Design and Implementation of a Cloud-Based Big Data Programming Service

Yen-Yu Lin, Shu-Ming Chang, Su-Shien Ho, Ying-Ti Liao, Hsing-Chang Chou, Shih-Chang Chen, Xuan-Yi Lin, Jiazheng Zhou, and Yeh-Ching Chung

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan, R.O.C.
{raylin, shuming, sushien, eiteki, hchou, shcchen, xylin, jzhou}@srlab.cs.nthu.edu.tw
ychung@cs.nthu.edu.tw

Abstract—With the rise of cloud computing in recent years, more computing resources have been brought from enterprises to normal users. Cloud computing plays a major role in big data programming, and makes it feasible for everyone to deal with massive data. However, there are always some obstacles keeping unskilled users from utilizing the power of cloud computing. For example, the experts in statistics need to analyze big data, which heavily relies on computation power, but they may not necessarily possess the ability to build up reliable, flexible and scalable clusters. To overcome the gap, we address the challenges and design a user-friendly service to let users execute their program on website or from eclipse plugin. Considering to big data, we offer two storage services for different scale of datasets. Furthermore, we support not only certain kinds of programming system, such as MapReduce and MPI, but also more general execution environments and languages, including CUDA, Java and C. For design evaluation, we implement the service and prove that the architecture takes the advantages of cloud computing and it has ability to deal with big data programming.

Keywords—Big Data; Cloud computing

I. INTRODUCTION

Cloud computing leads a trend which spreads from industries to academics [1]. The concept of big data helps enterprise get better understanding of customers and establishes the great paradigm of data processing [2]. In industry, enterprises spend lots of cost to train IT staff to adapt the revolution of the cloudified big data architecture, and IT specialists may need to set up MPI or Hadoop cluster for different purposes. For academia, building up an execution environment with inadequate resources is relatively more difficult than that for industry. Therefore, the first barrier [3] between user and infrastructure needs to be solved before starting big data programming. This barrier can be caused by several reasons, such as the lack of physical machines, the unskillful server hosting, or security issues. In order to deal with this problem, one of the best solutions is finding an already hosted service to process big data. Unfortunately, existing services often provide only specific programming language or frameworks. Besides, operation over the terminal brings user another obstacle, which changes user’s coding experience and working style. Cloud computing lets computing resources available for all user. However, the mainstream framework keeps untrained users away from the land of big data. To solve this problem, we propose and implement a user-friendly interface which tries to help users to start from scratch easily. The users are not necessarily familiar with server administration.

Big data programming is a time consuming process. In traditional way, a program is asked to be written and compiled on server side. Take MapReduce computations [4] for example, developers often write programs with Eclipse. The execution environment, however, is restricted to a Hadoop cluster, which reduces the efficiency of development. Developers iteratively do the same task to modify programs, i.e., programs are edited before exporting to .jar file, and later transferred to remote server. Furthermore, big data programming is utilized in diverse domains in different languages or with different implementations. Most popular programming paradigms/languages, such as MPI or R [5] are also widely used. To support these types of programs, we implement a unified platform, which is able to execute various kinds of languages without any extra configuration.

In addition to the convenience of usage, security is also an critical issue to be addressed. Both industry and academia own their confidential big data. In order to protect the confidential data, users are faced with a difficult question that whether the platform is secure and trustworthy. As mentioned above, solving the technical problems to ensure safety are one of the main considerations in our service design. In the proposed service, we isolate users from terminal operations which provides advantages to both service providers and users. First of all, users will not notice what really happens in the back-end, while focusing on their works. Secondly, adding a layer of APIs between client and service makes sure that programs are in control. The features of our service also make architecture flexible, scalable and maintainable.

This paper is organized as follows. Section II presents the related work including architectures or products that provide development environment. In section III, we propose an architecture of our service and describe the details of each components. Besides, the usage of service from the perspective of user is also explained. Section IV presents the implementation and evaluation of the proposed service. Finally, the conclusions and future work are given in the section V.

II. RELATED WORK

The Apache Hadoop framework is by far the most-utilized platform for big data processing. It provides a highly-efficient, scalable, and fault-tolerant architecture for that programmers can store their data and perform computation operations upon the data [6]. This framework not only provides flexible and configurable settings, but also leverages and optimizes available hardware resources. Although Hadoop is the de-facto solution for big data processing, the progress of setting up a Hadoop cluster and configuring software services is time-consuming and labor-intensive.

