2014

Compiler as a Service
-
Fault-tolerant Distributed C/C++ Compiler (FDCC)

18-842 GROUP PROJECT, SPRING 2014

AUTHORS: AKSHAY GANDHI, TAO YU, WENDI ZHANG, ADITYA JALTADE
GROUP 16

Abstract

Compilation is a resource intensive process. Sufficiently large projects require a lot of resources in terms of time and CPU resources for compilation. For developers of such projects, having access to such resources is many times not possible. Distributing a compilation process over multiple nodes in a network can help ease the resource requirements from a single node and result in faster compilation by utilizing the combined resources of all participating nodes.

This project aims to create a fault-tolerant distributed compiler service for open source C/C++ projects. This service, Fault-tolerant Distributed C/C++ Compiler (FDCC) will enable developers and other users to compile their code in a distributed environment, supporting a variety of compiler versions. Thus, this distributed compiler infrastructure provided by FDCC will serve to reduce the compile times of large projects, and at the same time also provide fault-tolerance to the whole process, to ensure that compilation proceeds even in face of node failures.

Table of Contents

1. Introduction

1.1. Purpose of this document

The specification and design document provides detailed requirements and high level design of FDCC.

1.2. Scope of the document

This document provides background relevant to and details of the implementation planned for FDCC prototype. Key data structures, communication techniques, and implementations details are included. Additionally a description of the planned demonstration environment and functionality to be evidenced are included.

1.3. Background Information

FDCC is a tool to help open source project developers compile their projects using various compiler versions and library versions. It can also be used for speeding up compilation of source code by using distributed compilation over a computer network. With the right configuration, FDCC can dramatically reduce a project's compilation time.

FDCC is designed to speed up compilation by taking advantage of unused processing power on other computers. A machine using this service can send code to be compiled across the network to a computer which has the daemon and a compatible compiler installed.

1.4. Motivation

Open source products are often distributed with source need to be compiled against various compiler version so that to make sure good end-user experience. While most open source developers are actually developing projects as a hobby and don't really have access to fast and varied compilation environments. In fact, setting up these environments is non-trivial in most cases. That's why we need FDCC.

The convenient properties of FDCC have made open source developers' work easier. These developers no longer need to install various environment on their own workstation, instead they can easily connect to our server to use FDCC service to gain the binary. When developers have to compile some complicated code, like Linux kernel, they can also choose to use FDCC, not only will the compile time be shortened, but by submitting the job to FDCC, the compilation would not occupy their own workstation.

In reality, there might be problems raised by the uncertainty of the environment when using FDCC. For example, a network performance problem may have an impact on the performance of FDCC; or even some nodes may fail while processing. This would not affect the final outcome of FDCC since it is designed for robustness. However, we prefer to generate a report to keep track of status

of the job during the whole process and let end-user see the report. Such a report would likely to help developers or network administrator figure out what could go wrong in the previous attempts. We can always keep these information logged for the purpose of auditing or some other issues like that.

1.5. Goals

• Provide access to varied compilation environment without any changes to the configuration of the native machine.
• Utilize a collection of server to allow for fault-tolerance as well as parallel processing.
• Shortened time of compilation when the source code is really large.
• Provide users/developers ways to track the process of compilation.

1.6. Application Areas

FDCC can be put into some different areas, one of the simplest concepts for the utilization of FDCC is to be able to run a compilation somewhere else when your own machine is too busy or otherwise does not have the required resources. Often, the hierarchical relation of a project can be explicitly acquired through the Makefile. Thus by getting the information we need, it is possible to break the project into several sub-projects so that to make it possible to execute more of the compilation in parallel. In general, any project for which the compilation can be easily parallelized is a contender for using FDCC.

1.7. Use Case

A sample use case would be compiling KDE's basic modules and libraries. One of the most frustrating aspects of open-source development is all the time spent waiting for code to compile. Compiling KDE's basic modules and libraries on a single machine takes around three hours, and that's just to get a desktop. Even with a Core 2 Duo, it's a lot of time to sit around and wait.
When using more than 6 nodes to compile the same modules, it is estimated to reduce the cost of time by 80%. And the user is free to do anything on his laptop during this period.

