Dependency Management System with Hadoop Streaming for Data-analytic Projects

Lin LI[1,a)] Sozo INOUE[1,b)]

Abstract: In this paper, we propose a distributed parallel processing system for data-analytic project, which manages dependency among data and analytic programs, and re-execute updated programs and dependent programs for updated data/programs. In the system, a data analyzer can specify the dependency, parts for requiring distributed parallel processing using Hadoop Streaming, and they can be processed only for updated and dependent part, with flexibly selecting parallel or sequential execution. The specification can also specify multiple executions for the same program for different data as a simple statement, while their dependencies are checked separately.

1. Introduction

In the era of cloud computing, requirements for analyzing large-scale data, such as usage log and shared data which are collected on the server are increasing. We call a project to analyze such large-scale data with various numeric and statistical method *data analytic project*. In a data-analytic project, the process to analyze data has several stages, and each of them often generates intermediate data and/or files. At the same time, a data-analytic project often has modifications to programs for analysis, or addition of data files. In such a case, we need to re-execute the analytic programs. However, the data including intermediate data and analytic programs have dependencies among them as for the order of execution. If we could organize such dependencies and only re-execute the minimum part of the analytic process satisfying the dependencies, we can reduce the total time of the data-analytic project, since the program-execution time is not negligible in big-data-analytic projects.

In this paper, we propose a processing system for data-analytic programs in which the dependencies among programs and data are managed, and only the programs which need to be re-executed upon the dependencies are executed. Moreover, the system can be executed on parallel and distributed system with Hadoop Streaming[2]. The proposed system has the following features:

(1) An analyst describes the dependencies among analytic programs and data on the configuration file named ”Dakefile”. Dakefile can also specify the part of processing to be executed parallel processing using Hadoop Streaming.

(2) When the analyst execute the programs, s/he can select whether to force execute whole programs or only to execute obsolete/unexecuted programs from the view point of the dependencies.

(3) Dakefile can also describe a program to be applied to multiple data files repeatedly. When a new data file are added, the program can only run for the new data file without repeating from the first data file which has been already run.

(4) The analyst can also choose whether the execution is done in serial or in parallel when they execute Dakefile. For example, s/he can execute serially for while the data size is small, and change to parallel when it becomes larger.

(5) The system provides supplemental functionality to execute only mapper or reducer part in Hadoop, and provides libraries for I/O of statistical analysis program R[3].

2. Related Work

In this section, we introduce some existing systems and point out the challenges for data-analytic projects.

Make system [1] is a popular tool to manage the build of software. In a compiler program language, multiple source files need to be compiled, and the processing has several stages. However, in the software development, some of the source files are changed and rebuild occurs frequently. Make system is able to detect and re-execute only the compiling which is dependent to the changed files. It means that the differential compilation is possible.

In data-analytic projects, the function of Make is also useful. However, it is also necessary to understand the dependencies among the subjects (programs) of processing. For example, Make system is not intended that `gcc` compiler itself is modified and re-compiled. An analysis program, which corresponds to the compiler, is modified frequently in data-analytic project, so it is necessary to understand such dependencies. Moreover, an analytic program may also be applied repeatedly for different data. For example, it is the case that the same process is repeated for a multiple sensor data files which are added daily. It is difficult to apply Make system to specify such repeated processes in a united manner. Although it is possible to describe the process for each file extension in Make system, it is not possible to identify the processing content only by the extension in data-analytic project. In addition, Make does not have parallel and distributed processing feature by default, and is insufficient in the project of large-scale data analysis.

MapReduce is a concept that has been simplified for paral-

[1] Kyushu Institute of Technology, 1-1 Sensui-cho, Tobata, Kitakyushu, 804-8550, Japan
[a)] lilin19840702@gmail.com
[b)] sozo@mns.kyutech.ac.jp

lel distributed processing, which consists of three phases: Map, Shuffle and Reduce. The map phase takes input records and produces output of (key, value) pairs. This is followed by a shuffle phase that groups the (key, value) pairs by common values of the key, and finally a reduce phase takes all pairs for a given key and produces a new value for the same key. A developer solves problems automatically by only simply programming the Map and Reduce processing. MapReduce manages data placement and task scheduling for parallel distributed processing. Hadoop Streaming [4] is to provide a MapReduce using the standard I/O of the UNIX shell. It allows users to write Map and Reduce processing in a language other than Java. In other words, if the user uses the standard I/O, MapReduce can be implemented in any language. In Hadoop Streaming, a Map program writes to standard output a single line separated value and key of each data item by a tab character. Moreover, a Reduce program read from the standard input in the same format. The result is a text which is output of Reduce sorted by the keys.