In recent years, there are many advancements in the area of web-based online development environments. The user experience of programming has been changed since mobile devices and web applications are popular now. Koding [7] is an example which is a revolutionized web-based online development environment. Different from normal online editor, Koding gives users a virtualized server and several gigabytes personal storage that could be considered as a standalone and private machine. On the website, users can invite co-workers to join the shared project and collaborate simultaneously. However, every coin has two sides, this website returns immediate results after every changes of developer’s codes. On one side of this feature, developers get an debug-friendly playground and it increases programming productivity. On the other side, Koding is designed for dynamic language programming, which is not suitable for big data programming.

A product similar to Koding is called CoderPad [8], which supports more languages than Koding. It uses virtualization technique to offer users isolated operating systems. Although developers could not access terminal as they are accustomed to, developers are able to configure environment as they want. Coderpad supports not only dynamic languages but also static languages such as C or Java.

The services mentioned above both are designed to help users overcome the technical gaps and provide user-friendly development platforms. For big data programming, these kinds of services cannot fulfill basic needs in coding. Editor is the most important tool which affects the quality of development. The quality of current online editors are not good enough to replace a full-featured IDE like Eclipse. Besides, programs written in Java or R often utilize third-party libraries for specific purposes. Thus, for best development experience, developers usually do their work in local desktop environment.

III. SYSTEM DESIGN AND IMPLEMENTATION

Our service is designed as a middleware between infrastructure and client. The design of our service is to allow users to submit and run their big data programs as easy as possible. We anticipate users do all works at client side, and need not to handle anything in console mode. Therefore, accesses to terminal and storage are offered through APIs. A centralized OpenID service is used to authenticate user accounts throughout all components in the proposed system.

Fig. 1 shows the overview of system architecture. Section A introduces aspects for user-side environment, and describes the scenario of usage. Section B details the architecture of proposed service and its components. Section C describes the runtime environments.

A. Client-Side Environment

There are three components at the client side: a browser, an Eclipse plugin and a storage client. Both browser and Eclipse plugin play a similar role within whole service, and storage is the place which stores user’s programs and big data.

1) Storage

Our service provides two kinds of storage with different characteristics to store programs and data. One is SSBox, a Dropbox-like storage service, that gives user a sharable and portable directory synchronized to remote server. Another storage is used for storing pure data on Hadoop distributed file system (HDFS).

Users develop programs in the local SSBox directory and programs are immediately synchronized to the remote server through storage client. If a program requires small-scale data as input, the input dataset can directly save into the same directory. On the contrary, when users require a large-scale and high-performance storage, users can access personal HDFS [9] with Eclipse plugin. Because Eclipse plugin has not only job submission feature, but also an HDFS explorer to handle the operations on HDFS.

The proposed service collects all results into cloud storage after user’s program finished. An important mission of the storage client is to automatically synchronize output data from server side. The storage client checks any modification in server side, and synchronizes the changes.

2) Eclipse Plugin and Browser

The Eclipse plugin provides a graphical and user-friendly interface, which assists users launching programs easily. The plugin asks users to login before any operations. After login, users will be granted the permissions to access any components of the service. An authenticated user inputs necessary information, such as job type, application path/arguments, input path and output path, into the job submission form to submit a job. The paths are used to indicate locations of application and dataset in the storage.

In usage workflow, user use cloud storage directory as Eclipse work folder. Anytime when program be compiled, user can launch the program by this plugin. In the back-end, the plugin sends job information to service, and the service executes specified program to process specified input data on cloud storage. To deliver the same experience as use in terminal, at the client side, users can get the instant console messages redirected from server side.

Our service provides a web-based portal, and the functionality of the portal is designed similar to Eclipse plugin. Consequently, web browser is an alternative submission agent to submit jobs.
B. Job Manager

Job manager is the core of our service. This isolated component is wrapped by REST API, and is the only way to connect to computational resources, and storage. Job manager is in charge of internal communications, account authentication, and program launch. To developers, job manager establishes an exclusive tunnel to send real-time messages. To the internal components, job manager controls the concurrency of active jobs, file transfering, and global error handling.