2. Functional Requirements

2.1. Description

We aim to provide Compilation as a Service. If a programmer feels that he does not want to utilize the resources at his local machine for compiling a big project or if he does not have the required resources for compilation at his disposal, he can upload his code and our system will do the job for him using a distributed approach.

2.2. Availability

Our system ensures that the user jobs are run to completion even in the presence of node and link failures. Our design is such that if a compiler node goes down, its responsibility can be offloaded to one of the other compiler nodes, thereby ensuring availability. Also, if the bootstrap node fails, there is sufficient connectivity to ensure that one of the other bootstrap nodes will take over the responsibility and ensure that the user does not see disruption in service and is able to download the binary once the compilation job completes successfully.

2.3. Technical Issues and Risks

- **Security**: Security is very crucial in a distributed environment. If our system was not being used for compiling open source projects, security measures would be needed to ensure that other developers do not get access to the code submitted by some other developer. But since all the submitted code would be open source anyway, we can overlook this security aspect.

- **Load Balancing:** Since the application is user facing, in order to keep the system responsive, overloading of any peer must be avoided. Any extreme latency due to overloaded network links or peers should be avoided since this could significantly impact the usability of the application. This intelligence is built into our system. It avoids overloading a subset of the servers more than the others.

- **Job Management**: The system needs to select candidate compiler nodes and distribute the job amongst them taking care of all the dependencies. This functionality is implemented in the bootstrap server nodes and is transparent to the user. The user should not be aware about the location and the IP addresses of the compiler nodes that are responsible for compiling the job submitted by them. Also the system provides means to track the status of a job while it is running, its completion information and other usage statistics via appropriate user interface.

- **Node Bootstrapping**: Our system allows compile nodes to be dynamically added to the network and leave the network at any time. Total network connectivity is ensured by a bootstrap service by providing multiple connections among the nodes. Also, when a new node is added to the network, its state is collected by the bootstrap node and this state is

passed on to the other nodes via heartbeat messages. Load balancing is implemented in our system to ensure that the resources at each node are utilized in the most optimal fashion and that none of the nodes are over-utilized or under-utilized.

- **Efficient Resource Allocation**: Since there can be multiple jobs running simultaneously on the system, it should be ensured that the execution of one job does not affect the other jobs in the system. The bootstrap nodes keep track of the global state, that is, the state at all the nodes and distribute the jobs among the different compiler nodes in a manner that all the jobs are executed as if they have the same priority.

3. Specific Requirements

3.1. User Interfaces

Most users today understand the concept of a Web portal, where their browser provides a single interface to access a wide variety of information sources. The FDCC portal provides the interface for a user to launch a compilation that will utilize the resources and services provided by us. It would allow users to specify node requirements and define compile jobs. From the perspective of the resource provider, the portal would be the interface to make the resource available/unavailable at any instant. Users will be informed after the compilation result is available. Another addition that we made was to provision for users to track progress of their jobs in real-time. Both users as well as providers.

3.2. Job Manager

Once the user uploads the source code, the system will launch compute intensive applications. Based on the Bootstrap node's information, it will identify the available and appropriate nodes to utilize the resource. This task is carried out by a job manager. After the Bootstrap server's job manager (the back-end) receives the source code, it will assign one super compile node. Then the Job Manager finds the available nodes and partitions the job based on the numbers of files that need to be compiled. Each job node (compile node) will maintain its status and send its status to other nodes. After all the compile nodes finish their job, they will send the compiled files back to super compile node. Now, super compile node will link the compiled files together and generate the executable file and send back to the back-end server node. After this job finished, the user can download the compiled binary from using the user interface.

3.3. Network Manager

The Network Manager ensures the node's connectivity to the rest of the distributed system. Through timely heartbeats, the network manager indicates the node's liveliness to back-end server node and detects neighbor nodes failures. Meanwhile, the network manager can add new joined nodes' information to back-end server node. Nodes' failures would trigger update messages throughout the network, resulting in a consistent network topology graph of the current network. The network manager would also be responsible for routing sub jobs to ordinary nodes and fetching back the result.