In this paper, we realize to describe the dependencies mentioned above and to realize differential processing with Hadoop Streaming. By this, the efficiency of data analytic project with large-scale is expected. In addition, the analyst can choose whether the execution is done in serial or in parallel before executing. The throughput of Hadoop is known to be high, but when the data size is small, the response time becomes slower than serial processing. If we can choose to execute serial processing at runtime, flexible parallel processing depending on the data size is possible.

3. Dependency Management System for Data-analytic Projects

In this section, the data and analytic programs of data-analytic project is formulated by a directed graph, so we describe a system that can implement differential processing. Using the formulation, we can describe the dependencies, including the update status of analytic program as well as data, and at run time, also can check the dependencies individually while collectively define an analytic program repeatedly. In addition, we realize to execute only the unexecuted programs under the formulated dependencies. Moreover, you can specify which portion of the project is executed in parallel distributed. An analyst can also choose whether the execution is done in serial or in parallel at run time. It provides supplemental functionality to execute only mapper or reducer part in Hadoop, and also provides libraries for I/O of statistical analysis program R.

3.1 Formalization of data-analytic project

In this section, we formulate the description of the project in order to clarify data-analytic project in this paper. Here, the formulation must satisfy the following requirements.

- It must represent the dependencies among the subjects of processing, unlike the dependencies of compilation such as the Make system. For example, in Make, it is not necessary to manage the dependencies of gcc compiler. However, since trials and errors often occur also for the processing programs themselves, we need to manage these dependencies in the data-analytic projects.
- The repetition of the analysis program is depending on the case. It to do in serial or in parallel should be treated flexibly. For example, when describing the project, the same kind of analysis is summarized in a single program. Then during execution, it is required that only execute analytic program for additional data.

Based on the requirements above, the analysis graph G is defined to represent data-analytic Project.

$$G = (P, D, I, O)$$

P is a set of program case, D is a set of data, I is a set of input, and O is a set of output.

We define a set of programs corresponding to analytic programs.

$$\hat{P} = \{p_1, p_2, \cdots, p_{\hat{n}}\}$$

For example, if we take time windows for sensor data, and assume a program that calculates moving averages move_average and a program that calculates moving standard deviations move_sd, then $\hat{P} = \{\text{move\_sd}, \text{move\_sd}\}$.

In fact, each of analytic programs may run multiple times for different input data. For example like above move_sd, if it is performed for each data file data1.csv and data2.csv, there needs to be distinguished between the executions of the programs of two cases. Therefore, we define *program cases* P as the following.

$$P = \{p^1_{i_1}, p^2_{i_2}, \cdots, p^n_{i_n}\}$$

Here, $1 \leq i_1, i_2, \cdots, i_n \leq \hat{n}$, denotes which program in $\hat{P}$ the program case corresponds to. In other words, p^j_i is a running case of analytic program p_i. Note that G is a graph that contains a vertex p^j_i.

The data of project is defined in the data set D.

$$D = \{d_1, d_2, \cdots, d_m\}$$

Here, it includes intermediate data and final data, and is therefore necessary to be expressed flexibly when it is implemented so that the dynamically generated data can be managed.

Between the elements of the data set D and the program set P, a set of output edge is defined by O, and a set of input edge is defined by I.

$$I = \{(d, p) | d \in D, p \in P\}$$

$$O = \{(p, d) | p \in P, d \in D\}$$

In other words, the relationships between analytic program and input data are represented by I, and those between the analytic program and output data are represented by I. Let G have no loop, namely, for any $u \in P$, there is no path from a u to u other than the path consisting of only u.

Figure 1 shows an example of the formulation.

3.2 System overview

Figure 2 shows the components of the system. The part of Hadoop is the existing Hadoop Framework, and the part of user and system are the newly developed by us. The system has the

