Improving the Reliability of Web Based Result Query Systems during High Traffic Periods

J Shiny Duela, M Roshni Thanka, Bijolin Edwin

Abstract: Displaying of examination results by a single central entity, for lakhs of students becomes a tedious task, and sometimes may also result in server crashing. These servers typically rely on heavy and often unrestricted threads spawned to handle each incoming request which is the reason why the server resources are used up quickly. We propose a solution that is three fold: First, multiple Volunteer entities are brought in to hold the data and donate a portion of their computing power to offload the enormous work placed on the central entity. Second, the central entity is changed to play the role of dispatcher that generates monitors and assigns extremely lightweight, independent processes (called agents) to each user request without requiring any additional hardware upgrade. Each agent will be responsible to satisfy their assigned user requests. Third, we introduce a load balancing technique derived from the ideas of autonomous agents load balancing techniques in cloud to provide load balancing among the Volunteer entities and the central entity such that the Volunteer entities can continue with its own tasks and not be overwhelmed by its Volunteer work while ensuring fast response time and better reliability and response to the user.

Index Terms: ***servers, concurrency, threading, actors, event-driven***

I. INTRODUCTION

The Internet has existed for decades but it is still growing stronger every day. More people are coming online and accessing millions and millions of pages and across thousands and thousands of servers, transferring enormous amounts of data every day across the vast network of cables that entwine the world. The increasing connections mean servers have to learn to deal with increasing traffic. For the most part, the progress in hardware power has kept up to meet the increasing demands. However, it may not be always possible for the servers to be upgraded to the latest hardware. Solutions must also be investigated on the software side.

One of the problems public institutions in our country face today is properly displaying exam results. The current scenario of looking at exam results of public institutions is dismal. It becomes difficult for both the candidates as well as the server side where they may have to deal with several server crashing problems and high traffic over the network. Also the candidates may have to wait for too long for the results to load.

The world of cloud computing has seen a lot of research effort put in to find increasingly efficient ways of managing and running the cloud and providing the cloud services. An important aspect of the working of cloud is load balancing. This topic has seen a lot of work put into it [1][5] and many different techniques have been put forward to address it. These ideas about load balancing researched for cloud applications have not seen wide use for ordinary servers and server clusters outside of the cloud environment. Being able to handle enormous traffic is an important property for any web application and it may be possible to obtain great performance benefits by bringing over the ideas from the cloud to other applications outside the cloud.

In addition to load balancing techniques, further improvements may be obtained by the use of the Actor model [9] for better concurrency and through the improved efficiency and resource utilization provided by the use of lightweight processes [10] over threads that tend to be used in normal servers.

Revised Manuscript Received on July 22, 2019.

Dr.J.Shiny Duela, Department of CSE,Jerusalem College of Engineering,Chennai,India
Dr.M.Roshni Thanka, Department of CSE, Karunya University, Coimbatore,India
Dr.Bijolin Edwin, Department of CSE, Karunya University, Coimbatore,India

II. RELATED WORK

This section evaluates some of the previous research done towards progress in load balancing, fault tolerance and handling scale in web systems.

Singh et al. [1] proposes an autonomous agent based load balancing algorithm (A2LB) to address utilization, throughput, response time, scheduling and scalability. Whenever a VM becomes overloaded, the service provider has to distribute the resources in such a manner that the available resources will be utilized in a proper manner and load at all the virtual machines will remain balanced. A2LB mechanism comprises of three agents: Load agent, Channel Agent and Migration Agent. Load and channel agents are static agents whereas migration agent is a special category of mobile agents.

Lightner et al. [2] introduces the idea of providing fault tolerance in distributed server clusters through the use of supervisors that can monitor operations and restart them if they crash. It maintains node managers that keep track of the running operations and automatically troubleshoot them in case of any failures.

Chechina et al. [3] presents a systematic evaluation of Erlang. It presents a comprehensive evaluation of the scalability and reliability of Erlang using three typical benchmarks and a case study. It investigates the scalability limits of distributed Erlang for engineering reliable systems, identifying network connectivity and the maintenance of global recovery information as the key bottlenecks.

Vaughn [4] shows how to translate the longtime promises of Actor model into practical reality. It introduces the tenets of reactive software, and shows how the message-driven Actor model addresses all of them–making it possible to build systems that are more responsive, resilient, and elastic and build robust systems that are simpler and far more successful.