3.4. Reliability

Reliability is a measure of the probability that our service will perform its intended function for a specified interval. We try to implement FDCC by using a shadow bootstrap node in case the original bootstrap node goes down. The shadow bootstrap node would have a complete copy of all the information stored in bootstrap node, when the bootstrap node crashes, the shadow bootstrap node will broadcast a message to every node and will behave like a bootstrap node.
Each node also has to broadcast their heartbeat message to every other nodes, as a result when some nodes crash while compiling, we will find out the fact that those nodes crashed and re-

distribute their unfinished jobs to other active nodes. The users won't be affected or even notice that there's something wrong with our inner nodes.

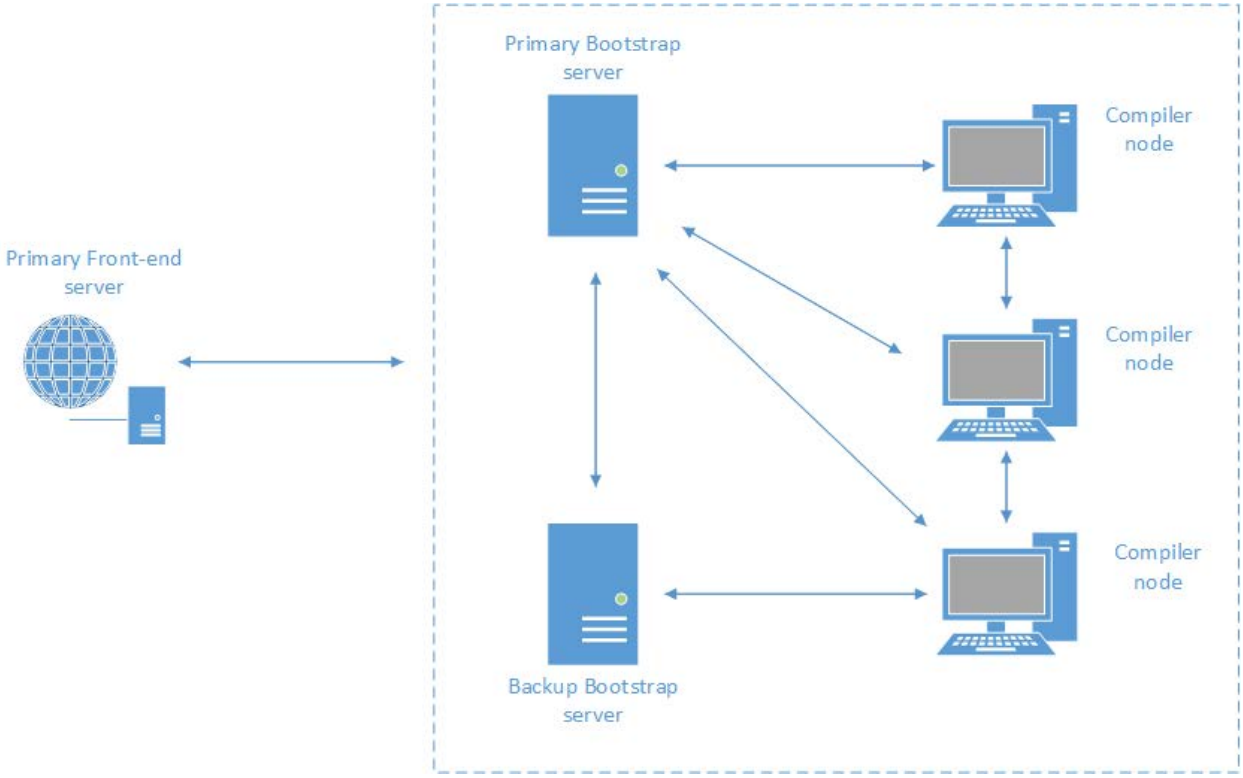

Figure 1. Internal structure of the system

3.5. Performance

The measure of performance for jobs executing in this environment would be the turn-around time for submitted jobs. The FDCC is suppose to achieve a lesser total turn-around time for a job as compared to the time required to execute the complete job on any single node of the system.

3.6. Scalability

The system should be able to handle a large number of nodes in the system with multiple user jobs running in parallel. The large scale would impact the amount of information maintained and network traffic handled at any node. DFCC strive to support as many nodes as it can.

3.7. Portability

Right now the FDCC will only support Linux environment since we still need some support of existing tools in Linux. Extending the system to include Windows and Mac based hosts can be a future extension to the project so as to making users' projects more portable.