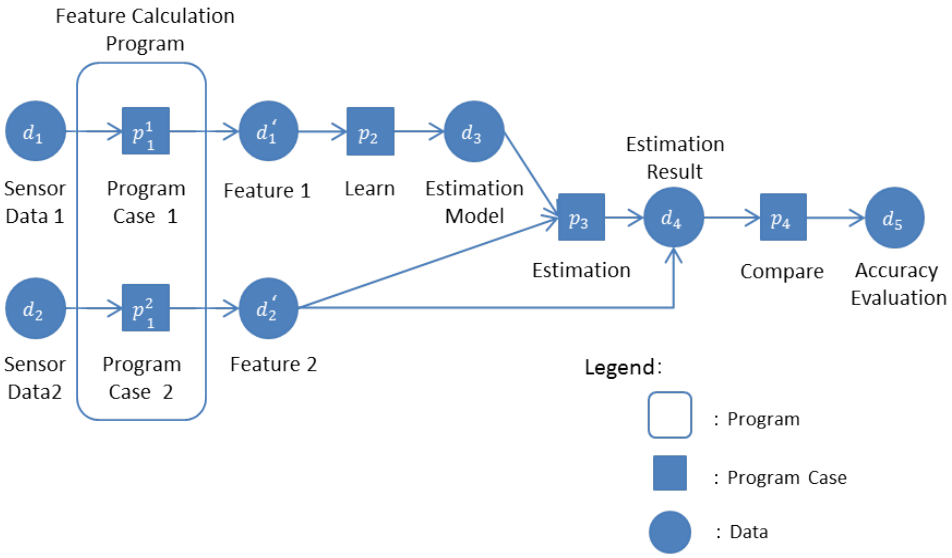

Fig. 1 Example of analysis graph G

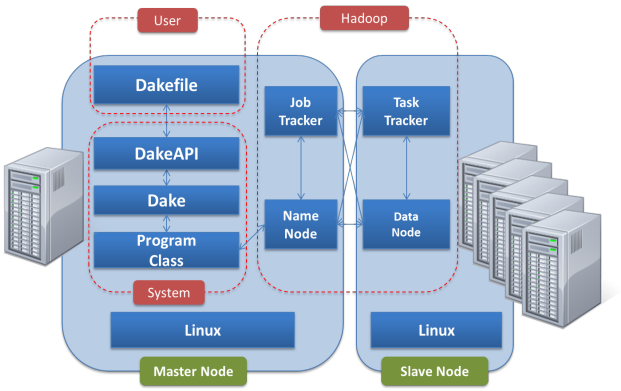

Fig. 2 System Components

following modules.

Dakefile Module: written by analysts and the analytic graph G is described in the Dakefile.rb file.

DakeAPI Module: the module to define the grammar of Dakefile module.

Dake Module: the main module in the system, interprets the Dakefile defined by analyst based on DakeAPI, and manage the object for analytic graph using the Program module introduced below. In addition, based on the command given by the command line arguments, it executes the analysis programs.

Program Module: defines an object-oriented class, such as the state variables and methods for each analytic program in the analytic graph G.

Default Mapper Module: a default mapper program when the mapper program is omitted in parallel and distributed processing in Dakefile.

Hadoop Streaming Module for R: functions to handle standard input and standard output, when Hadoop Streaming is used with the statistical analysis software R.

These modules are written by Ruby, where the flexibility of Ruby is utilized, such as invocation of dynamic method name or affinity for meta-programming.

We descripted each module in the following.

3.3 Dakefile

Dakefile described analytic graph G by users. It corresponding to makefile. The following is format of Dakefile.

```
Program "program name", ["program file name", ⋯ ] do |sources|
   Data ["input file name", ⋯ ] => ["output file name", ⋯ ]  do
         (Processing Content)
   end
end
```

or

```
Program "program name", ["program file name", ⋯ ] do |sources|
   Stream ["input file name", ⋯ ] => ["output file name", ⋯ ] do
          Mapper  "mapper program file name" (, number of parallel execution)
          Reducer "reducer "program file name" (, number of parallel execution)
   end
end
```

Any of the above can be repeated.

In either case, "Program Name" specifies the elements of the program set, and an array of "Program file name" specifies the name of the program file that the program is used at runtime, which are used for checking dependency when the program is running.

Also, if Data is specified, the array of "input file name" means the name of input data file, and the of "output file name" means the program cases. How to execute the analytic program is written in the (processing contents), and it uses the elements the array of "program file name" here.

If Stream is specified instead of Data, "input file name" and "output file name" are the same to above, and "mapper program file" and "reducer program file" mean to execute by parallel and distributed processing. Moreover, you may also specify the number of parallel execution in Mapper and Reducer. However, they cannot be always consistent when running in parallel, in which case we suggest setting 1 as the number of parallel execution.

In addition, since Dakefile is program written by Ruby , it can specify repeated definition of the analysis graph, such that multiple program cases are generated for one program. Thereby, it can

treat repeats of analytic programs flexibly either in individually or united.

In the line of Program, |sources| means the array of all input data file which depends on this program, and it can handle the newly created file in the preceding programs, which cannot be specified in the Dakefile otherwise.

