Taylor & Francis
Taylor & Francis Group

Applied Artificial Intelligence
An International Journal

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/uaai20

Modeling and Analysis of Hadoop MapReduce Systems for Big Data Using Petri Nets

Dai-Lun Chiang, Sheng-Kuan Wang, Yu-Ying Wang, Yi-Nan Lin, Tsang-Yen Hsieh, Cheng-Ying Yang, Victor R. L. Shen & Hung-Wei Ho

To cite this article: Dai-Lun Chiang, Sheng-Kuan Wang, Yu-Ying Wang, Yi-Nan Lin, Tsang-Yen Hsieh, Cheng-Ying Yang, Victor R. L. Shen & Hung-Wei Ho (2021) Modeling and Analysis of Hadoop MapReduce Systems for Big Data Using Petri Nets, Applied Artificial Intelligence, 35:1, 80-104, DOI: 10.1080/08839514.2020.1842111

To link to this article: https://doi.org/10.1080/08839514.2020.1842111

Published online: 14 Nov 2020.

APPLIED ARTIFICIAL INTELLIGENCE
2021, VOL. 35, NO. 1, 80–104
https://doi.org/10.1080/08839514.2020.1842111

Taylor & Francis
Taylor & Francis Group

Modeling and Analysis of Hadoop MapReduce Systems for Big Data Using Petri Nets

Dai-Lun Chiang[a], Sheng-Kuan Wang[b], Yu-Ying Wang[c], Yi-Nan Lin[d], Tsang-Yen Hsieh[d], Cheng-Ying Yang[e], Victor R. L. Shen[f,g], and Hung-Wei Ho[g]

[a]Financial Technology Applications Program, Ming Chuan University, Taoyuan City, Taiwan; [b]Department of Electrical Engineering, Ming Chi University of Technology, New Taipei City, Taiwan; [c]Department of Applied Foreign Languages, Jinwen University of Science and Technology, New Taipei, Taiwan; [d]Department of Electronic Engineering,Ming Chi University of Technology, New Taipei, Taiwan; [e]Department of Computer Science, University of Taipei, Taipei, Taiwan; [f,g]Department of Information Management, Chaoyang University of Technology, Taichung City, Taiwan; [g]Department of Computer Science and Information Engineering National Taipei University, New Taipei City, Taiwan

ABSTRACT

Information technological advances have significantly increased large volumes of corporate datasets, which have also created a wide range of business opportunities related to big data and cloud computing. Hadoop is a popular programming framework used for the setup of a cloud computing system. The MapReduce framework forms a core of the Hadoop program for parallel computing and its parallel framework can greatly increase the efficiency of big data analysis. This paper aims to adopt a Petri net (PN) to create a visual model of the MapReduce framework and to analyze its reachability property. We present a real big data analysis system to demonstrate the feasibility of the PN model, to describe the internal procedure of the MapReduce framework in detail, to list common errors and to propose an error prevention mechanism using the PN models in order to increase its efficiency in the system development.

ARTICLE HISTORY
Received 21 September 2019
Revised 11 May 2020
Accepted 21 October 2020

Introduction

Modern advances in computer technology and Internet system have led an increasing number of corporations and small businesses to setting up webpages and posting their information on computers. This has resulted in an enormous volume of datasets that continue to grow, which in turn spurred the development of concepts such as cloud computing and big data. At present, the research is focused on the development of business opportunities related to the analysis of big data. This kind of resource-heavy research has been made possible by the contributions of cloud computing made by corporations such as Google, Yahoo, and Apache (Jadhav, Pramod, and Ramanathan 2019; Valero 2018; White 2009).

Hadoop is a popular programming framework used for the setup of cloud computing systems. The MapReduce framework forms the core of the Hadoop

CONTACT Victor R. L. Shen rlshen@mail.ntpu.edu.tw Department of Computer Science and Information Engineering, National Taipei University, Taiwan

program for parallel computing. The Map function sorts datasets into <key, value> pairs that are then distributed to various nodes for parallel computing. The Reduce function collects the sorted datasets and yields the results. Because Hadoop is an open-source program, the system developer can rewrite a generation method of the <key, value> pairs, for sorting and sequencing of the data sets, and collecting and sequencing of the MapReduce framework. This requires that the system developer should have a comprehensive understanding of the MapReduce framework. In the absence of customization by the system developer, Hadoop uses its default settings. However, this can produce the results that were not anticipated by the system developer. This case underlines the importance of developing guidelines to help the developer construct the systems they envisage.