4. System Design

4.1. User Interface

The FDCC system consists of a User Interface through which the user can interact with the FDCC system. The interface will allow a user to submit a project to the system for compilation, monitor the status of the compilation. When the compilation is done, the user can download the binary using this user interface.

The user interface will be a website, which we plan to build using Django. The Django front-end will communicate with the Erlang back-end, when it wants to retrieve any information for the user.

4.2. System Architecture

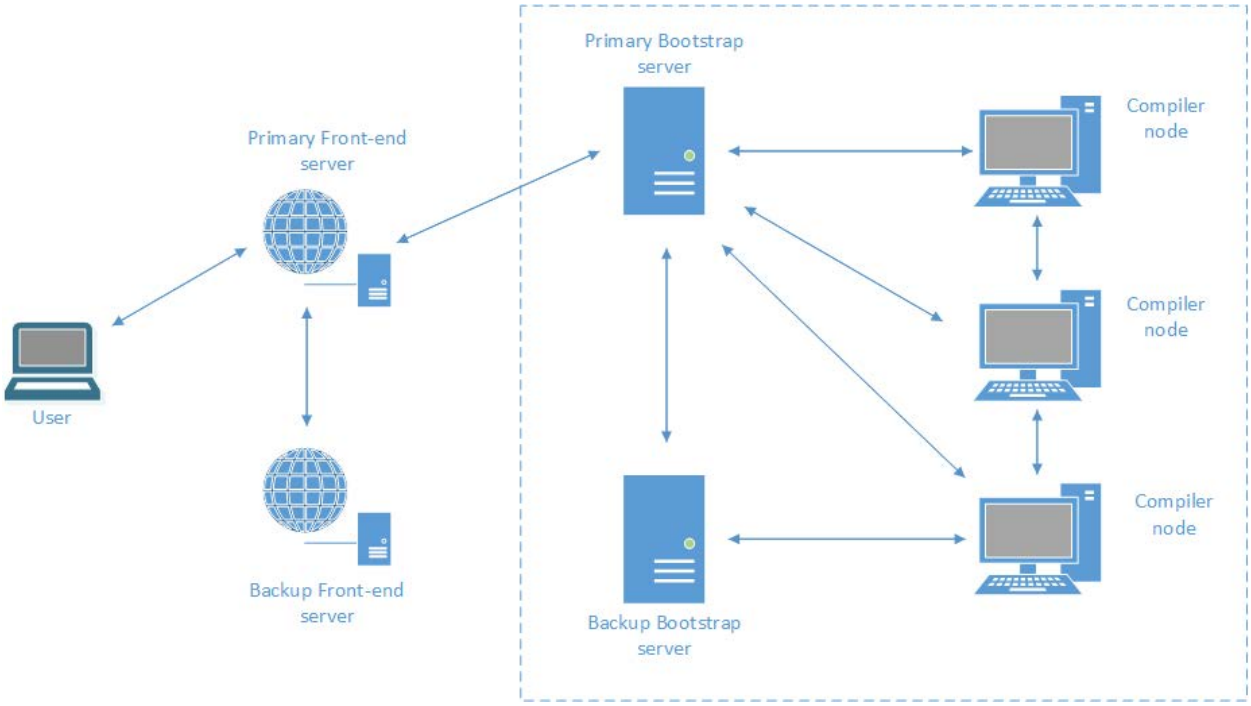

Figure 2. FDCC System Architecture

The system architecture for FDCC is show in the figure 2. It consists of front-end web portals, bootstrap servers and compiler nodes. Each of these components is described in detail, in section 4.3.

4.3. System Components

As seen from the architecture diagram, the FDCC system consists of various nodes each fulfilling a separate kind of responsibility. These nodes can be physically separate machines, but in reality they are just different user level Erlang processes running on the machines.

All the nodes in the system, including the bootstrap nodes, the compile nodes and the super compile nodes, will have a similar architecture but will vary in the duties they perform.
Any node's process will have the following two components - a job manager and a network manager.

- Job Manager - The job manager as described in section 3.2 will be responsible for division of work from the front-end to the super compile nodes and from super compile nodes to the compile nodes. Having this module on each node will enable each node to perform different operation while being a member of different compile networks at the same time.
- Network Manager - As described in section 3.3, the network manager module will be responsible for ensuring connectivity between nodes. This module resides on each node to perform this necessary function.