Dobale et al. [5] speaks about the benefits of maintaining whole files in a node rather than splitting them across several nodes into chunks. Here a Load Balance Nearest Search Algorithm is presented to cope with the load imbalance problem. Load is transferred from heavily loaded node to physically closed lightly loaded node. The cost in memory is balanced by the savings obtained in avoiding complex hashing calculations to find the correct location of a file chunk.

Ariharan et al. [6] proposes a load balancing algorithm that focuses on reducing wait time and evenly distributing the load. It introduces neighbour awareness and prediction mechanisms to further improve the selection process of nodes for random walk. The proposed algorithm selects the least loaded node from the neighbour list for the random walk. It finds selecting least loaded node for random walk performs well than probability based selection or the improved random walk algorithm.

Rajan et al. [7] brings forth a model called capsule-oriented programming, where programmers describe a system in terms of its modular structure and write sequential code to implement the operations of those modules. It shows how to fuse the notion of a module with the notion of an activity and an ownership domain and provide simultaneous benefits in terms of both software evolution and program performance.

Hao, in his work [8], provides an overview of the Open Telephony Protocol (OTP), a set of battle tested design patterns for building concurrent and highly scalable systems utilizing the power of actors. It introduces the concepts through the lens of Elixir, a programming language that runs on the Erlang VM and how it can be applied for big benefits in the web ecosystem.

The literature exemplifies the progress made in the area of load balancing in cloud computing. These improved techniques have not seen extensive application in distributed web servers. This is an area of application that is still in research. Some of the literature describes the ideas that can be applied for improving efficiency of the cloud load balancing techniques when applying them to web systems.

III. EXISTING METHODOLOGY

The current industry standard for web facing servers, such as exam result query systems are Apache and its like, servers that depend on heavyweight threads and require a lot of expensive resources to manage thousands of visitors. This works well in most cases but does not deal well with large number of concurrent users. These systems are not as reliable as it needs to be and requires great expense on powerful hardware in order to bring it up to the required level.

The meteoric rise of cloud computing has lead to a lot of research on improving load balancing in the cloud. The ideas of autonomous agents have been applied extensively in cloud computing systems for better load balancing, which involves sending out these autonomous agents to do the work of discovering the resources and using them to keep track of underloading and overloading of the resources.

Singh et al. [1] proposed the ***Autonomous Agents Load Balancing algorithm (A2LB)*** that capitalizes on these ideas. It makes use of three kinds of agents: *Load Agent, Channel Agent, Migration Agent.* In Fig1.1 The Load Agent keeps track of VM utilization in different data centres. The Channel Agent receives request for additional resources and sends out the Migration agents. The Migration Agents travel to data centres and query the Load agents in order to find the free resources. This information is communicated back to the Channel agent. The Migration agent is a self contained program that will run itself in the data centres it travels to in order to query it. The Channel agent keeps track of the Migration agents it sends out and the status of the VMs as given by the Migration agents.

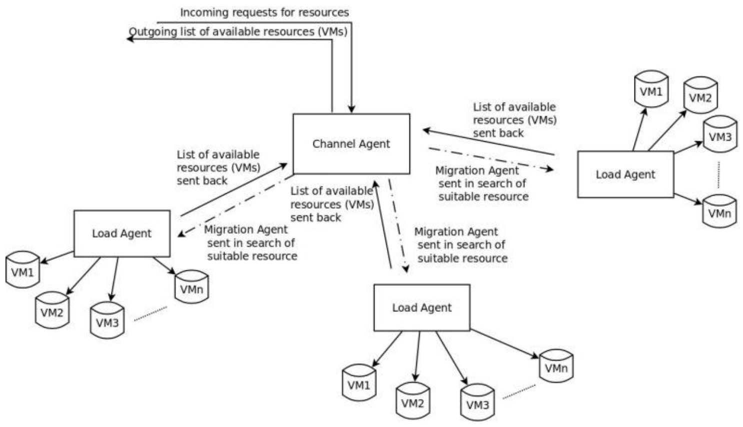

Incoming requests for resources
Outgoing list of available resources (VMs)
Channel Agent
List of available resources (VMs) sent back
List of available resources (VMs) sent back
List of available resources (VMs) sent back
Migration Agent sent in search of suitable resource
Migration Agent sent in search of suitable resource
Migration Agent sent in search of suitable resource
Load Agent
Load Agent
Load Agent
VM1
VM2
VM3
VMn

Figure1.1: Architecture of A2LB

Utilizing autonomous agents for load balancing in web application servers is not a topic that has been extensively studied. By utilizing these ideas of autonomous agent load balancing, it is possible to provide good response time and high throughput in non cloud based web applications, especially when trying to utilize servers that cannot be dedicated fully to the application's needs.