Despite its widespread applications, few researchers have verified whether the MapReduce framework is reachable or not. Rather, the majority of research work into MapReduce has focused on its space utilization and time efficiency. Computation tree logic (CTL) (Camilli et al. 2014) and linear temporal logic (LTL) (Hallé and Soucy-Boivin 2015), for example, are two analysis approaches that use mathematical models to track the conditional transitions for system verification.

This study employs a Petri net model to verify the reachability of the MapReduce framework and to assist the system developer in developing parallel MapReduce systems. Petri net theory was developed by Dr. Carl Adam Petri at the University of Bonn in Germany in 1962. A Petri net is a mathematical and graphical tool which is widely used in modeling and simulating system behaviors under various circumstances. It comprises places, transitions, arcs, and tokens; and offers a module of expressing a system which is concurrent, asynchronous, distributed, parallel, nondeterministic, or stochastic (Mazhar Rathore et al. 2018). In the system simulation, a high correlation has been demonstrated to exist between liveness and deadlocks, which can be tested and analyzed using the attributes of Petri nets. Petri nets provide a visually interactive tool (Wu and Zhou 2010) and have been widely applied in a number of fields.

The use of Petri nets benefits the modeling and analysis of the MapReduce framework because Petri nets are well suited to the description of parallel computer models. They can also be represented by stringent mathematical expressions as well as intuitive graphical expressions. The complete representation of the MapReduce framework, the system development examples, and the common errors made by this study will hopefully be of great help to the system developer in the development of parallel MapReduce systems.

The remainder of this study is organized as follows. The origin of the MapReduce framework in Hadoop, the structure of the Petri net model and its attributes are explained in Section 2. In Section 3, the Petri net model is presented to explain how the MapReduce framework can assist us in the system development. In Section 4, the common errors that arise in the development of

parallel MapReduce systems and the means of preventing them are listed. The proposed approach is also compared with other common analysis methods. Section 5 outlines the contributions of this study and provides the system developer with a standard model with a low error rate.

Literature Review

In this section, we present a detailed introduction of Hadoop and the basic components with properties of a Petri net. Then, simple MapReduce programs are used to conduct reachability tests, and CLT as well as LTL, two common analysis methods, are presented. Finally, WoPeD (Workflow Petri Net Designer), the Petri net software tool, is introduced.

Framework of Hadoop MapReduce

Apache Hadoop is an open-source programming framework for distributed data-intensive applications implemented using Google's MapReduce framework and Google File System. It can decompose distributed data sets and applications into multiple smaller tasks and then delegates them to various nodes for parallel processing (White 2009).

Hadoop uses the Hadoop Distributed File System (HDFS) to store data sets. The purpose of the HDFS is to develop a distributed file system that can be constructed using commercially available hardware. The HDFS features scalability, cost efficiency, flexibility, and fault tolerance.

MapReduce is a distributed processing library that enables applications to be written for easy adaptation to parallel execution by decomposing the entire job into a set of independent tasks. An application written using the MapReduce library is organized as a set of independent tasks which can be executed in parallel (Neil, Gunther, and Tomasette 2014).

The framework of MapReduce has been implemented using the one published by Google. In the Map phase, Hadoop converts the input data into <key, value> pairs that are easy to manage. In the Reduce phase, its function is to collect all the <key, value> pairs for a specific key and transform them into a new <key, value> pairs, where the value of a key is the specific one. Their value is in a list [$value_1$, $value_2$, etc.] of all the values that are <key_1, [$value_1$, $value_2$, etc.]> pairs whose key is the specific one across the entire input data sets (Neil, Gunther, and Tomasette 2014). By managing the Map and Reduce frameworks, the system developer can control all of the distributed processing procedure in a system (Valero 2018). A flowchart of MapReduce is depicted in Figure 1.