We plan to leverage the distributed compiler infrastructure provided by the open source project 'distcc' [1].

4.3.1. Bootstrap Node

The bootstrap node is responsible bootstrapping new compile nodes in the P2P network consisting of other compile nodes. Apart from having the job manager, and network manager consists of a separate module which is the Adapter. The Adapter is responsible for interacting with the web application. This is the module which will listen to and serve data to the user interface.

The bootstrap server being a rendezvous point for new nodes, it has to be tolerant towards failures. We plan to provide fault-tolerance through shadowing of the bootstrap node. An exact replica of the bootstrap node will be kept "warm" and updated with the current global state of the system at all points in time. When a primary bootstrap server goes down, the "warm" server can take over the responsibilities and act as the primary.

4.3.2. Super Compile Node

A super compile node is the one which initiates the compilation process. This node will invoke a distributed compiler and distribute the compilation work between various compile nodes. This node will also be the one that performs the final linking of the project. The super compiler node is also responsible for ensuring that the compilation process is not affected by compile node failures. If a compile node is found to have failed, the super compile node will assign the unit of

compilation work to another compile node, thus making progress on the whole compilation process.

The super compile node communicates with the other nodes in the system through the network manager module.

4.3.3. Compile Node

A compile node is the worker node for the FDCC system. A compile node performs the actual compilation of the files that are assigned to it by the super compile node. The only interaction that the compile nodes have with other nodes in the system is through exchange of heartbeat messages which piggyback the current status of the node.

A compile node can join/leave a network as it pleases and it will not affect the outcome of the compilation.

4.4. System Interactions

The following are the main interactions in the system that allow us to capture the essence of the whole system.

4.4.1. Join Network - A new compile node wants to be a part of the network.

When a new node wants to join the network, it instantiates its own network manager instance which is used to communicate with the bootstrap node. The bootstrap node helps the new node with a complete list of all peers. It also notifies the other peers of the joining of the new node. The network managers of all these nodes now manage heartbeat messages and other book-keeping that needs to be performed to ensure application level connectivity.