IV. PROPOSED ARCHITECTURE

Our goal is to develop a reliable web based result query system that can work well based on the concepts of centralized load balancing, autonomous agents, the actor model with the use of Volunteer servers. Load balancing is done by improving on and utilizing the ideas of autonomous agents described in [1] and has been derived from cloud computing technologies to provide a better solution .

a. Conceptual framework

The Figure1.2 gives ideas about load balancing described in the literature were targeted for cloud applications and has not seen wide use for ordinary servers and server clusters outside of the cloud environment. Being able to handle enormous traffic is an important property for any web application and it may be possible to obtain great performance benefits by bringing over the ideas from the cloud to other applications outside the cloud. Result query systems, as shown in figure, for examination results are usually run on ordinary servers maintained by the organization responsible for the examination. These servers have high performance requirements but frequently suffer from problems.

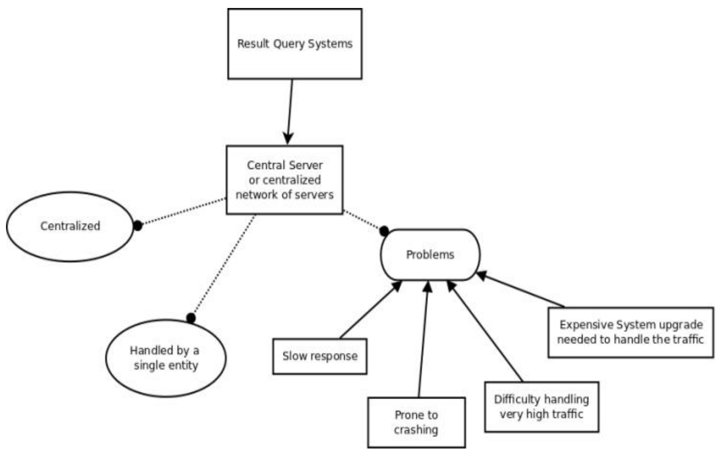

Result Query Systems
Central Server or centralized network of servers
Centralized
Problems
Handled by a single entity
Slow response
Expensive System upgrade needed to handle the traffic
Prone to crashing
Difficulty handling very high traffic

Figure1.2: Properties of a typical result query system

Introducing volunteers, as shown in Fig. 1.3, to donate some of their infrastructure may solve some problems but not enough.

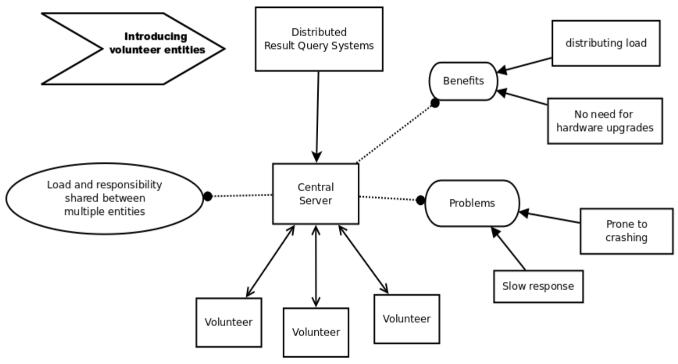

Introducing volunteer entities
Distributed Result Query Systems
distributing load
Benefits
No need for hardware upgrades
Load and responsibility shared between multiple entities
Central Server
Problems
Prone to crashing
Slow response
Volunteer
Volunteer
Volunteer

Figure1.3: Introducing volunteer entities to the system

In Fig1.4,the goal is to try and solve all the current problems faced by such result query systems by borrowing the ideas of load balancing from cloud computing paradigm with adaptations to suit the given use case, as shown in figure.

b. Architecture overview

The proposed solution consists of the use of a central server that is responsible for receiving the user requests and communicates with multiple Volunteer servers that hold the user's data. It makes use of lightweight, independent, isolated processes to handle the user requests and communication and is able to handle more traffic due to the lightweight nature. The central server is responsible only for managing the communication and is (mostly) freed from the responsibility of managing the user data.

The central server is capable of generating agents for each user, which are capable of sending messages and knows how to respond to the next message received. The agents can make any local decisions in response to a message that it receives. The function of agents is to interact with the user and perform tasks on their behalf. They also interact with the other agents to collaborate and delegate tasks. They usually reside on a distributed environment and interact with non-agent computer systems.