The key points that have earned the MapReduce framework of Hadoop praise and success include the following items:

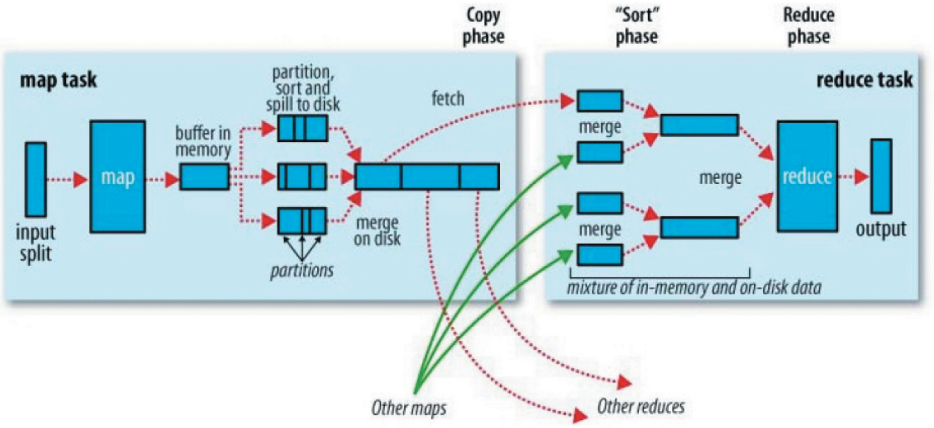

Figure 1. Flowchart of MapReduce (White 2009).

- It is easy to grasp, even for software engineers without experience in distributed systems.
- Since Hadoop is an open-source framework, the system developer can customize its contents, such as <key, value> pairs generation method, as well as data distribution, thereby gaining greater control over their designs as well as greater flexibility.
- MapReduce can process a wide variety of problems, such as Google searches and data mining.
- The Hadoop system is highly scalable; i.e., it can comprise thousands of computers.

Petri Nets

An ordinary Petri net is described by using a four-variable network (Mazhar Rathore et al. 2018; Wu and Zhou 2010): $PN = (P, T, A, M_0)$, where P denotes a set of places, T denotes a set of transitions, A denotes a set of arcs, and M_0 denotes an initial marking. The places in a Petri net comprise a finite set:

$$P = \{p_1, p_2, \cdots, p_n\},\ n > 0.$$

Generally, places are depicted with circles to signify the status of an object in the system. The presence of a token on a place means that this condition exists in the system. The transitions comprise a finite set:

$$T = \{t_1, t_2, \cdots, t_m\},\ m > 0.$$

Transitions are depicted with bars and indicate that the event described changes the state of the system. At the same time, these transitions also indicate the start and the termination of the system procedure. A transition fires if it meets the conditions such as weight, whereupon a token transits from

its input Place i to its output Place j, which implies that the state of the system has been changed (Natesan et al. 2017; Xiong, Fan, and Zhou 2010).

An arc describes the correlations between places and transitions. It runs in a single direction and shows the direction of the token's transition. A weight function $w(p_i,t_j)$ can be added to an arc to show the weight it needs to transit from Place i to Place j. Initial Marking M_0 denotes a set of tokens in the system. When a set of tokens changes, it indicates that the transition is fired. It means that the happening or action of an event, i.e. the dynamic behavior, is described in the Petri net.

A fundamental Petri net is used to describe the state change of a discrete event dynamic system, where a complete Petri net model includes places, transitions, arcs; and the flow of tokens are denoted as the state change of a dynamic system (Matsuzaki 2017; Saisai et al. 2016; Zhang, Zhang, and Ya 2010; Birzhandi et al. 2019). The basic Petri net model is depicted in Figure 2.

In Figure 2, there are three places, two transitions, four arcs including two input arcs and two output arcs, two tokens, and one Petri net marking. This basic model is denoted as

$$P = \{p_1, p_2, p_3\}$$

$$T = \{t_1, t_2\}$$

$$A = \{(p_1, t_1), (p_1, t_2), (t_1, p_2), (t_2, p_3)\}$$

$$M = [M(p_1), M(p_2), M(p_3)] = [2, 0, 0]$$

$$w(p_1, t_1) = 3,\ w(p_1, t_2) = 1,\ w(t_1, p_2) = 1,\ w(t_1, p_3) = 1.$$

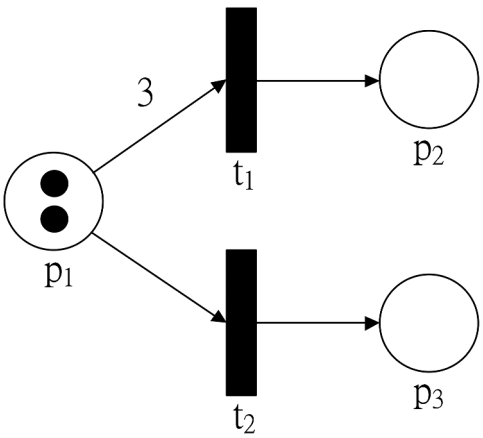

Figure 2. Basic Petri net model.

