1 Complete
Sub-job 1.2 Complete
2. Divides Job 1 into subjobs and add to queue
Participant Server
1. Send POST for job 1
Compute Server
3. User Agents gets code for connnection to Compute Server
5. Sub-job 1.2 code and data is transferred
Compute Client
8. Merge results of sub-job 1.1 and 1.2.
9. CC gets back merged result
4. Establishes a connection with Compute server and request a job
6. Processing of sub-job 1.2
7. Send compute result of sub-job 1.2
User Agent
Process flow diagram

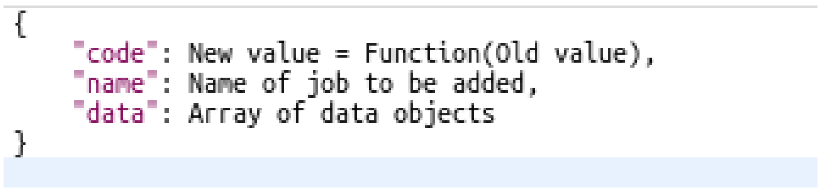
Fig. 4.4

Adding a new computation job

To add a new job to the server, the user can communicate with a RESTful interface. A new job can be added by sending a POST request at the **/job** route of the interface.

REST is an architectural style that defines a set of constraints and properties based on HTTP. Web Services that conform to the REST architectural style, or **RESTful** web services, provide interoperability between computer systems on the Internet. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations. Other kinds of web services, such as SOAP web services, expose their own arbitrary sets of operations.

The format for adding a new job will be as follows.



```
{
    "code": New value = Function(Old value),
    "name": Name of job to be added,
    "data": Array of data objects
}
```

Fig. 4.5

code: This field will contain the array of objects which have been processed by the specified function.
name: This field will contain the name of the job.
data: This field will contain the job

Divide the job into sub-jobs

The job at hand is subdivided into smaller number of jobs called sub jobs. The size of sub jobs can be set in the code module setting the size of job as number of objects in the sub job. The size can vary according to number of clients we have i.e. if we have larger number of machines in to compute the tasks , the relative size of the sub job is smaller and vice-versa.

Send the sub jobs to the client

The created sub jobs are sent to different clients after checking if there is any pending sub job from the previous job . If present then the sub job is extracted from the queue and the results are computed otherwise the clients process the new subjobs at distributed locations.

Merge the results

When all the subjobs are completed and the queue is found to be empty , the results from different clients are collected and then merged as a result of the complete job.

Return the final result to the client

The final merged result is sent back to the client who has posted the job.

CHAPTER-5

RESULTS AND DISCUSSION

5.1 Sample-1

A POST request to **/job** with the following data will add a new job which has the function to eliminate space from a list of names.

Code:

```
        "code": "data_fixed = data.map(function(item,index) { return item.split(' ').join('');});"
```

Fig. 5.1

This code in JavaScript will take the Array of names and eliminates space from them.

Name:

This will be specified in the name field in the JSON.

```
          “name”: “Job to join space separated Names”
```

Fig.5.2

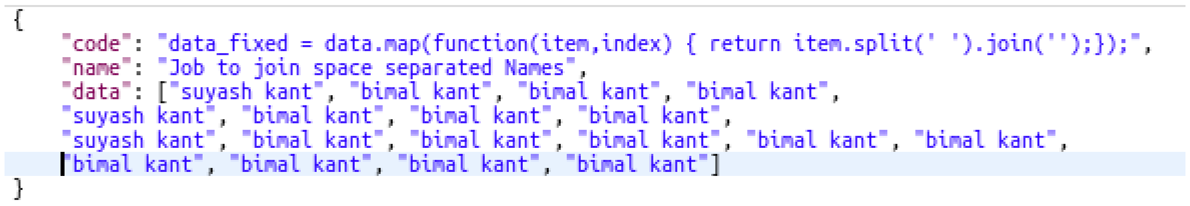
Data:

This will specify the data to which is being processed in the job.

```
        "data": ["suyash kant", "bimal kant", "bimal kant", "bimal kant",
          "suyash kant", "bimal kant", "bimal kant", "bimal kant",
          "suyash kant", "bimal kant", "bimal kant", "bimal kant", "bimal kant", "bimal kant", "bimal
kant", "bimal kant", "bimal kant", "bimal kant"]
```

Fig. 5.3

Combining the above field we get the body of POST request as follows:



```
{
    "code": "data_fixed = data.map(function(item,index) { return item.split(' ').join('');});",
    "name": "Job to join space separated Names",
    "data": ["suyash kant", "bimal kant", "bimal kant", "bimal kant",
    "suyash kant", "bimal kant", "bimal kant", "bimal kant",
    "suyash kant", "bimal kant", "bimal kant", "bimal kant", "bimal kant", "bimal kant",
    "bimal kant", "bimal kant", "bimal kant", "bimal kant"]
}
```

Fig. 5.4

5.2 Sample-2

A POST request to **/job** which can be used to filter the table if their marks is less than 40.

