Map Reducer Architecture Document

Phase 4

Group 5: Abi Parekh, Emmanuel Sevieux, Stephen O’Connor
Date: 12/11/22

TABLE OF CONTENTS

1 Introduction 2
1.1 Document overview **Error! Bookmark not defined.**
2 Software Architecture overview 2
3 Software design description **Error! Bookmark not defined.**
3.1 Workflow **Error! Bookmark not defined.**
3.1.1 Component interfaces **Error! Bookmark not defined.**
3.1.2 Component design description **Error! Bookmark not defined.**
3.2 Map Class **Error! Bookmark not defined.**
3.2.1 Component interfaces **Error! Bookmark not defined.**
Component design description **Error! Bookmark not defined.**
3.2.2 Workflows and algorithms **Error! Bookmark not defined.**
3.2.3 Software requirements mapping **Error! Bookmark not defined.**
3.3 Reduce Class **Error! Bookmark not defined.**
3.3.1 Component interfaces **Error! Bookmark not defined.**
3.3.2 Component design description **Error! Bookmark not defined.**
3.3.3 Workflows and algorithms **Error! Bookmark not defined.**
3.3.4 Software requirements mapping **Error! Bookmark not defined.**
3.4 Final Class **Error! Bookmark not defined.**
3.4.1 Component interfaces **Error! Bookmark not defined.**
3.4.2 Component design description **Error! Bookmark not defined.**
3.4.3 Workflows and algorithms **Error! Bookmark not defined.**
3.4.4 Software requirements mapping **Error! Bookmark not defined.**

1 Introduction

All the Architecture information will be captured within this Architecture Document. The design information includes but is not limited to Software Architecture and Process Flow.

2 Software Architecture overview

The Phase-4 Map Reducer Software is a Microsoft Visual Studios based Program that is executed from the Command Line. Below Figure 1 describes the different stages of the program. The Next figure describes how the subsystems fit together while Figure 2, specifically how the Data is passed around the system. The Phase 4 Software can be broken into 4 different subsystem with additional common utilities (see below)

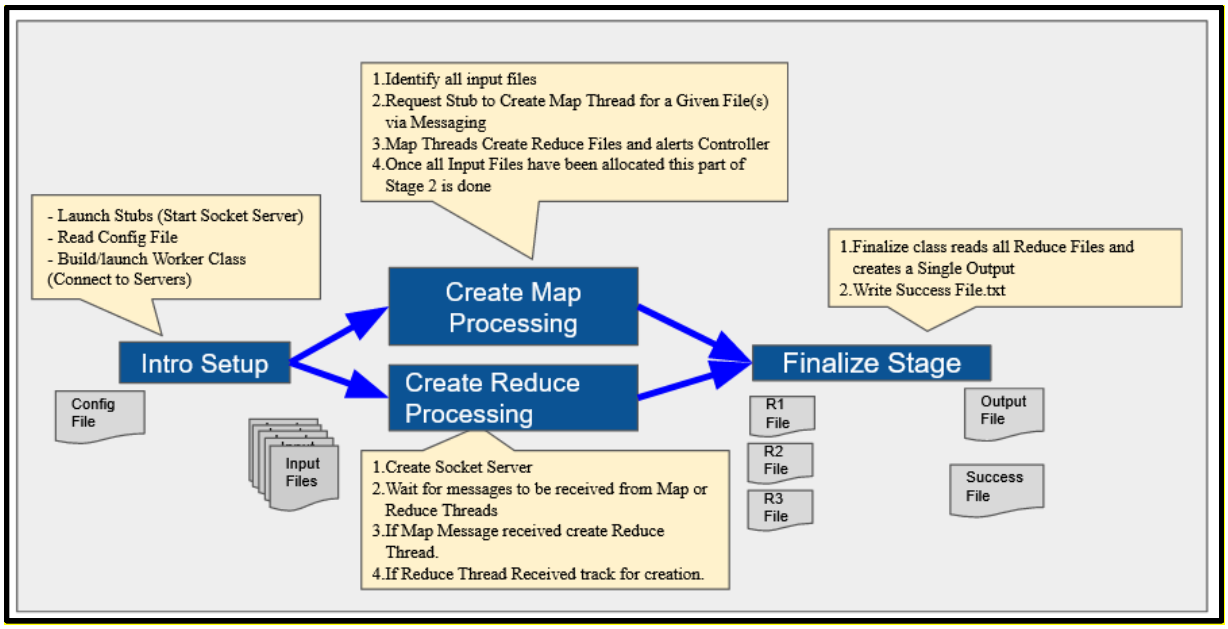
1. **MapReducer Controller**: Top Level Class which validates the input configuration, Requests Map and Reducer threads created, processes heartbeat status of Map and Reducer Class, and creates Final Output.
2. **Stub:** Sits and listens for messages (using sockets). The stub will get a message from the controller to tell it to create child processes (either map or reduce).
3. **Mapper DLL**: Extracts individual words (tokens) from files and exports results to intermediary directory
4. **Reducer DLL**: Arranges tokenized words into alphanumeric sorted order and sums the total occurrence of each word.
5. **Utilities**:

a. **Socket Common Functions**: Set of Common Classes that create and listens to Sockets.

b. **File Common Functions**: Set of Common Functions for Reading and Writing from File IO that is used by all of the software components.