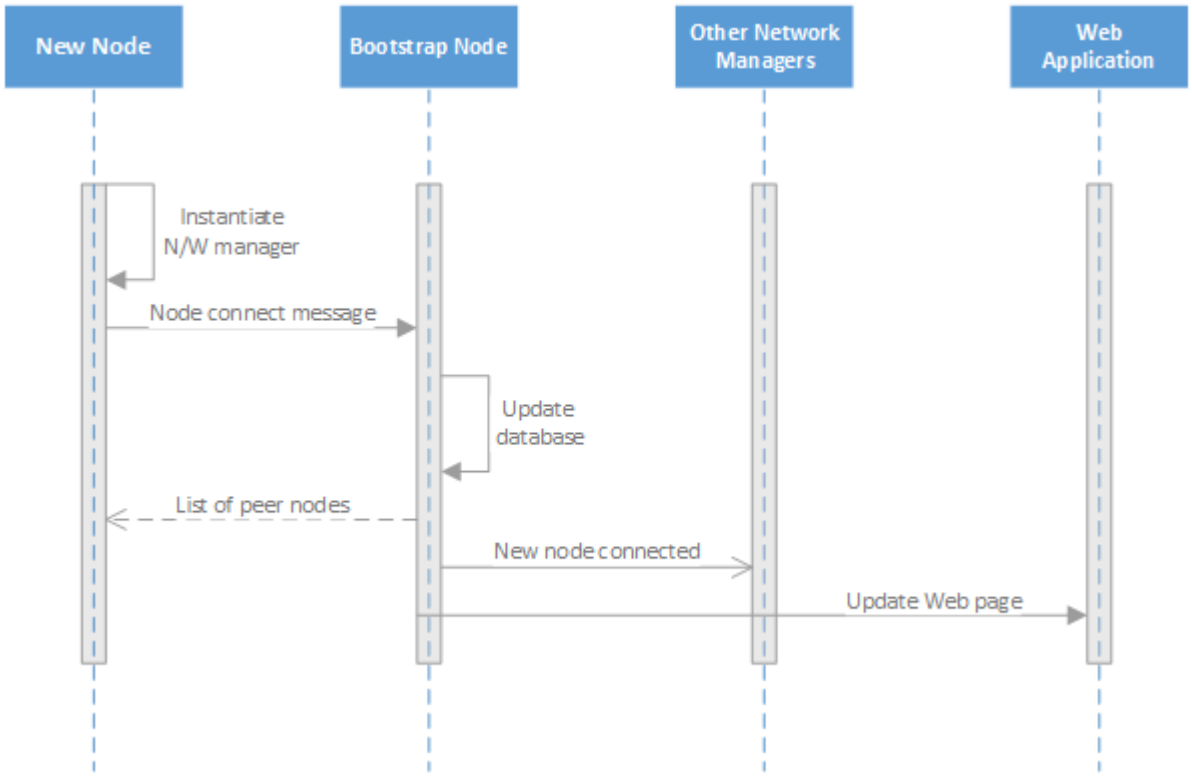

Figure 3. Sequence Diagram - New node joins the system

4.4.2. Submit project - A user submits a project using the web interface.

When a user submits a project through the web portal, the web application (front-end) passes on the project tarball to the back-end Erlang [2] process. This back-end Erlang process (mostly this process also doubles as the bootstrap process) then selects a group of compiler nodes from the list of all available nodes. It selects a super compile node from these participating nodes using the Bully algorithm.

Once this assignment is done, it asks the web application to update the status of the job to ‘Assigned’.

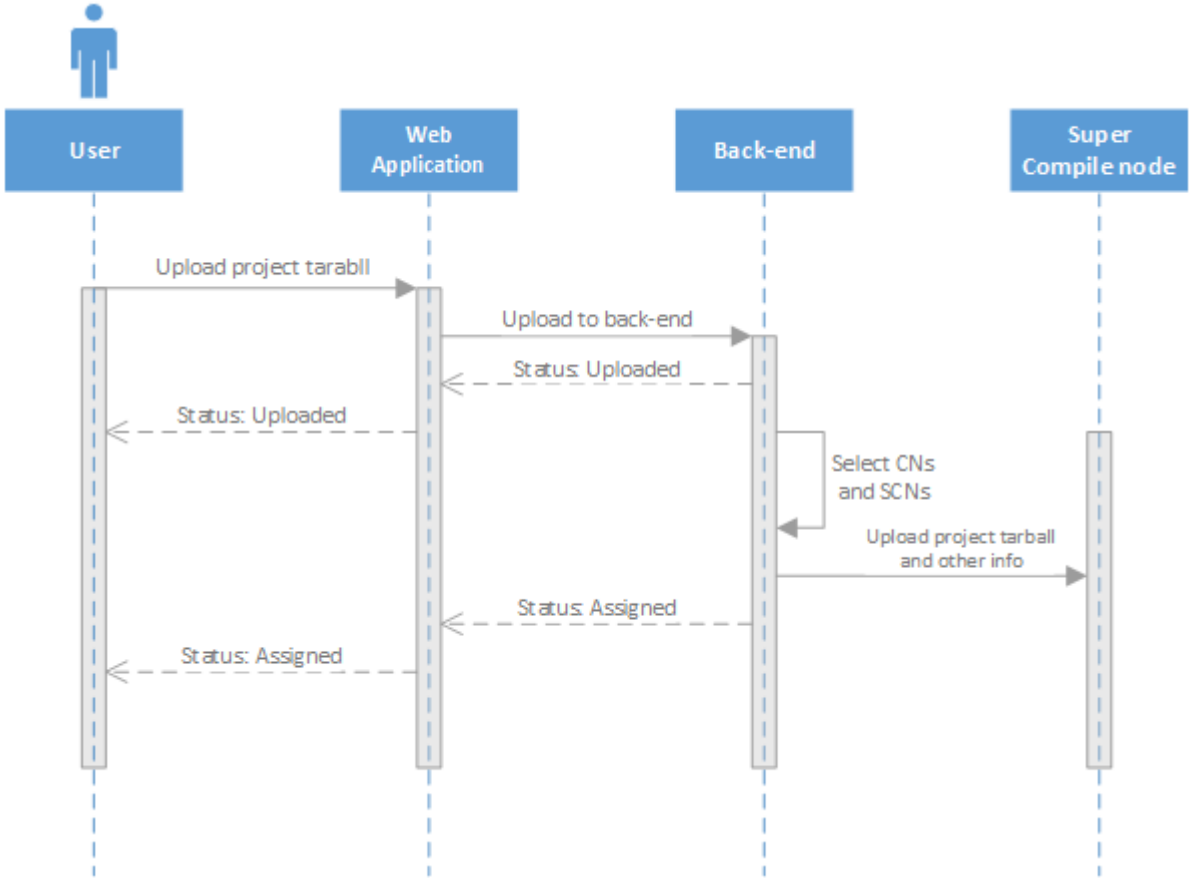

Figure 4. Sequence Diagram - User submits a job.