Code:

```
“code”: “data_fixed=data.map(function(item,index) { if(item[1] < 40) { return item[0] } });”
```

Fig. 5.5

This code in JavaScript will take the Arrays of objects and return names of objects which have marks < 40.

Name:

This will be specified in the name field in the JSON.

```
        “name”: “Job to filter objects with marks < 40”
```

Fig. 5.6

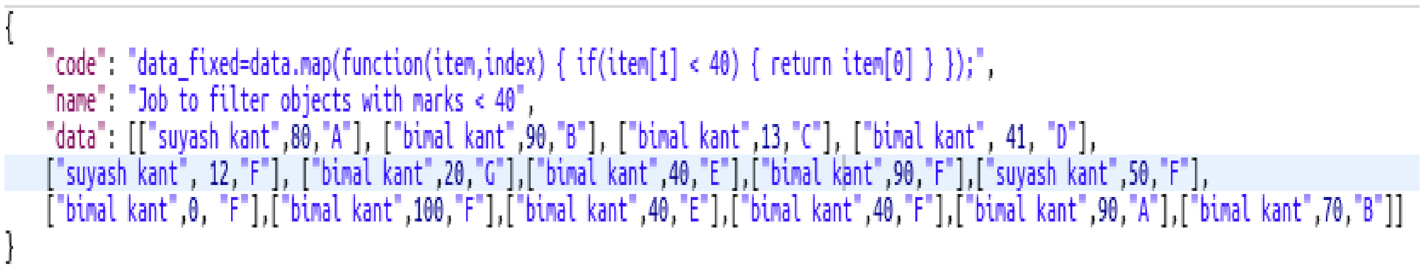
Data:

This will specify the data to which is being processed in the job.

```
        "data": [["suyash kant",80,"A"], ["bimal kant",90,"B"], ["bimal kant",13,"C"], ["bimal kant", 41, "D"],
          ["suyash kant", 12,"F"], ["bimal kant",20,"G"],["bimal kant",40,"E"],["bimal kant",90,"F"],["suyash kant",50,"F"],
          ["bimal kant",0, "F"],["bimal kant",100,"F"],["bimal kant",40,"E"],["bimal kant",40,"F"],["bimal kant",90,"A"],["bimal kant",70,"B"]]
```