The Figure below shows the End to End Flow Diagram for the program. The Program is broken into 3 different stages: Introduction Stage, Processing Stage, and Finalization Stage. The Introduction Stage is where the Configuration File is validated and the Controller is constructed. The Processing Stage is where all the files are parsed and shared between the different threads which results in a set of Reduced Files. The last stage is the Finalization Stage which merges all the Reduced Files into a Single Output

File. Additionally, all the Controller class is destructed and the Socket Server is shutdown.



1.Identify all input files
2.Request Stub to Create Map Thread for a Given File(s) via Messaging
3.Map Threads Create Reduce Files and alerts Controller
4.Once all Input Files have been allocated this part of Stage 2 is done
- Launch Stubs (Start Socket Server)
- Read Config File
- Build/launch Worker Class (Connect to Servers)
1.Finalize class reads all Reduce Files and creates a Single Output
2.Write Success File.txt
Create Map Processing
Intro Setup
Create Reduce Processing
Finalize Stage
Config File
Input Files
R1 File
R2 File
R3 File
Output File
Success File
1.Create Socket Server
2.Wait for messages to be received from Map or Reduce Threads
3.If Map Message received create Reduce Thread.
4.If Reduce Thread Received track for creation.

Fig. 1: End to End Flow Diagram

Additionally, Figure 3 identicats how data is passed between the Threads/Processes via Sockets. Prior to the start of the program there should be a set of STUB processes that have socket servers waiting for the controller to connect to. At startup the Controller will create a Server Socket for the Map and Reduce Threads to communicate with. The IP Address and Port for the controller and stub sockets will be provided to the Controller via the Configuration File. The Controller will send a message to one of the Stubs to create a map or reduce thread. The created mapper and reducer threads will send messages back to the controller indicating its status as well as a message indicating the file it created.

