ISSN No:-2456-2165

Distributed Workspace Using Cloud Technology and Version Controlling

Nikhil Pujari
Department of Information Technology
St. Francis Institute of Technology
Mumbai, India

Rohit Choudhari
Department of Information Technology
St. Francis Institute of Technology
Mumbai, India

Rahul Shirolkar
Department of Information Technology
St. Francis Institute of Technology
Mumbai, India

Dr. Minal Lopes
Department of Information Technology
St. Francis Institute of Technology
Mumbai, India

Abstract:- Efficient management of stakeholders in any web project is need of the time. There are lots of stakeholders involved in a web project and managing everyone and everything becomes messy. To make the job easier there for platforms that facilitate distributed development. With this system, "Distributed Workspace", features of existing applications have been integrated and more features have been appended to make development much easier. It is done by the algorithm which is based on microservice architecture. The application aims to provide better source code accessibility and version control. It automates project operations and other development tools all in one place. Mainly, it eliminated the overhead of downloading the project files to the local machine. All these make a platform where users can works, coordinate with each other and share their resources, bridging the communication gap in a more efficient manner than the existing applications.

Keywords:- *ERP, Distributed Workspace, Developer Workspace, Cloud Computing, Live Code Editor, Version Control.*

I. INTRODUCTION

In today's world communication is a key to the success of any project. The work "Distributed Workspace" is based on the aspects of providing ways for better communication between project team members [1]. This is done using *distributed cloud servers* i.e. more than one server acting together to perform a large task in the system. In this system these distributed servers are Amazon's EC2 instances. EC2 is a service that provides us to compute on virtual machine. These distributed servers communicate with each other using API and are connected in a network storage. This system uses an amazon storage service called the *EFS*, designed to provide scalable, elastic, concurrent and encrypted file storage. Every EC2 instance makes use of this file system.

The objective of this work is to develop a platform where resources & work of every developer can be viewed and shared by authorized co-workers. This platform which can be used in a website developing organization, where website resources can be bundled in one place [3].

This creates a workspace for every member in a project to view and use the resources for development. This workspace gives the developers the luxury to host their project within the same platform without the need to download the repository and run it on a local machine [3]. This system provides features like creating a roadmap, assigning task and contact members. An efficient algorithm and such added functionalities make this as a practical application.

II. RELATED WORKS

This section discusses the literature surveyed to build the idea of distributed workspace with unique features. It was identified that some existing workspaces use only cloud computing, only distributed computing or only version controlling. The idea proposed in this paper integrate these features and aims to overcome the drawbacks of these existing projects.

- *Microsoft Azure*

Windows Azure platform offers a runtime execution environment for managed code to host and run scalable solutions. Each Windows Azure Compute instance is also a Virtual Machine (VM) instance created by the platform and only the number of instances is configured by the team hosting the application. Every VM instance runs an Azure agent to connect and interact with the Windows Azure fabric. Every VM has a local file system which can be utilized by the web/worker role instance during their life-time, but once the VM instance is shut down, VM and local storage will go away. Azure maintains 3 different instances of every application on the cloud and the end-user will not be aware of which instance is serving the specific request. Hence persistent storage is required to support the application data and this can be met using the Windows Azure Storage Service. With geographically distributed canters, Windows Azure Compute provides developers with the functionality to build, host and

manage applications on the cloud. Application developers can connect to Windows Azure portal using Windows Live ID and choose a hosting account to host applications on the cloud and a storage account to store data or any relevant content on the cloud. Certain applications can use either the hosting or storage accounts or both. The accounts enable developers to host and deploy applications on the Windows Azure platform. Windows Azure presently supports three roles; web role instance, worker role and VM role.

- *Palantir: Coordinating distributed workspaces*

By our observation Palantir not only requires the main user to have knowledge of their own set of work but it also requires the user to have knowledge about the changes done by his/her developer colleague. Palantir provides the developer with a graphical display. Specifically, Palantír allows developers to better coordinate their activities by providing each developer with a graphical display that shows the set of artifacts they are modifying, meta-data about those artifacts (e.g., artifact name, version number, author information, etc.), and the severity and impact of the modifications being made in parallel by other developers. Knowing this information allows developers to better assess the ongoing activities and accordingly coordinate their activities amongst each other.