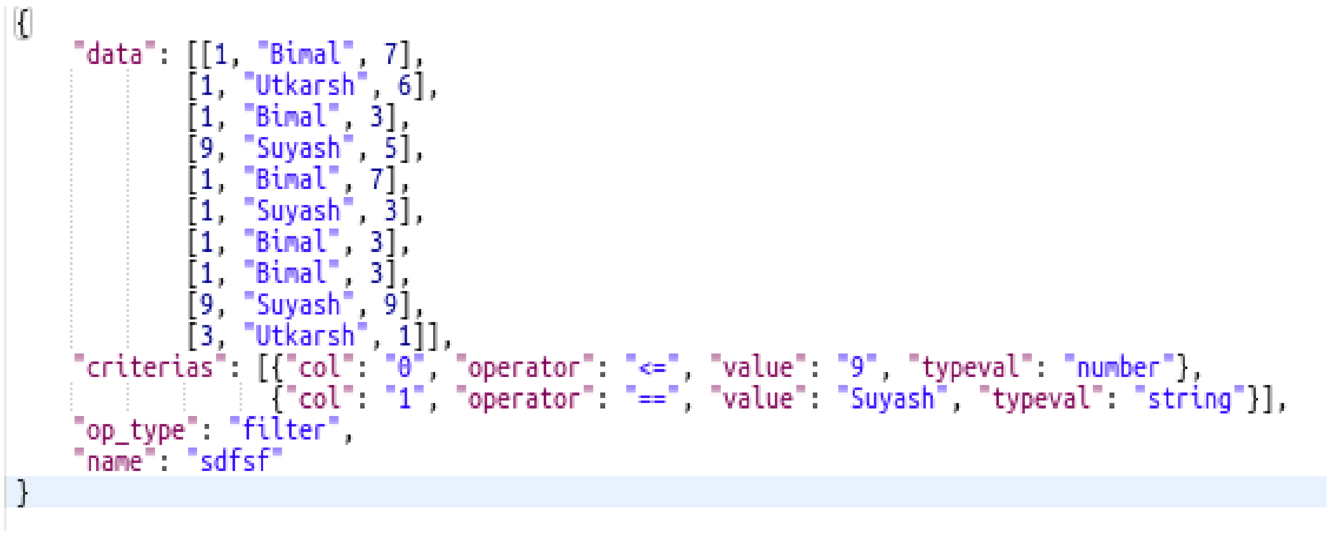
Fig. 5.7

Combining the above field we get the body of POST request as follows:



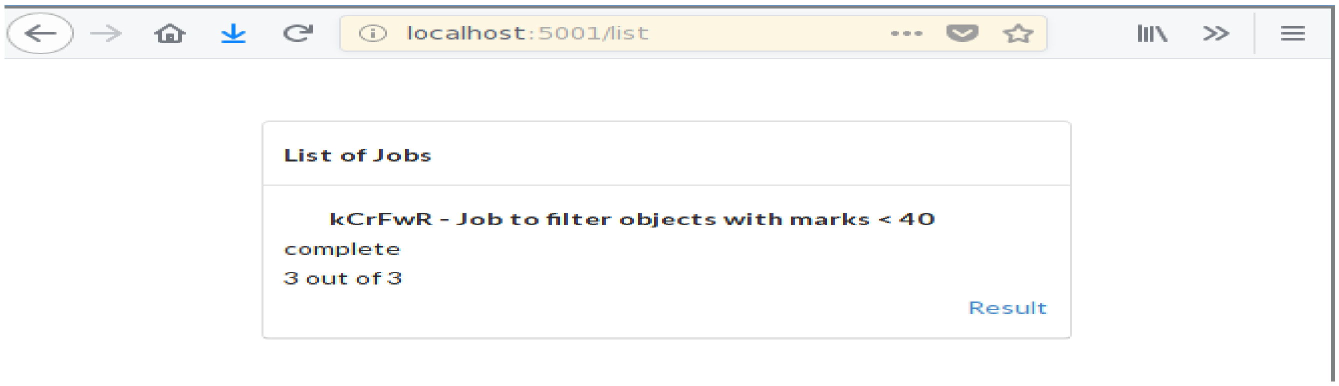
```
{
    "code": "data_fixed=data.map(function(item,index) { if(item[1] < 40) { return item[0] } });",
    "name": "Job to filter objects with marks < 40",
    "data": [["suyash kant",80,"A"], ["bimal kant",90,"B"], ["bimal kant",13,"C"], ["bimal kant", 41, "D"],
    ["suyash kant", 12,"F"], ["bimal kant",20,"G"],["bimal kant",40,"E"],["bimal kant",90,"F"],["suyash kant",50,"F"],
    ["bimal kant",0, "F"],["bimal kant",100,"F"],["bimal kant",40,"E"],["bimal kant",40,"F"],["bimal kant",90,"A"],["bimal kant",70,"B"]]
}
```

Fig. 5.8



```
{
    "data": [[1, "Bimal", 7],
             [1, "Utkarsh", 6],
             [1, "Bimal", 3],
             [9, "Suyash", 5],
             [1, "Bimal", 7],
             [1, "Suyash", 3],
             [1, "Bimal", 3],
             [1, "Bimal", 3],
             [9, "Suyash", 9],
             [3, "Utkarsh", 1]],
    "criterias": [{"col": "0", "operator": "<=", "value": "9", "typeval": "number"},
                  {"col": "1", "operator": "==", "value": "Suyash", "typeval": "string"}],
    "op_type": "filter",
    "name": "sdfsf"
}
```