1) REST API

The outer layer of job manager is wrapped by REST API and the APIs are open worldwide. In this paper, we implement the APIs to Eclipse plugin and web-based portal. This interface provides two major features. First feature is the universal account authentication mechanism [10] which utilizes OpenID and integrates to a LDAP server. Every OpenID is connected to a unique LDAP account, and a user will be granted full access permission if his account is authenticated successfully by our authentication API. The second feature is job submission. We design a general description format for diverse programs. Once users put programs on our storage, users can follow the description format and send a request to our REST API to trigger programs.

2) Website

As mentioned above, we address the recommended development environment Eclipse for MapReduce program. However, big data programming is not restricted to MapReduce and Hadoop framework. For example, some legacy programs written in C language are not feasible to port to MapReduce. Instead, MPI is a better choice to speed up C programs and increase utilization of distributed computing power [11]. We assume that MPI programs are not always edited on Eclipse, so our service provides another interface for those users who don’t install Eclipse.

In addition to supporting the same features as Eclipse plugin, X Window redirection function is implemented and given on website. For the programs using X library, we provide a personal display which won’t conflict with others. If users turn on X support, users are capable to receive real-time image redirected by service.

3) Workflow System

Workflow system is the core component of job manager. The system controls a series of processes for diverse program
executions. Once the workflow system accepts a job submission, the job information will be passed through profiler and the workflow system dispatches the job to different runner according to the loading.

\(\text{a) Profiler}\)

Jobs are described with certain format by user. When workflow receives a submission, profiler will analyze the application type, input size, input source and arguments, etc. Then, profiler generates temporarily environment variables for each job at runtime, and each job is standalone and can decrease the reciprocal effect. After job’s profile is established, the profiler will send job information to the certain program runner.

\(\text{b) Data Adapter}\)

Counter to storage client, data adapter retrieves the program and input data according to job settings. Our service provides two kinds of storage services. A cloud storage which synchronizes files to user’s local directory and the HDFS explorer. Because our service aims at dealing with big data but not general input dataset, we assume that the best storage to store data is HDFS. So, if users put their input data at local storage client, data adapter will attempt to retrieve corresponding directory from cloud storage and save to HDFS. In the other hand, if users already upload input data through HDFS explorer, data adapter will treat these data ready-to-run, and do nothing.

\(\text{c) Instant Message Redirector}\)

Instant console and running status are always the critical information to developer. Therefore, we implement a real-time messages communication mechanism. Once Eclipse plugin or website client connects to the server, it establishes a unique connection. Both two clients include the same library and use long-polling socket to communicate with the server. To present messages given in console, Eclipse shows every standard output from service in console perspective. In order to send real-time console, we design a messages redirection policy. Every client is asked to join a room named by job id. If the server finds out any messages from execution engine, workflow system will redirect the messages and broadcast in the room. The redirector manages all messages from the server, so we hide other unnecessary messages for client. With the message redirector, users can get the same experience as in terminal mode.

\(\text{d) Program Runner}\)

Fig. 1 shows three different runners within workflow system. Job submission is not executed before sending by profiler, so runner takes the responsibility for job execution. Each job type has its own running environment. Like generic runner, the runner is capable to compile user’s program and run the program with correct settings resolved by profiler. The settings include arguments, internal input path, internal output path, compilation options, etc. All types of runners have separated job queues. Owing to serve big data program, job queue is designed for leverage computation power and I/O loading. Once job started, program runner will generate few lines of commands according to settings, then execute the program.

e) Error Handler

Like normal workflow system, any stage could occur errors. To handle these errors/exceptions, we design an error handler as a global event listener. The error handler is not tend to replace the message redirector. Error handler is a standalone and fault-tolerant service, which keeps working even the message redirector is failed. Because of the isolation design of service, users only need to know whether their job is running or failed. Therefore, the job of error handler is to simply send an error message to users if error occurs, and trigger the post process to clean up job data.

\(\text{C. Runtime Environments}\)

Three clusters of different runtime environments are shown in the bottom of Fig. 1. To share the big data, we assume that the better storage for the data is HDFS. Therefore, the HDFS installed on the Hadoop cluster shares the file system to others by using NFS. Clusters mount the HDFS at the same path, then job runner can map the input paths to internal paths. Each virtual machine has the same storage settings; however, operations to files will raise permission errors. To overcome the authorization problem, an LDAP service is used to manage local accounts. Consequently, an OpenID account maps to a local account, and permissions of this account are granted by LDAP service. After users are authenticated and authorized, workflow system then gets the privilege to walk through clusters.