III. PROBLEM DEFINITION

Through the rigorous literature survey, it is observed that, in a typical distributed project work, where more number of people contribute, it is difficult to communicate with each other in a true manner [1]. The common problems encountered by any website developer are assigning sharing, tasks, hosting, having a progress report.

The work in this paper, aims to solve these problems by providing an environment where users can work in synchronism, controlled by a project administrator. The other objectives of this work is to provide flexible version control. While working on different technologies and modules, it may sometimes become necessary to revert back to the older versions of the modules. It is observed that this is extremely difficult in most of the existing workspace solutions because the current file is modified by multiple users simultaneously. [2].

The drawback of the systems which provide version control is that it mandates the entire project repository to be downloaded on the local machine making them less user friendly.

IV. METHODOLOGY

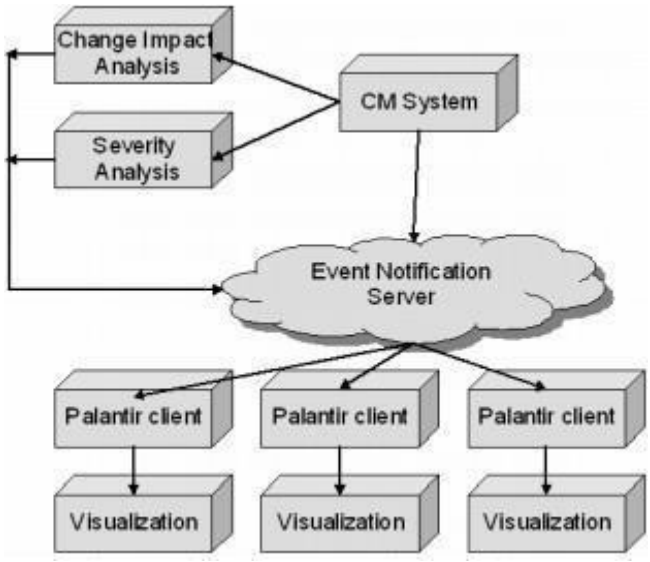

Fig 1 Conceptual Architecture of Palantír.[6]

The above figure displays the architecture used for the creation of Palantir. The main role of the Palantir Client is to intercept the severity and change impact events and translate them into an internal representation that is subsequently used by the visualization block. At a minimum, the visualization component will show which artifacts are being checked-out and checked-in and the severity and impact of each of the changes. In doing so, developers are presented with an increased level of awareness of other developers' activities. We intend to develop a range of visualizations from which each developer can choose the one they prefer. Each of these visualizations will be based on a different balance among the amount of information displayed, interface usability, and intrusiveness of the interface.

A. *About the system*

Distributed workspace is categorized in two parts, a) front end: It is the dashboard available for the admin, and b) the backend that handles the files and maintains version control. The front end dashboard interacts with the user/admin where they can edit files, add files, add employees, view their work etc. The backend consists of two servers, 'the php server' and the 'Linux powered got server'. All actions performed in frontend are received at the backend. The steps are as follows:

By referring to Fig 2, the flow of the project can be understood. First any user would visit the website having information about us and how he/she could use the service. When a new user signs up, if it is a company, it will register itself and get a repository allocated with its company name on the server. If it is an employee, he would have to enter the unique company pin(known to the company admin only) to be registered into the company.

ISSN No:-2456-2165

Once the user has signed up he/she would be prompted to sign in. On successful sign in, the website would open up either an admin dashboard, if sign in is performed through a company email id or to the user dashboard, if sign in is performed through an employee's email id.

In an admin's dashboard, the admin can add/Delete employees in his company and add projects and upload necessary files into the repository. The admin assigns a project manager whose role is to select required employees added in the company to his project and maintain the repository and maintain a To-Do list. In a user's dashboard, the user/employee can view his colleagues in the company he works, view projects assigned to him, keep a track of his works and tasks in the To-Do list and view files in the repository assigned to him and edit code using built-in code editor.