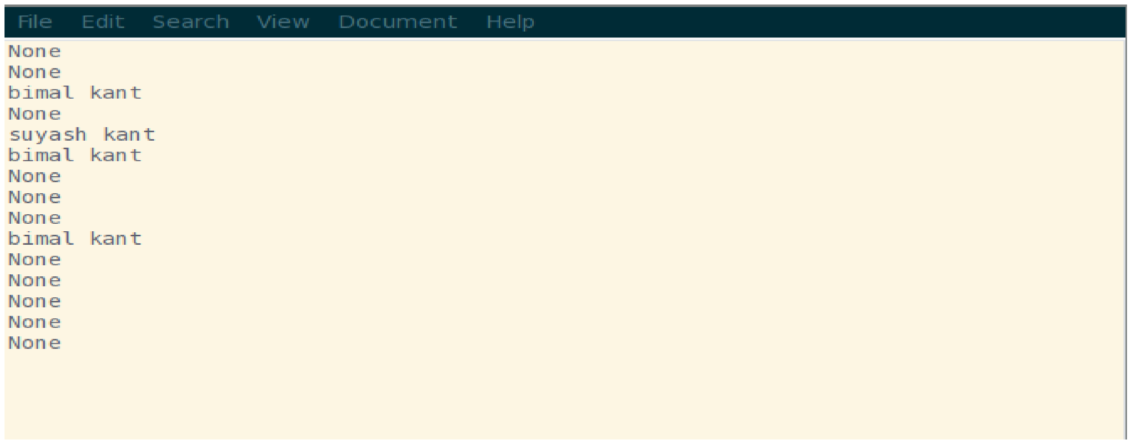
Fig. 5.9



List of Jobs

kCrFwR - Job to filter objects with marks < 40
complete
3 out of 3

Fig. 5.10



```
None
None
bimal kant
None
suyash kant
bimal kant
None
None
None
bimal kant
None
None
None
None
None
```

Fig. 5.11

5.3 Sample - 3

A POST request to **/job** which can be used to filter records with city = Kanpur.

Code:

```
"code": "data_fixed= data.map(function(item,index) { if(item[1] === \"Kanpur\") { return {\"name\": item[0] , \"phone number\": item[2] } } });",
```

Fig. 5.12

This code in JavaScript will take the Arrays of records and return names of people with city = Kanpur.

Name:

This will be specified in the name field in the JSON.

```
"name": "job to filter records with name Kanpur",
```

Fig. 5.13

Data:

This will specify the data to which is being processed in the job.

"data": [["suyash kant","Kanpur","99999191991"], ["bimal kant","Dehradun","9299239292"], ["bimal kant","Kanpur","9282929922"],

["bimal kant", "Dehradun", "9282929922"], ["suyash kant", "Surat","9282929922"], ["bimal kant","Lucknow","9282929922"],

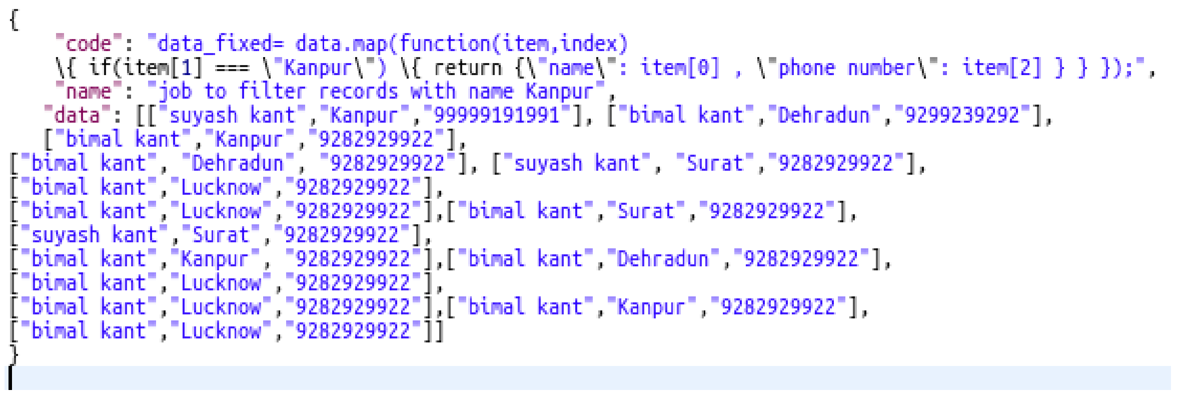
["bimal kant","Lucknow","9282929922"],["bimal kant","Surat","9282929922"],["suyash kant","Surat","9282929922"],

["bimal kant","Kanpur", "9282929922"],["bimal kant","Dehradun","9282929922"],["bimal kant","Lucknow","9282929922"],