The transition of tokens in Petri nets is capable of modeling complex behaviors including various restrictions and parameters. As a result, Petri nets are widely applied in various fields.

Properties of Petri Nets

The properties of standard Petri net models can be divided into behavioral and structural properties (Mazhar Rathore et al. 2018). The former refer to the initial marking functions of Petri nets, whereas the latter describe the structures of Petri nets. Furthermore, our Petri net model can be described by using the following property.

Reachability

If the firing begins from M_0, all reachable places will be derived after a series of firing transitions. Thus, the Petri net will be viewed as reachability. This property is mainly used to determine whether the operations of a work procedure are feasible or not.

Verification equation: $M_i = A^T \cdot X + M_0$

In the above equation, A is the incidence matrix of a Petri net, M_0 is the initial marking of the model, and the M_i marking is the verification goal, where i denotes any marking of the Petri net. Using this equation, we can obtain the transition vector X that enables the marking to be reachable from M_0. If all of the markings in the Petri net can be reachable from M_0, then the Petri net has reachability.

Incidence Matrix

In terms of analysis, a Petri net is a mathematical and graphical tool that has a greater mathematical ability to describe algebraic equations using matrices. A Petri net can be replaced with an incidence matrix A, which is defined as output flow matrix A^+ minus input flow matrix A^-. The former one is an $n \times m$ matrix with n and m denoting the numbers of places and transitions, respectively. Each incidence matrix indicates the relationship between a place and a transition, defined as

Input Flow Matrix: $A^- = [a_{ij}^-]$, $a_{ij}^- = I(p_i, t_j)$
Output Flow Matrix: $A^+ = [a_{ij}^+]$, $a_{ij}^+ = O(t_i, P_j)$
and $a_{ij}^+ = w(i,j)$, $a_{ij}^- = w(j,i)$.
The incidence matrix is $A = A^+ - A^-$,

where a_{ij}^+ is the weight of the arc from transition i to place j; while a_{ij}^- is the weight of the arc from place i to transition j. If the place and the transition have no correlation, both a_{ij}^+ and a_{ij}^- are 0s.

Petri Net Model of MapReduce Framework

We convert the MapReduce framework into a Petri net model according to (Wu and Zhou 2010). This MapReduce model is a basic example describing the operations of the MapReduce framework. Below is the algorithm, and the Petri net model is depicted in Figure 3. We assume that this model has two nodes for the sub-jobs after a split.

Algorithm: MapReduce Framework	
INPUT:	The data a user wants to analyze.
OUTPUT:	The data analysis results after MapReduce.
PROCEDURE:	
Step 1:	Start MapReduce program.
Step 2:	The user inputs the data.
Step 3:	The mapper begins the map function to generate <key, value> pairs.
Step 4:	According to <key, value> pairs, cut the job into many small sub-jobs, and then split to every node.
Step 5:	Part 1 and Part 2 process their sub-jobs.
Step 6:	Merge all data sets from every node to the Reducer and perform the reduce function.
Step 7:	Output the data after being reduced.

Reachability Analysis Using Petri Net Model

We convert the visual Petri net model in Figure 3 into a mathematical matrix form and use the mathematical model for reachability verification. $P_0 \sim P_7$ and $t_0 \sim t_5$ denote the places and transitions, respectively.

Notations: P_0 denotes Main, P_1 denotes Mapper, P_2 denotes Part 1, P_3 denotes Part 2, P_4 denotes Processed Data 1, P_5 denotes Processed Data 2, P_6 denotes Reducer, and P_7 denotes Output. t_0 denotes Input, t_1 denotes Split, t_2 denotes Process, t_3 denotes Process, t_4 denotes Merge, and t_5 denotes Reduce.

$$A = \text{Incidence Matrix} =$$

$$\begin{array}{c} \\ t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{array} \begin{array}{c} \begin{array}{cccccccc} P_0 & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 \end{array} \\ \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} \end{array}$$