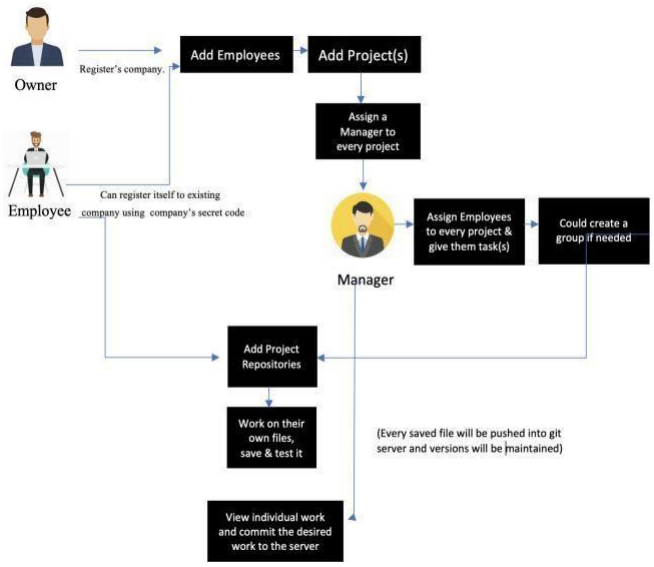

Fig. 2 Flow of Distributed Workspace

At the backend, it is mainly making use of two servers, one is the php server and other is the git server. While creating a project a git repository is created in the git server. Admin and manager are assigned with the developer (dev) branch and employees with their own branch named after their email id. When the user has to access the project files, a request is sent to the git server to load the files in the frontend. The editor will have a file structure column, which will be received from the server. It would only be the structure and not the files to save the server load. Once the employee selects a file, the request would be sent to the server and the file will be fetched and displayed in the editor. When the employee saves the file, it is stored in the php server. When the work is done, the employee commits the file, with a commit message. Once this is done the files saved on the server are committed and pushed to the git repository.

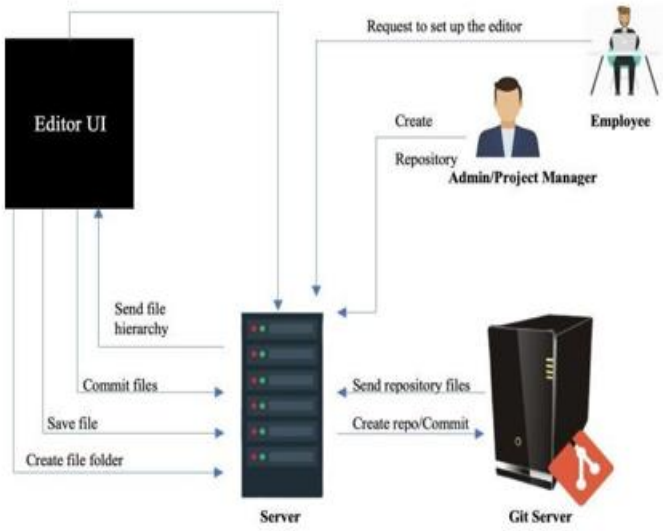

Fig. 3 Architecture of Distributed Workspace back-end

It can be seen that at the backend five servers dedicatedly perform their respective tasks. The operations performed through these servers are explained further.

Operation 1: Source code related operations

Step 1: Fetch source code related files to EFS file system from git.

Step 2: Send the hierarchy of the files. *Step 3:* Perform operation on files the EFS files by using the root id and the file name and the type of operation to perform on the file.