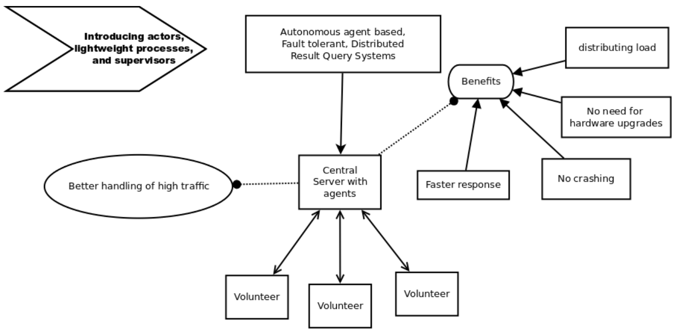

Introducing actors, lightweight processes, and supervisors
Autonomous agent based, Fault tolerant, Distributed Result Query Systems
distributing load
Benefits
No need for hardware upgrades
Better handling of high traffic
Central Server with agents
Faster response
No crashing
Volunteer
Volunteer
Volunteer

Figure1.4: Introducing agents and fault tolerance

Each user's agent selects a Volunteer server to retrieve the result information from. The selection is done by consulting a Volunteer status agent to identify which Volunteer server is known to be free. If a user's agent tries and fails to communicate with the Volunteer server, it will inform the volunteer status table of the kind of failure that occurred before trying again with a different Volunteer server.

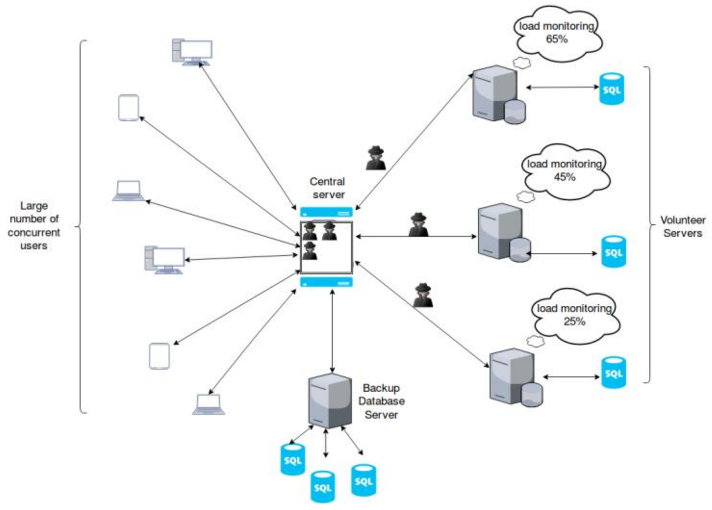


Figure1.5 Architecture Overview

The Volunteer server is not required to fully dedicate all its resources for the requests coming from the central server. It can set a limit on how much resources can be used. If the limit is exceeded, the Volunteer server can reject requests.

Hence we propose a solution that provides better reliability and fault tolerance to the overall system by distributing the load thereby reducing the problem of server crashing and incredible high traffic over the network, thus enabling the candidates to view their results at a much faster time and the results produced are authentic.

Components of the system:

The proposed system can be described in terms of four main modules.Fig1.6 specifies the module of the system. The **Central Server** is the front-end that receives the incoming user requests. It generates and supervises the **Agents** for each incoming request. It also creates and supervises the **Server Status Table.** The supervision is carried out via a *supervision tree* [2], a construct by which Central Server can monitor the Agents and Server Status Table and restart them if they crash.It makes sure no single Agent monopolizes the resources for too long by applying *preemptive scheduling* [11]. It is also responsible for reclaiming the resources used to be reallocated once the agent is done with its task.