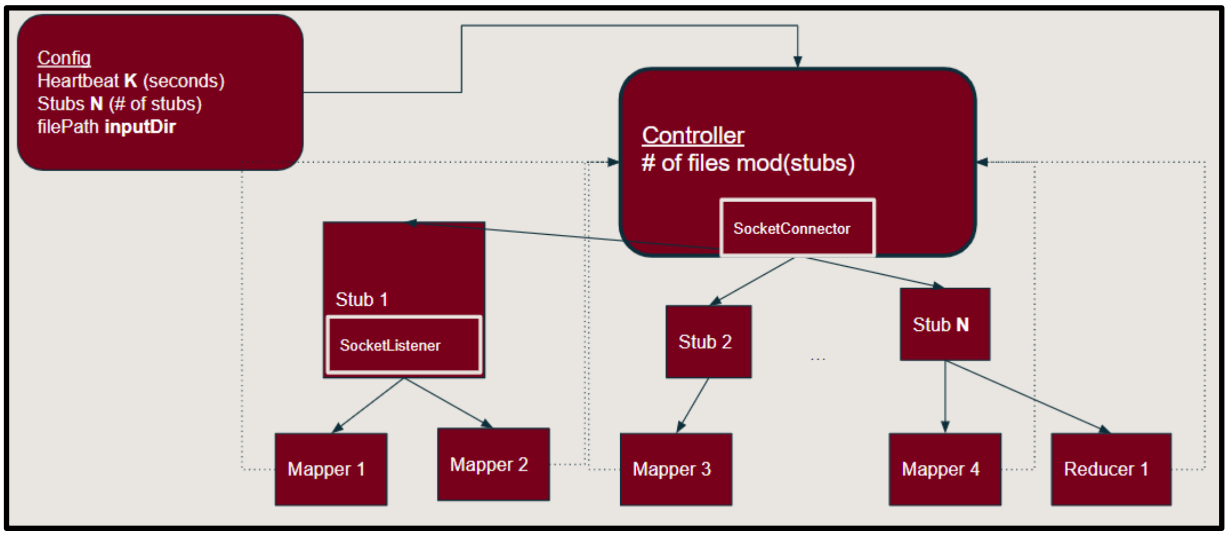


Fig 2: Basic Control Flow of MapReduce threads

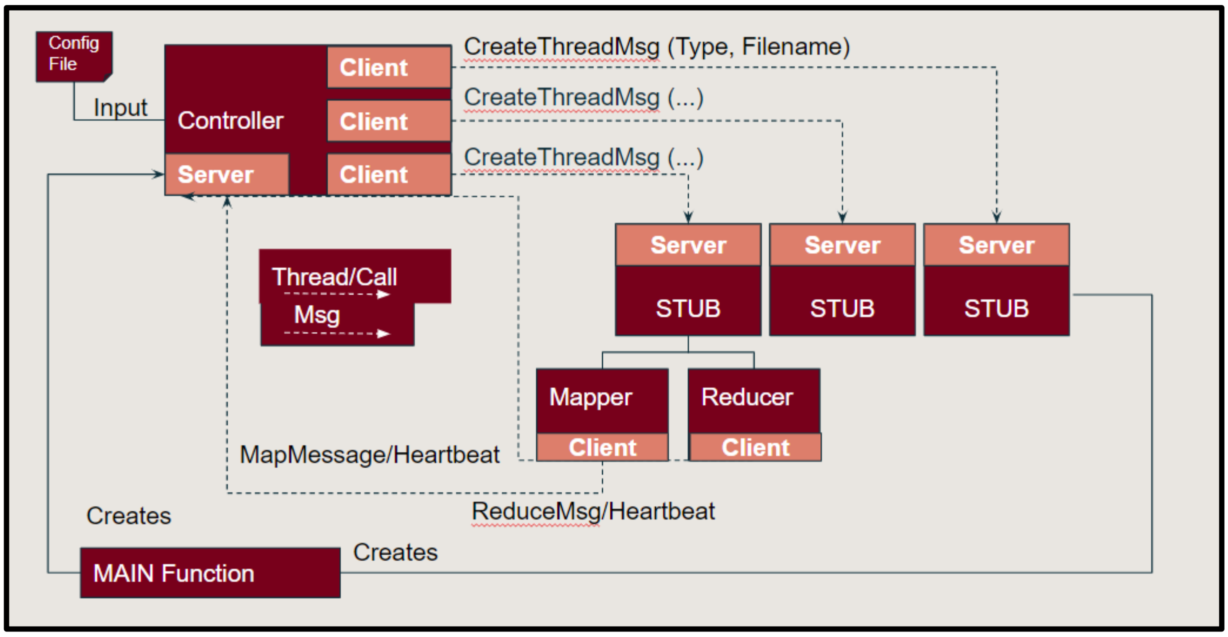


Fig 3: Detailed Subsystem Block Diagram of MapReduce

Design of Map Reducer software

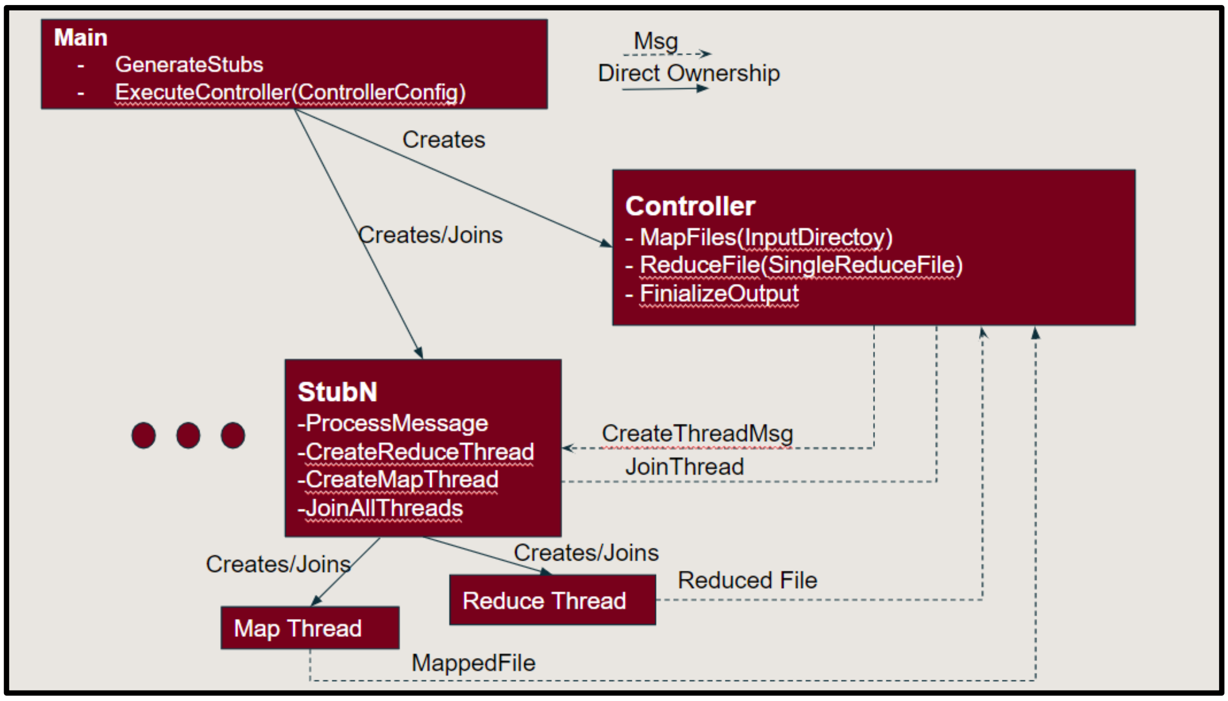


Fig 3: Workflow connection with the stubs and the controller