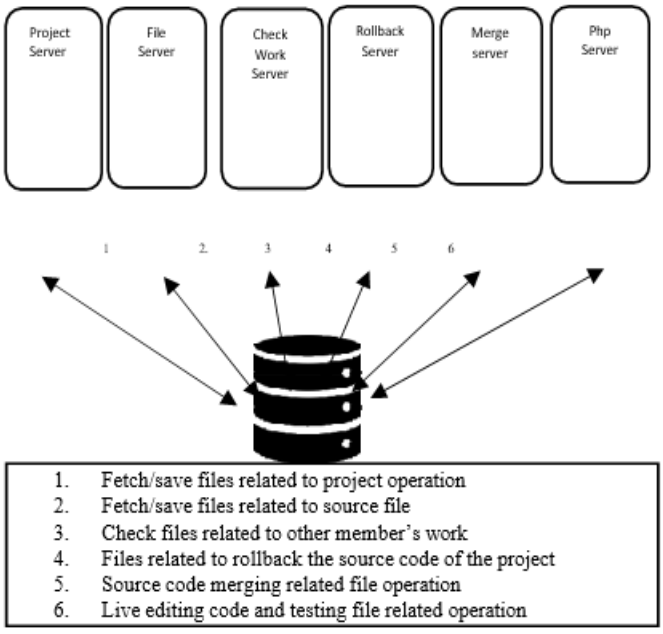

Fig. 4 Microservice Architecture

ISSN No:-2456-2165

Operation 2: Test the work live.
Step 1: Load the editor from the above scenario. *Step 2:* Save the files on the server.
Step 3: Access the result of the file by going to the link: http://live.dwspace.tech/root_id.
Step 4: The php server will check the files from the EFS system and retrieve the necessary files from it.

Operation 3: Project related operation
Step 1: Create repository using the git init –bare command.
Step 2: Add branches to the repo using the name of the user.
Step 3: Perform project related operation on the repository like delete branch, delete repo.

Operation 4: Rollback the branch
Step 1: Load the logs for that users on page load of the corresponding project.
Step 2: Select the log for resting the head.
Step 3: Set request to the rollback server.
Step 4: Server will fetch the files on the EFS system and then perform git reset –head command on the source files.
Step 5: Force push the code and the git tree on the remote repository.

Operation 5: Merge 2 branches
Step 1: Load the repository files on to the EFS server. *Step 2:* Return the branches w.t.r. to the project. And send back to the user. *Step 3:* user will send the base and upcoming branch *Step 4:* Server will fetch the base branch and then the upcoming and branch.
Step 5: Server will perform merge operation on to the base branch.
Step 6: Server will then find the '=======' sign for conflict detection.
Step 7: Send back the hierarchy to the user. Step 8: Use will fix the conflict and the commit the code.

```
<!DOCTYPE html>
<html>
<head>
  <title>GC Merge Demo</title>
  <link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
<<<<<<< HEAD
  <h1>Grand Circus Merge Demo</h1>
=======
  <h1 class="header">Merge Demo</h1>
>>>>>>> 1d46372af5a97f8ef05b9eecb82712382cc5f31c
  <p>
    Demo the merge.
  </p>
  <p class="footer">
    Grand Circus Detroit
  </p>
</body>
</html>
```

Fig. 5 Format of conflict detected by git

Fig 5 shows the example of how the merge server sends a conflict file when a conflict is detected. The conflicted file returned by the 'git handler' has a specific format which denoted where and on which branch the conflict has been raised. The code between '<<<<<<< HEAD' till '=======' is the part of code (in the base branch) which is different from the code in the incoming branch starting from '=======' till '>>>>>>> (followed by the hash code of the incoming branch)'.

Distributed workspace project simplify this task of identifying the conflict by separating the two files and highlighting the conflicted area in the files.

The task for identifying this conflict is simplified in the system where the conflict file is separated as two different original files and the conflicted area is highlighted.

Operation 6: Checkwork
Step 1: Send request to the checkwork server for creation of the other branch source code file on the EFS system.
Step 2: On the completion of the creation. Server will retun a root key to the user.
Step 3: User can access the details form workdone. Dwspace .tech/root_id