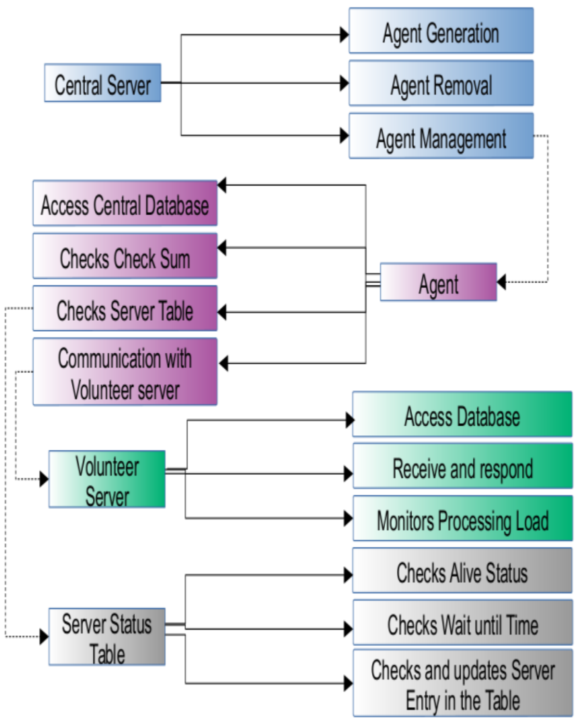


Figure1.6. Modules of the system

The **Agent** is a lightweight, independent, isolated process spawned by the Central Server to handle the user request. It identifies the available Volunteer Server from the Server Status Table and forwards the request to it and awaits the response. If no response takes too long or it is a reject, it will update the Server Status table accordingly and try a different Volunteer Server. If proper response is obtained, it will return a response to the user with the data received from the Volunteer Server.

The **Volunteer Server** receives the request from the Agent and will first check if its resource usage has not exceeded the set limit. If it is exceeded, it will return a reject response along with a *time to wait,* which specifies how long any Agent should wait before contacting the Volunteer Server again. If the resource usage is within limits, then the Volunteer Server will perform the requested task and return the results back to the waiting Agent. The Volunteer Servers are completely independent of each other.

The **Server Status Table** is used to hold information about the current known state of the Volunteer Servers. The Table is initially empty and entries are inserted as the Agents communicate with Volunteer Servers and gets updates about their current situations. The Agents will look at the Server Status Table entry before attempting to contact the Volunteer Server to make sure it is able to receive the request from the Agent. The Agent will also request the Server Status Table to update an entry corresponding to a Volunteer Server if the Agent receives a reject from the Volunteer Server or if the Volunteer Server times out. The updates are asynchronous, meaning that the Agent will simply carry on with its work after sending an update to the Server Status Table. This increases efficiency of the Agent as it is not spending time waiting on the Server Status Table. The table1.1 shows the structure of the Server Status Table.

Volunteer Server IP	Wait until (unix time seconds)	Alive?
123.146.24.55:443	1515346780	True

Table1.1: structure of Server Status Table

From Table1.1 If the Agent receives a reject from a Volunteer Server, it will update *Wait until* column of the Server Status Table with the *time to wait* value returned by the Volunteer Server in that Volunteer Server's table entry. If the request from the Agent takes too long i.e. the Volunteer Server is not responding, the Agent will update the *Alive?* column of the Volunteer Server's entry to *False* in the Server Status Table. Algorithms describing how the modules work together are given below.

Algorithms in Central Server

```
Central_server( )
Input:Request from user;

Initialize Supervision_Tree() process;
Initialize Volunteer_Server_Status_Table and place it in Supervision_Tree();
Initialize backup database connection and place it in Supervision_Tree();
for each incoming user request
{
Generate Agent() that will process the request;
Deallocate Agent() when its work is done and free the resources;
}
```

```
Supervision_Tree()
Input: worker process
Maintains and manages a collection of worker processes;
Monitors the health of the worker process;
Reclaims resources when the worker process has completed its task;
If required, restart the worker process if it crashes;
```

Supervision_Tree() is an OTP pattern [8] that is used to supervise worker processes [2] assigned to it and forms an essential part of the ***Central_Server()***'s working.

Algorithm for Agent

```
Agent()
Input: request data from user, list of ip addresses of the Volunteer Servers.
Output:HTML page response
```

```
Select ip address of Volunteer_Server() from list;
Check Volunteer_Server() availability and send request to it;
Update the Volunteer_Server_Status_Table according to the response;
If maximum attempts exceeded, retrieve data from backup database;
```

The Volunteer_Server_Status_Table is responsible for maintaining information about the Volunteer Servers, including availability and wait times for contacting the Server. The Agent's behavior is inspired by the working of *Migration Agents* and *Channel Agents*[1].