Hadoop cluster accepts only MapReduce programs, and GPU cluster with CUDA library installed can run CUDA programs. Generic cluster is designed for general-purpose jobs. In addition to supporting MPI, our generic cluster accepts customized script. We define some environment variables for users, and users are free to write their own script to compile programs, specify input data, run batch task and so on.

\(\text{IV. EVALUATION}\)

To evaluate our design, we have implemented and deployed the proposed service under the name of SSBDS, which is available as a part of the UniCloud project [12]. Currently there are dozens of users and hundreds of various big data computation jobs, including MapReduce, MPI and R programs have been carried out on the platform. As the part of evaluation, we prepare three types of programs.

\(\text{A. MapReduce Program with Eclipse Plugin}\)

First program is a MapReduce job that takes 160MB raw text as input data, and calculates the word counts.

![MapReduce application at local directory.](image)
Fig. 2 shows a Hadoop MapReduce program, which is placed at the synchronized directory of the cloud storage service, SSBox. In the local directory, the executable program is exported as .jar file. Like Dropbox client, the storage service synchronizes all files under the directory to storage server.

Our input data is ready and already uploaded to remote HDFS with Eclipse plugin. As figure 3 shows, we uploaded a directory named “input”, which include 3 text files. Like the file explorer, all operations are done by the plugin.

Users configure the submission settings in our plugin as shown in Fig. 4. The path of the program is the relative path in cloud storage.

Fig. 5 shows the messages redirector which redirects the console in real-time in console perspective.

In Fig. 6, after a job is completed, the output directory specified above is synchronized with the result data. Users can retrieve the result data without any HDFS commands.

Using desktop to submit a job and view result at local directory brings users a friendly environment, as they run the programs locally.

B. R Program with Service Website

Like Eclipse plugin, our website service provides similar user experience. In this section, we submit a R program via website with local input data. Besides, our R program uses X Window to draw charts, which are redirected by our service.

As shown in Fig. 7, we write a customized script and place R script at the same directory called “R”. To evaluate the other cloud storage functionality, we put small-scale input data at the “R_input” folder as shown in Fig. 8.

Fig. 7. The R application.
Fig. 8. R input data.
The submission shown in Fig. 9 requires the same fields as in Eclipse submission settings. We specify “R” as program path, and “R_input” as input path.

With clear interface, users can get the current stage of job and consoles as shown in Fig. 10.

Besides console redirection, our service redirects X Window to client through browser. As shown in Fig. 11, the chart drew by R application is forwarded to client.

V. CONCLUSIONS AND FUTURE WORK

As big data became popular in recent years, the demands of massive data processing increase. We notice that the curve of building up big data development environment is extreme steep. Therefore, we propose and implement the service aiming at providing a robust and friendly development environment. To store user’s massive data, we design two cloud storage spaces for different purposes that adapt the advantages of the distributed file system. We implement a storage client to synchronize user data to remote server automatically. Users can launch their programs since both programs and input data are located at our cloud. In order to send execution information, we design a message redirector to forward consoles to client. To provide user-friendly and isolated environment, we design different mechanisms to overcome the gaps, and try to satisfy most situations.

In the current stage of our service, we still have a long way to support various use cases. Without reaching computing servers, users cannot customize and modify settings to suit their own requirements. As mentioned in related work, virtualization technique can be another choice to build a personal virtual machine with our service installed. Secondly, our service to execute big data programs is not yet convenient for repetitive scientific jobs. Like for MapReduce jobs, a regular analysis model or programs would be used in several datasets with dozens of various arguments. Delegating execution cannot solve this kind of problem and save user’s time. Therefore, adapting description language like BPEL (business process execution language) as submission could obviously enhance user experience.

Moreover, we attend to show all the messages as possible as we could; however, our messages redirector cannot determine whether the message is necessary for user or not. A more powerful mechanism is needed for messages passing, to give user more comprehensive usage.

In the future, we plan to add more features to improve our service and keep refining the environment to suit everyone’s requirements.

REFERENCES

[6] Bhandarkar, M. MapReduce programming with apache Hadoop. in Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on. 2010. IEEE.