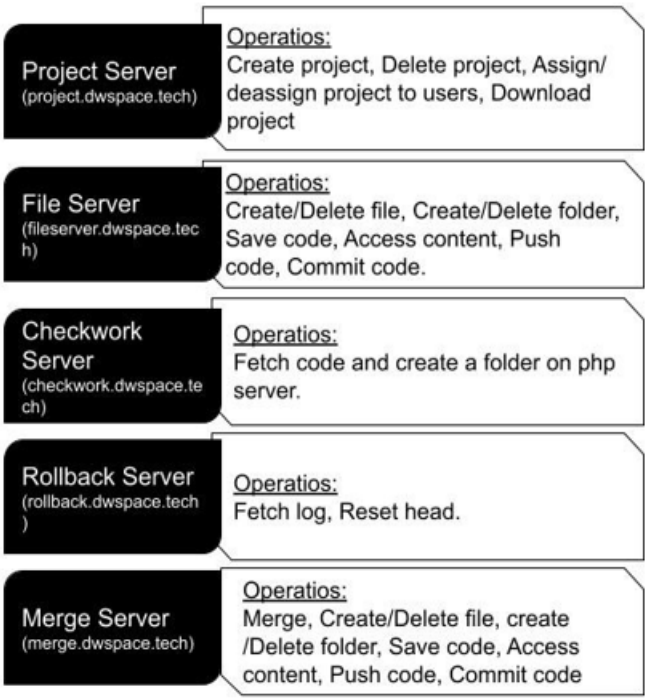

Fig. 6 Micro server and its operations

ISSN No:-2456-2165

V. EXPERIMENTATIONS AND RESULTS

A) Application of the algorithm in the system

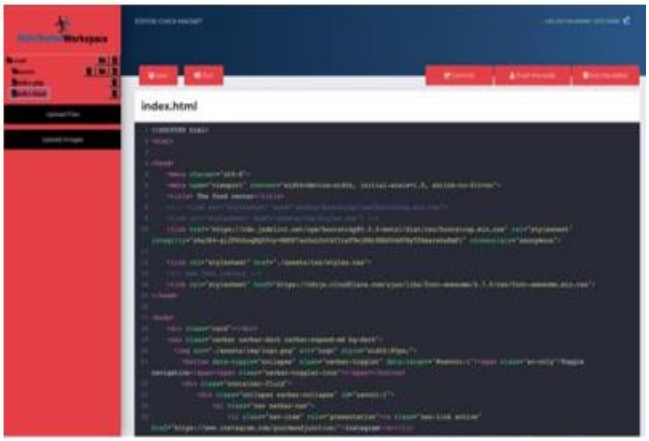

Fig 7 Admin Editor

Fig 7 shows the admin is assigned with a 'developer(dev)' branch. This branch is considered as a production branch where the code stored is the final expected output of the project. The dev branch has a default file structure which can be altered by creating a new file/folder or by uploading files/images from the local machine. Every change done in the repository has to be committed with a commit message and should be pushed.

Fig 8 shows the merge conflict page. As mentioned above, the developer(dev) branch is the ones whose code is considered as the final deployment version. As other employees are working on their respective assigned branches. At the end, the admin has to merge the code.

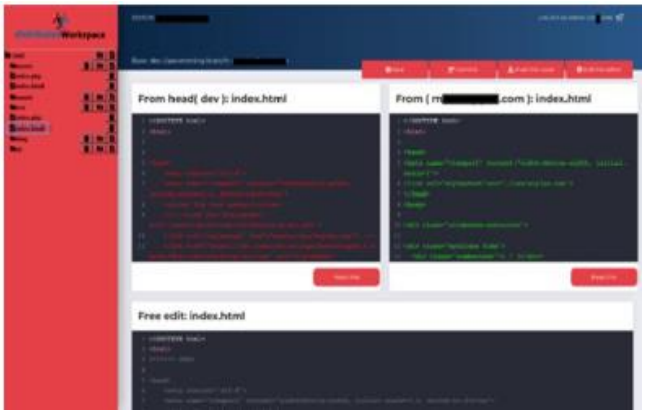

Fig 8 Handle Merge Conflict

During the merge process it is very much possible that a conflict might arise between the 'dev' branch and the employee branch. This is catered by this module, where the admin decides which code stays in the 'dev' branch. The code highlighted in red, depicts that this part is different from the other branch and here the conflict has arisen.