Figure 1 is an example for a graph G, and the Dakefile is as the following:

Example of dakefile :

```
Program "Feature", ["Feature.rb"] do |sources|
   Stream ["SensorData1.csv", "SensorData2.csv"]=>["Feature1.csv", "Feature2.csv"] do
          Mapper  "TimeWindow.rb", 1
          Reducer "FeatureCalculation.rb"
   end
end

Program "Learn", ["Learn.R"] do |sources|
   Data  ["Feature1.csv"] => ["EstimationModelData"] do
         system("Learn.R")                #call as system command.
   end
end

Program "Estimation", ["Estimation.R"] do |sources|
   Data  ["EstimationModelData", "Feature2.csv"] => ["EstimationResultData"] do
         system("Estimation.R")           #call as system command.
   end
end

Program "comparative", ["comparative.rb"] do |sources|
   Data  ["EstimationResultData", "Feature2.csv"] => ["AccuracyEvaluationData"] do
         system("compare.rb")          #call as system command.
   end
end
```

In the example above, after feature calculation for sensor data 1 and 2, the former is used for machine learning, and the latter is used for the evaluation by feeding into the generated model after machine learning. Here, the part of feature calculation is using Hadoop Streaming.

3.4 DakeAPI module

The grammar of Dakefile is defined in DakeAPI module, and called from Dakefile, such as the methods of Program, Data, Stream, Mapper, and Reducer.

3.5 Dake module

Dake module is the main program in this system, it run the analytic program based on the given command by command-line arguments. The following is the format of run command.

```
> dake.rb (COMMAND) (local)
```

In the format, COMMAND is one of the options below:

- all (or empty): only re-execute unexecuted program by described graph G in Dakefile. The dependent program cases are executed first.
- all!: regardless of whether or not latest, all programs are executed.
- clean: remove any d in output edges $(p, d) \in O$. Namely, it removes all intermediate data and final data in this project.
- (program name): you can enter the name defined in Program or Stream in Dakefile.The program name identifies the program node in the graph G, and the node and its unexecuted ancestor are executed.
- (program name)!: force execute the program node identified by the program name and its ancestors.
- (program)-map: almost the same as above (program name), but if Stream is written in parallel distributed processing, only the mapper part which is described by Mapper is executed.

In the format, local can be omitted. If local is specified, the part is executed by local machine even if it is written by Stream as parallel distributed processing. In other words, the execution is done in serial on the master node.

3.6 Other modules

Other than the Dakefile module, DakeAPI module, and Dake module described above, we modules to implement the objects to define the properties and behaviors of programs and program cases. Among them, Program class is the class to define each program case, in which the properties of each program case such as related program/data files, and dependent program cases, and methods to be called when the program case is executed. Moreover, Stream class inherits Program class, and plays a role to be MapReduce program, which has properties to designate mapper program and reducer program in MapReduce architecture. Of course, Program class and Stream class have both methods to execute checking obsoleteness, and to force execute the program cases.

The system also provides supportive modules, such as DefaultMapper module which defines the default mapper program which divides the data by space and generates key and value data. Also, the system provides modules for default I/O functions for statistic software R, in which key and value data as standard input are converted to a data frame format in R, and a data frame object is output to standard output with the column named 'key' being key values.

4. Conclusion

In this paper, we proposed the dependency management system "Dake" for data-analytic projects, in which dependencies among data and analytic programs are managed, only the necessary programs are executed under the dependency, and a user can selectively adopt parallel and distributed processing with Hadoop Streaming dynamically.

Program cases for the same program can be executed in parallel, but program cases of different program are not parallelized for now even if they are independent each other. The future work is to optimize scheduling of program executions.

References

[1] Andrew Oram , Steve Talbott , "make", O ' REILLY,1997

[2] Tom White, "Hadoop", O ' REILLY, 2010

[3] R Development Core Team (2011). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

[4] Q. Ethan McCallum, Stephen Weston, "Parallel R", 2011

[5] Lin Li, Yuichi Hattori, Sozo Inoue, "Study of Version Management Method for Data-analytic Project",Proceeding of the 12th SOFT Kyushu Chapter Annual Conference, December 10, 2011, Saga, Japan.

[6] Lin Li, Hirotaka Hokazono, Yuichi Hattori, Sozo Inoue "Differential Processing for Data-analytic Projects with Parallel and Distributed Processing", Proceeding of Multimedia, Distributed, Cooperative and Mobile Symposium(DICOMO2012), pp. 7 pages,July 4, 2012, Ishikawa, Japan.