4.4.3. Perform compilation - A super compile node assigns and performs the compilation.

When the compilation is scheduled on the super compile node, it invokes the distributed compiler and assigns the compilation units to the ordinary compile nodes. The individual compile nodes perform their compilation and pass back the compiled object files to the super node. When the super node has collected all the necessary object files, it proceeds to the linking phase. Once linking is done, the binary is ready for the user to download. This is updated on the website.

During this whole process, all participating nodes are exchanging heartbeat/status messages, to stay up-to-date with the current global state. These have been omitted from the sequence diagram for reasons of brevity.

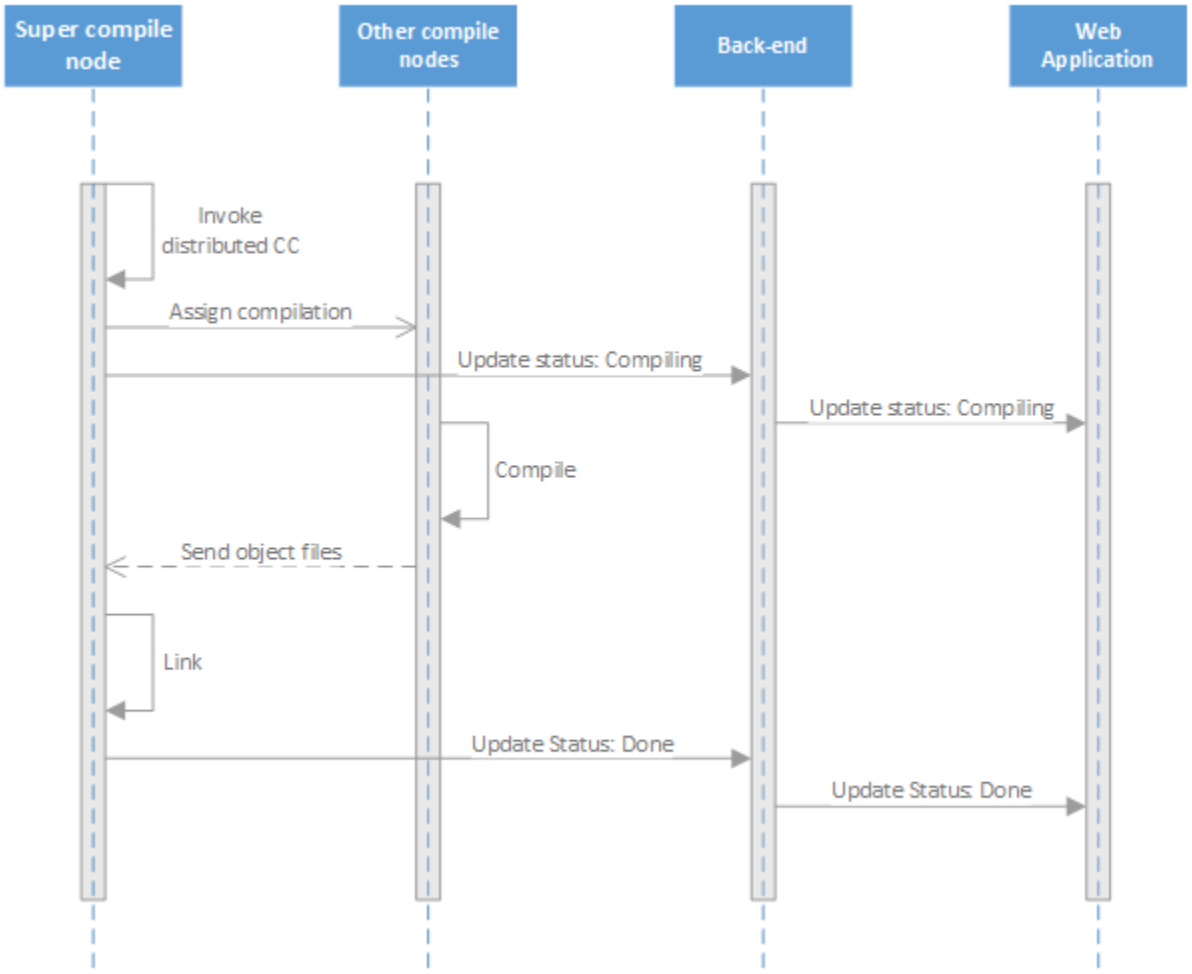

Figure 5. Sequence Diagram - Compilation process

5. Demonstration

5.1 Demonstration environment

The demonstration environment will consist of several hosts connected via Ethernet/Wi-fi. The system will consist of front end servers, tracker nodes and compiler nodes and. We will have a couple of bootstrap server nodes to which new nodes get connected. The client will submit the job using one of the front end nodes which will be connected to the bootstrap nodes. The bootstrap node will distribute the compilation job among the compiler nodes that it controls. The interface at the front end server will display the status of the compilation job.We are going to provide file level granularity.The interface will display the state in terms of the number of files compiled. We plan to provide additional status information, such as the average time for compilation, the number of compilation jobs completed till in a specified period of time and such other statistics using graphs.

5.2 Capability

If you compare the binary generated by compiling the job locally and the one generated using our distributed compiler, there will be an exact match. This renders the distributed version of compiler compatible with the local compilation task.

5.3 Performance

The time taken to compile the job locally using the native compiler is more compared to the time taken to compile the job using our distributed compiler. This results in significant performance improvement in terms of time in comparison to the native compilation approach.

5.4 Fault tolerance