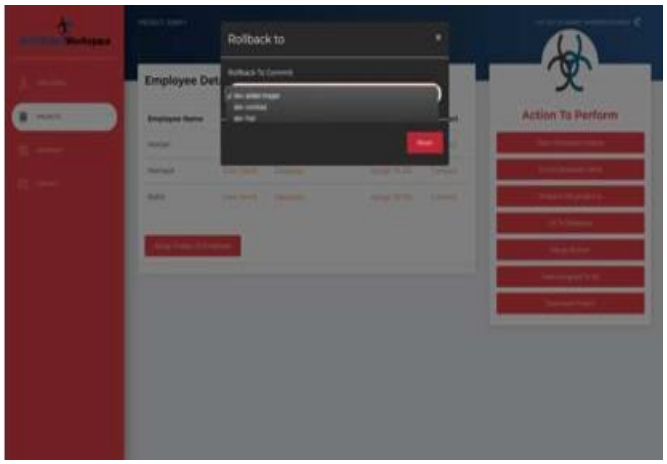

Fig 9 Rollback

Fig 9 shows Rollback feature used in version control. It is possible that at some point the admin needs to retrieve the code with older version compared to current version code which user has committed recently. This can be done by clicking on it. the rollback button and selecting the commit message. All changes will be retrieved till the point where that selected commit was performed.

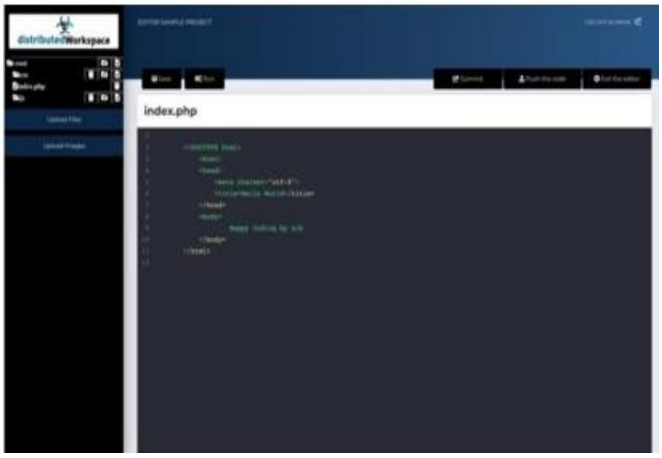

Fig 10 User Editor

Fig 10 shows the user editor. It works similar to the admin editor. It displays the file allocated to the respective user branch. Any work performed, when just 'saved', is stored on the file server, once it is committed and pushed to the user repository, it is saved in the 'git' server.

B) Performance analysis

Although the current version of this system is built at the lowest level of hardware configuration, it has gained a very decent range of performance. This can be easily scaled up using vast service options from AWS and any other available cloud service providers at the time of deploying this project live for commercial use.

ISSN No:-2456-2165

Table 1.2 shows the performance test results of important modules of this project and the time taken to perform it.

Table 1.2 Application performance analysis

Sr no.	Module Name	Operation performed	Time take n (in sec)
1.	All CRUD Operations	Operations performed on the mySQL database like IN-SERT, UPDATE, DEL ETE.	<1
2.	Project creation	Git repository created on the Git server (installed on EC2).	3.78
3.	Editor loading	Fetching file hierarchy response from the server.	2.30
4.	Loading a file in the editor area	Fetch the file content from the hierarchy sent.	1
5.	Upload Image	Image is base64 format to the hierarchy selected.	2.1
6.	Save File	Content from file content is fetched and stored in the corresponding file hierarchy.	1.2
7.	Commit	All unsaved files are saved on the php server and a 'git commit' command is executed.	1.5
8.	Push	'git push' command is executed	2.3
9.	Load rollback logs	Logs of all the of the corresponding branch is requested	5
10.	Populate branches options during merge	Fetch all the branch name belonging to the corresponding project.	2.3
11.	Merge operation	Fetch base branch and the merge with in coming branch	4.5
12.	Download project	Fetch files from git server, store it in a folder, convert into zip and return this folder name in response.	3

Above performed task were carried out on branches which sized around 12MB. These numbers can vary depending on the size of the files. The git which could perform a merge of two 12 MB branches in 4 seconds could take even 10 minutes for a 100MB sized branch. Likewise, an image of 200 KB takes less than 2 seconds to be stored but an image of 8 MB could take more than 5 seconds. This depends upon the file size and the server capacity.