Algorithm for Volunteer Server

Volunteer_server()
Input: *student register number and date*
Output: *corresponding student data*

```
Retrieve CPU load, Retrieve memory load;
Determine system load from retrieved load information;
If system load within predefined limit
{
if (register number and date present in database)
Retrieve and return data of the student from the database;
else
    send error message;
}
else
Generate time out interval according to current load;
return time out interval;
```

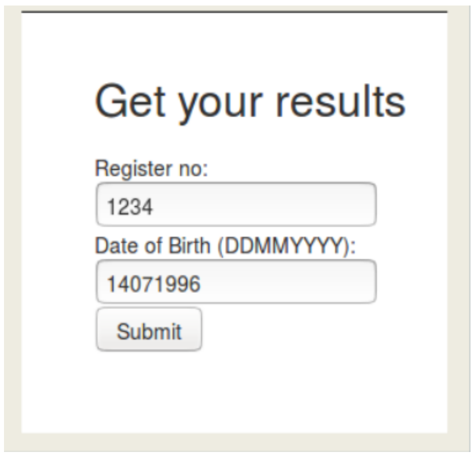
V. IMPLEMENTATION AND ANALYSIS

We make use of two major tools know as Erlang and Elixir. Erlang is a general purpose, concurrent, functional programming language which has major properties of immutable data, and pattern matching. Erlang runs on the BEAM virtual machine (VM) that is optimized for maximum utilization of multicores for greatest performance impact. It leverages its own scheduler and utilizes the power of lightweight processes over threads which allows for greater number of simultaneous operations than could have been provided by resource heavy threads. Elixir is a language based on Erlang that runs on the BEAM VM and carries over the benefits and features of Erlang with a more modern syntax.

We use Elixir in Fig1.7 for building the central server due to its ability to handle high number of requests even given limited resources by leveraging the power of lightweight processes provided by the BEAM VM. The analysis below is based on running the central server on a single Raspberry Pi 3, demonstrating its ability to provide high performance even with limited resources.

The volunteer servers are built with Python due to its simplicity and compatibility with a wide range of hardware and libraries for gathering CPU and memory information, necessary for load monitoring. The analysis was carried out with 5 Volunteer Servers running on the same Raspberry Pi among which only 2 are configured to reply, 2 always time out, and 1 always sends a 'reject' signal back to the central server.

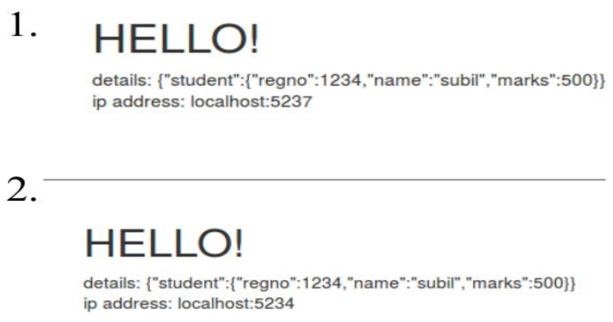
The fig1.8 shows the login page where the user enters their details. The central server will select an available volunteer server and return a result page, as shown in the figure, using the data returned from the volunteer server.


Get your results

Register no: 1234

Date of Birth (DDMMYYYY): 14071996

Figure1.7: Login page


1.

HELLO!

details: {"student":{"regno":1234,"name":"subil","marks":500}}
ip address: localhost:5237

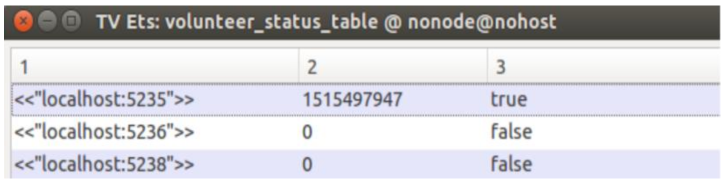
2.

HELLO!

details: {"student":{"regno":1234,"name":"subil","marks":500}}
ip address: localhost:5234

Figure1.8: Response formulated from data obtained from different Volunteer Servers

The volunteer status table in Fig1.9 maintains the details about the non working volunteer servers, successfully preventing the agents from trying them unnecessarily and thus save resources.