We can demonstrate that our system is fault tolerant by disconnecting any of the nodes from the network and still ensure that the compilation job completed successfully. We plan to design our system in such a manner such that whenever a compiler node goes down, its task can be delegated to other functional compiler nodes. Also if the bootstrap node goes down, one of the other bootstrap nodes can take over its responsibility and ensure that the compilation succeeds.

6. Project Management

6.1. Team Organization

For a group project, one of the most challenging aspects is working out a schedule for several members of the team, each having different commitments. To overcome this, we will mostly be communicating over email and chats. To ensure that everyone is aware of the progress, we meet every Thursday for 2 hours to discuss the progress and assign work.
Each task will have a task leader assigned who will be responsible for ensuring that the particular task gets completed in a timely manner.

6.2. Task List

[AG: Akshay Gandhi, TY: Tao Yu, WZ: Wendi Zhang, AJ: Aditya Jaltade]

Task	Assigned to
Infrastructure	All members
Set up Django and Erlang development environments	
Set up and initialize git repository	
Web Application	TY, WZ
Develop Django web pages with basic layout of pages	
Develop interfaces to bootstrap server	
Develop data structures and stubs for communication	
Bootstrap Server - Part 1	AG, AJ
Manage new nodes connecting to the system	
Assign compile nodes based on load balancing techniques	
Develop interfaces to interact with Web Application	
Integration Cycle 1 (Web Application & Bootstrap server)	TY, WZ

Implement communication between Web Application and Bootstrap server	
Bootstrap node - Part 2 & Super node	AJ, AG
Implement communication protocol between bootstrap node and compile nodes.	
Implement fault-tolerance techniques for compile node failures.	
Implement linking phase	
Compile node	AJ, AG
Implement distributed compiler process spawning	
Implement progress tracking message communication	
Integration Cycle 2	TY, AG
Integrate bootstrap nodes with super nodes and compile nodes	
Testing and bug fixing	WZ, AJ
End-to-end testing and bug fixing	

6.3. Project Schedule

Project Milestone	Timeline
Requirement Gathering	
Brainstorming and project idea definition	Week 1
Design	Week 2-4
High level architecture discussions	
Reading literature	
Design document	
Implementation	Week 5-11
Implementing Web Application	
Implementing Bootstrap server	
Implementing super compile nodes and compile nodes	
End-to-end integration and testing	Week 12-14

7. References

[1] distcc - https://code.google.com/p/distcc
[2] The Erlang Programming Language - http://www.erlang.org