["bimal kant","Lucknow","9282929922"],["bimal kant","Kanpur","9282929922"],["bimal kant","Lucknow","9282929922"]]

Fig. 5.14

Combining the above field we get the body of POST request as follows:

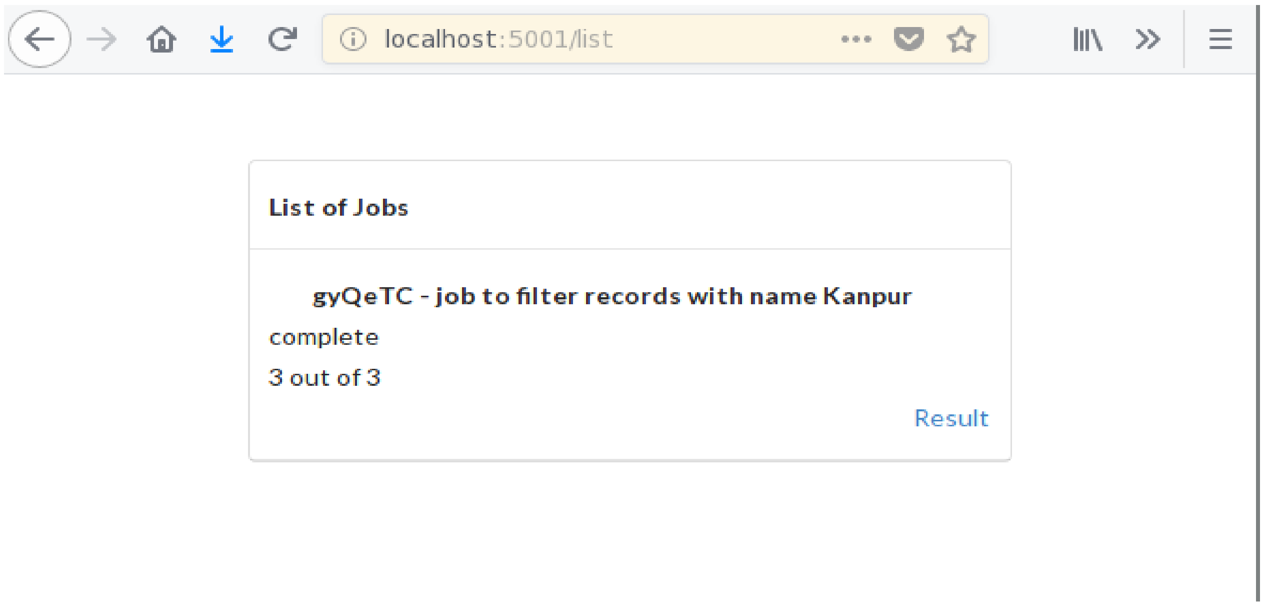


```
{
    "code": "data_fixed= data.map(function(item,index)
    \{ if(item[1] === \"Kanpur\") \{ return {\"name\": item[0] , \"phone number\": item[2] } } });",
    "name": "job to filter records with name Kanpur",
   "data": [["suyash kant","Kanpur","99999191991"], ["bimal kant","Dehradun","9299239292"],
   ["bimal kant","Kanpur","9282929922"],
["bimal kant", "Dehradun", "9282929922"], ["suyash kant", "Surat","9282929922"],
["bimal kant","Lucknow","9282929922"],
["bimal kant","Lucknow","9282929922"],["bimal kant","Surat","9282929922"],
["suyash kant","Surat","9282929922"],
["bimal kant","Kanpur", "9282929922"],["bimal kant","Dehradun","9282929922"],
["bimal kant","Lucknow","9282929922"],
["bimal kant","Lucknow","9282929922"],["bimal kant","Kanpur","9282929922"],
["bimal kant","Lucknow","9282929922"]]
}
```

Fig. 5.15

Output:

After the sub-jobs have been executed and merged we get following result:



List of Jobs

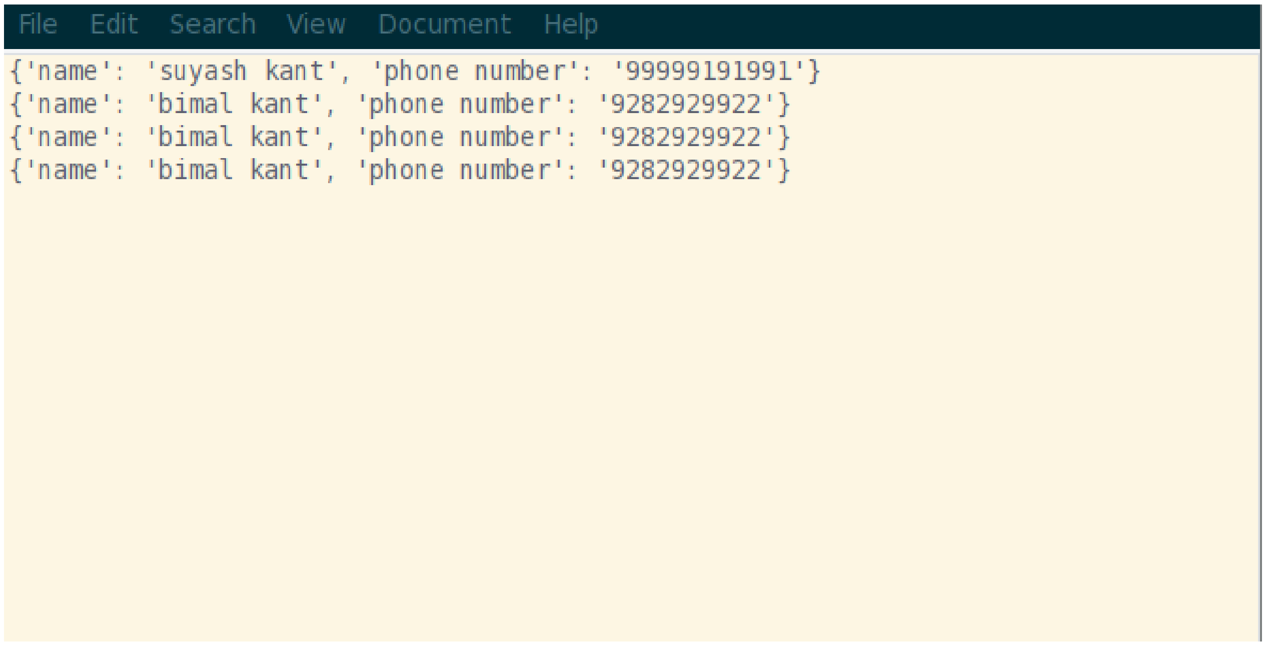
gyQeTC - job to filter records with name Kanpur

complete

3 out of 3

Result

Fig. 5.16



```
{'name': 'suyash kant', 'phone number': '99999191991'}
{'name': 'bimal kant', 'phone number': '9282929922'}
{'name': 'bimal kant', 'phone number': '9282929922'}
{'name': 'bimal kant', 'phone number': '9282929922'}
```

Fig. 5.17

CHAPTER - 6

CHALLENGES

Not everyone wants to allow their system to be used for processing.

Notify people clearly that some task is running in the background.
SOLUTION: Allow them to opt out from having their systems do such background processing.

Security problem: data being sent can be modified.

Use some form encryption. Use hash value to check that data being processed is the same.

If a sub-job fails to execute for some reason then other jobs will wait forever

SOLUTION: Assign a timer to every sub-job. If the result of a sub-job cannot be computed within that time-frame then move it to the bottom of the Job Queue so other jobs can be executed.

CHAPTER-7

CONCLUSION AND FUTURE WORK

The system presented in this project is one of the first study about the realization of a new solution of distributed computing using web as infrastructure. Final results of this work have showed that such a system can be realized with good performance and a simple architecture, using well known technologies and softwares, standard protocols and specifications, to obtain a worldwide installation free client of distributed computing, that is the main goal of the project.

Some aspects have still to be analyzed. For example an important issue concerns the security related to the code executed by the client and to the results received by the server: on the client side we must assure that the code executed does not cause malfunctions, data loss or other problems (thinking about the system in terms of offered service as third part); on the server side we have to pay attention to the data returned by the computation, because of the client can potentially send malicious or wrong results to the server. Future activities will be devoted in resolving these issues to obtain a robust system, general in terms of architecture and powerful in terms of performance.

REFERENCES

[1] https://en.wikipedia.org/wiki/Global_Internet_usage

[2] https://w3techs.com/technologies/details/cp-javascript/all/all

[3] http://www.speedtest.net/global-index

[4] Distributed Computing Through Web Browser F.Boldrin, C. Taddia, G. Mazzini Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th