VI. FUTURE SCOPE

Distributed Workspace, while building aimed at eliminating the hardships taken by the members of a web project. The system is believed to overcome most of the problems that existed in the current system by providing the development means all at one place, in a user friendly fashion. Though, it is always said that everything has a margin of improvement in them. So while consistently improving the existing features in this system, the system has impressive future scope as well.

As Distributed Workspace is capable of managing multiple projects, the system can provide a periodic *project report*, on the basis of which the admin could understand whether or not the given task is completed by the employees . The efficiency at which the project is being built can also be calculated. There is a vast scope of improvement in our editor. We can provide a number of *production tools and extensions for our editor* like, syntax checker, parenthesis highlighter, theme change, beautify tool, live share, auto completion, multiple cursor, etc. We could add *website templates* to make the work of developers easy, using the templates a lot of workload on the developers could be reduced. We can give the testers a dedicated area to *create test cases* and run the , for checking errors.

The features we provide to the public free of cost could be optimized and *customized according to the clients/companies requirements*, like providing an error detection system and also equipping the system with a compiler if need be. Another source of revenue can be providing *domain names* of their own company name, as the free one will have our domain name(dwspace.tech). We could also add *video conferencing* to the system so that communication is easier between the members and a more professional environment can be maintained.

VII. CONCLUSION

The main focus of our project was to overcome the drawbacks of existing systems and provide better communication, by providing the users with a workspace to communicate. We have tried to integrate practical features into the application which other applications may lack. Our algorithm makes distributed computing more efficient with the help of microservers, each assigned with an independent role. This can be seen in the application performance table, above in the report. The system has taken care of security aspects by giving a special privilege to the admin and controlled access/privileges to the employees. We are making a project environment that will be used for distributed product development on a rapid scale using agile project management. The project with its efficient algorithm has tried to be a better alternative and at the same time believe that there is more scope for improvement and make it a purposive application.

There are more applications like Github, Amazon Code Commit[9], BitBucket, Google Cloud Source Repositories[10], etc with similar concepts. However, this system is been inspired by these mentioned application's core concept of version control. This system has made maintaining the codes simpler for students or individuals, working in a small-scale project to use the system without worrying about the complications of version management. By providing a simple user interface and giving project management tools like roadmap, to-do, contact(chat), live-testing, etc., this system stand out from the other similar technologies in the market.

REFERENCES

[1]. Sutherland, Jeff, et al. "Distributed scrum: Agile project management with outsourced development teams." *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. IEEE, 2007.
[2]. MacGregor, Steven Patrick. *Describing and supporting the distributed workspace: towards a prescriptive process for design teams*. Diss. University of Strathclyde, 2003.
[3]. Padhy, Rabi Prasad, Manas Ranjan Patra, and Suresh Chandra Satapathy. "Windows azure paas cloud: an overview." *International Journal of Computer Application* 2 (2012).
[4]. Daud, Nik Marsyahariani Nik, Nor Azila Awang Abu Bakar, and Hazlifah Mohd Rusli. "Implementing rapid application development (RAD) methodology in developing practical training application system." *2010 International Symposium on Information Technology*. Vol. 3. IEEE, 2010.
[5]. Zahariev, Alexander. "Google app engine." Helsinki University of Technology (2009): 1-5.
[6]. Sarma, Anita, and Andre Van Der Hoek. "Palantir: coordinating distributed workspaces." *Proceedings 26th Annual International Computer Software and Applications*. IEEE, 2002.
[7]. Wiil, Uffe Kock, and John J. Leggett. "Workspaces: the HyperDisco approach to Internet distribution." *Proceedings of the eighth ACM conference on Hypertext*. 1997.
[8]. Fernandez, Daniel J., and John D. Fernandez. "Agile project management—agilism versus traditional approaches." *Journal of Computer Information Systems* 49.2 (2008): 10-17.
[9]. Amazon Web Services, *Amazon CodeCommit Documentation*. July 09, 2015. Available: https://docs.aws.amazon.com/codecommit/index.html
[10]. *Google LLC, Cloud Source Repositories | Documentation*. August 26, 2021. Available: https://cloud.google.com/source-repositories