TV Ets: volunteer_status_table @ nonode@nohost

1	2	3
<<"localhost:5235">>	1515497947	true
<<"localhost:5236">>	0	false
<<"localhost:5238">>	0	false

Figure1.9: Volunteer Server Status Table with the non working server information

Name	Highest Rate of Requests	Mean Rate of Requests	Count
connections	54.1 / sec	26.14 / sec	6505
requests	72.7 / sec	39.80 / sec	9413

Table 1.2: load test results with the proposed system

Name	Highest Rate	Mean Rate	Count
connect	231.8 / sec	106.03 / sec	19551
requests	332.2 / sec	155.16 / sec	28322

Table 1.3: load test results with a PHP-MySQL centralized server setup

Load tests were applied in Table1.2 and Table1.3 which show that around 19,000 TCP connections were established in with a mean rate of 106 connections per second. The load tests also show that with around 28,000 requests over a 2 minute period, the central server is able to field on average around 155 requests per second, with a highest rate of around 332 requests per second. This is impressive considering the less powerful hardware of the Raspberry Pi. The same load tests when applied to a standard PHP-MySQL centralized server setup under the same conditions show result in much worse performance compared to the proposed system.

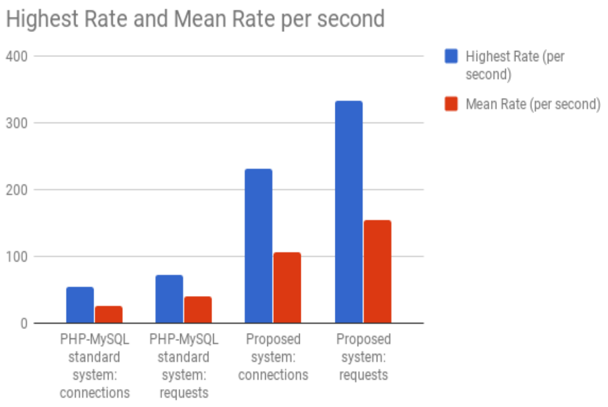


Figure1.10 A comparison of two test results

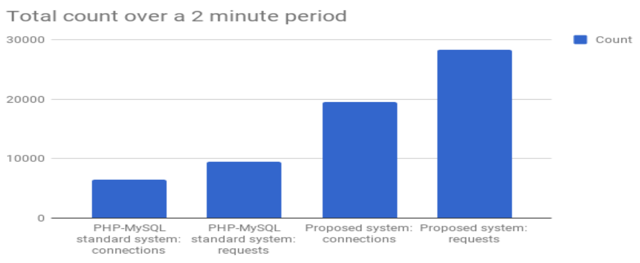


Figure1.11 bar graph for the test results

A visual comparison of the test results of the two systems is shown in the bar graph of Fig1.10

The next graphs Fig1.12 and Fig1.13 shows that the proposed system was able to field around 1300 simultaneous users attacking the server without crashing and was able to maintain around 800 simultaneous connections.

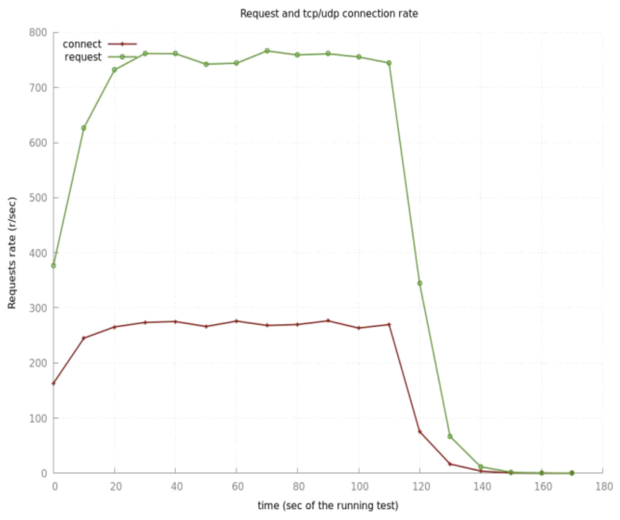


Figure1.12: Connections and requests throughput

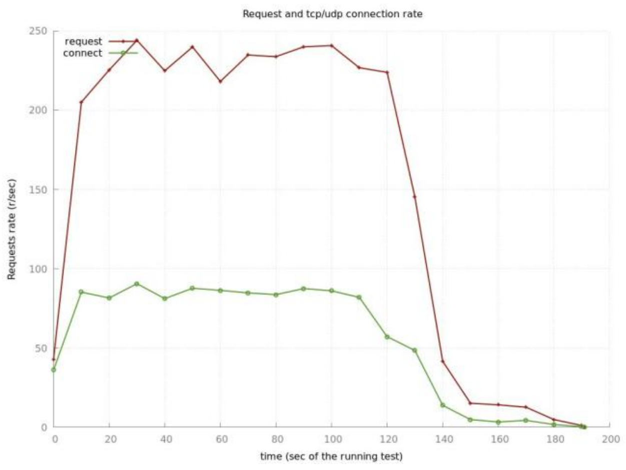


Figure1.13 Results of the proposed system

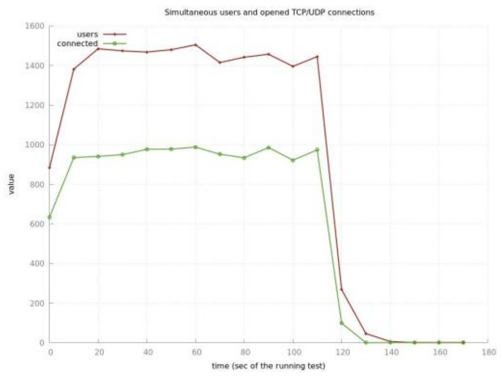


Figure1.14: New incoming users over test period

VI. CONCLUSION

This work contributes an improvement in the existing situation of slow, unresponsive result query systems through the application of autonomous agent based load balancing techniques derived from the techniques applied in cloud systems. The proposed system is able to provide far better performance and is able serve more users simultaneously than the existing, conventional systems. It does it by leveraging the available resources obtained from volunteers who have a stake in the service the result query systems provide, while avoiding overloading any of the volunteers. In the future, it may be possible to expand the application of this system to more general web systems in addition to the specific use case of result query systems.

REFERENCES

1. Singh, Aarti, Dimple Juneja, and Manisha Malhotra. "Autonomous agent based load balancing algorithm in cloud computing." Procedia Computer Science 45 (2015): 832-841.
2. Lightner, Scott, and Franz Weckesser. "Fault tolerant architecture for distributed computing systems." U.S. Patent No. 9,201,744. 1 Dec. 2015.
3. Chechina, Natalia, et al. "Evaluating scalable distributed Erlang for scalability and reliability." IEEE Transactions on Parallel and Distributed Systems (2017).
4. Vernon, Vaughn. Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka. Addison-Wesley Professional, 2015.
5. Dobale, Ms Radha G., and R. P. Sonar. "Load balancing in cloud." International Journal of Engineering Research and General Science 3.3 (2015): 160-167.
6. Ariharan, V., and Sheeja S. Manakattu. "Neighbour Aware Random Sampling (NARS) algorithm for load balancing in Cloud computing." Electrical, 7. Computer and Communication Technologies (ICECCT), 2015 IEEE International Conference on. IEEE, 2015.
7. Rajan, Hridesh. "Capsule-oriented programming." Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, 2015.
8. Hao, Benjamin Tan Wei. "The Little Elixir & OTP Guidebook." (2016).
9. Agha, Gul A. Actors: A model of concurrent computation in distributed systems. No. AI-TR-844. MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, 1985.
10. Ritson, Carl G., Adam T. Sampson, and Frederick RM Barnes. "Multicore scheduling for lightweight communicating processes." Science of Computer Programming 77.6 (2012): 727-740.
11. Leung, Joseph Y-T., and M. L. Merrill. "A note on preemptive scheduling of periodic, real-time tasks."*Information processing letters* 11.3 (1980): 115-118.

AUTHORS PROFILE

Dr. J. Shiny Duela obtained her B.E., and M.E., degree in Computer Science and Engineering discipline and Ph.D in Information and Communication Engineering from Anna University,Chennai. She is presently working as Assistant Professor, Department of Computer Science and Engineering, at Jerusalem College of Engineering, Chennai. She has published papers in national and international conferences as well as in reputed journals. Her research area includes Cloud Computing, Internet Technologies, Machine Learning and Blockchain Technology.

Dr. M. Roshni Thanka received her Bachelor of Engineering degree (Computer Science) in 2006 and Master of Engineering (Computer Science) degree in 2008.She has completed her Ph.D. in the areas of Cloud Computing. She is currently working as an Assistant Professor in the department of Computer Science and Engineering in Karunya Institute of Technology and Sciences, Coimbatore. Her research interest is mainly based on Machine Learning, Cloud Computing and IoT.

Dr. E. Bijolin Edwin is an Assistant Professor in Department of Computer Science and Engineering, Karunya Institute of Technology and Sciences, India. He received his Master of Engineering in Computer Science and Engineering from Anna University, Chennai, India. He has completed his Ph.D. in the areas of Cloud Computing. His research interests include Cloud Computing, Machine Learning,IOT. He is an Life time member of Computer Society of India.