Firing Sequence: t_0, t_1, t_2, t_3, t_4, t_5

$M_0 = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$

t_0

$M_1 = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$

t_1

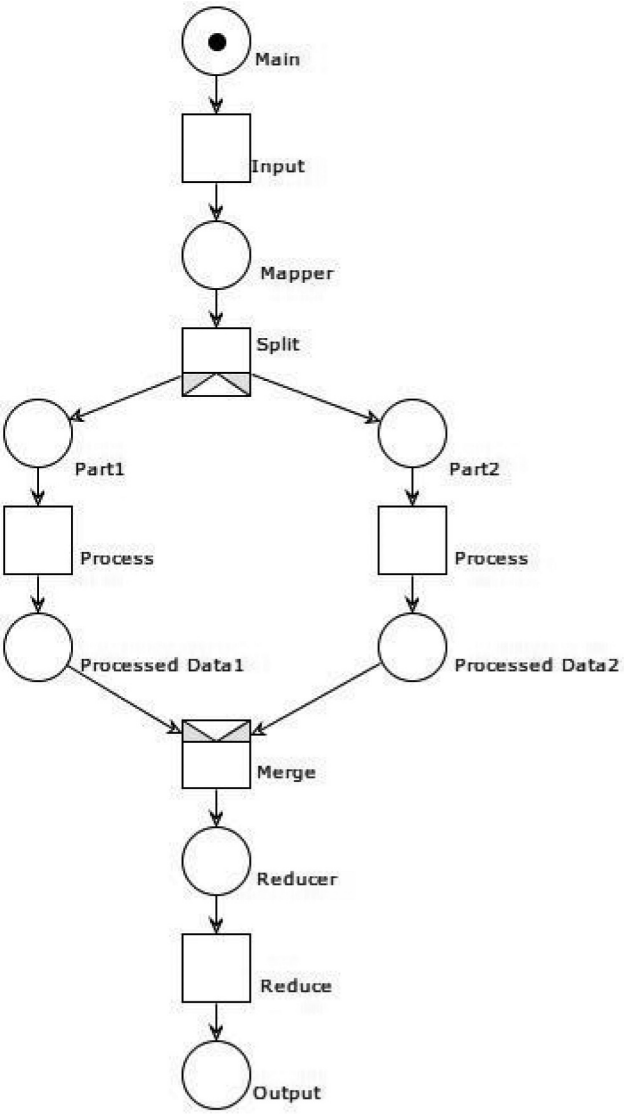

Figure 3. MapReduce Petri net model.

$$M_2 = [0\ 0\ 1\ 1\ 0\ 0\ 0\ 0] \xrightarrow{t_2} M_3 = [0\ 0\ 0\ 1\ 1\ 0\ 0\ 0] \xrightarrow{t_3} M_4 = [0\ 0\ 0\ 0\ 1\ 1\ 0\ 0] \xrightarrow{t_4} M_5 = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0] \xrightarrow{t_5} M_6 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]$$

This presents the state changes in the Petri net model after various transitions are fired. In the next step, we start to verify the reachability of the Petri net.

$$M_6 = A^T \cdot X + M_0$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

From the above equation, we get $t_0 = 1$, $t_1 = 1$, $t_2 = 1$, $t_3 = 1$, $t_4 = 1$, $t_5 = 1$. $M_6 = A^T \cdot X + M_0$, such that marking M_6 can be reachable from marking M_0.

Software Tool – WoPeD

WoPeD (Freytag and Sänger 2014) is an open-source software system developed at Cooperative State University Karlsruhe under the GNU Lesser General Public License (LGPL) for the creation and analysis of Petri net models. The goal of WoPeD is to create a user interface that is easy to use for analysis, modeling, and simulation. The developed interface is depicted in Figure 4. WoPeD is the ideal Petri net analysis software for research personnel, professors, and students due to its user-friendly operation interface and easy-to-use procedure for checking the model vulnerabilities. It is widely applied all over the world and has been successfully used in numerous lectures and student projects.

The functions of WoPeD include the following:

- Process and Resource Editor
- PNML Compliance and Import/Export Formats
- Soundness Checker
- Interactive Simulator and Coverability Graph Visualization
- Quantitative Simulation and Capacity Planning
- Quantitative Simulation and Capacity Planning
- Research Support Tool
- AProMoRe Repository Front-End

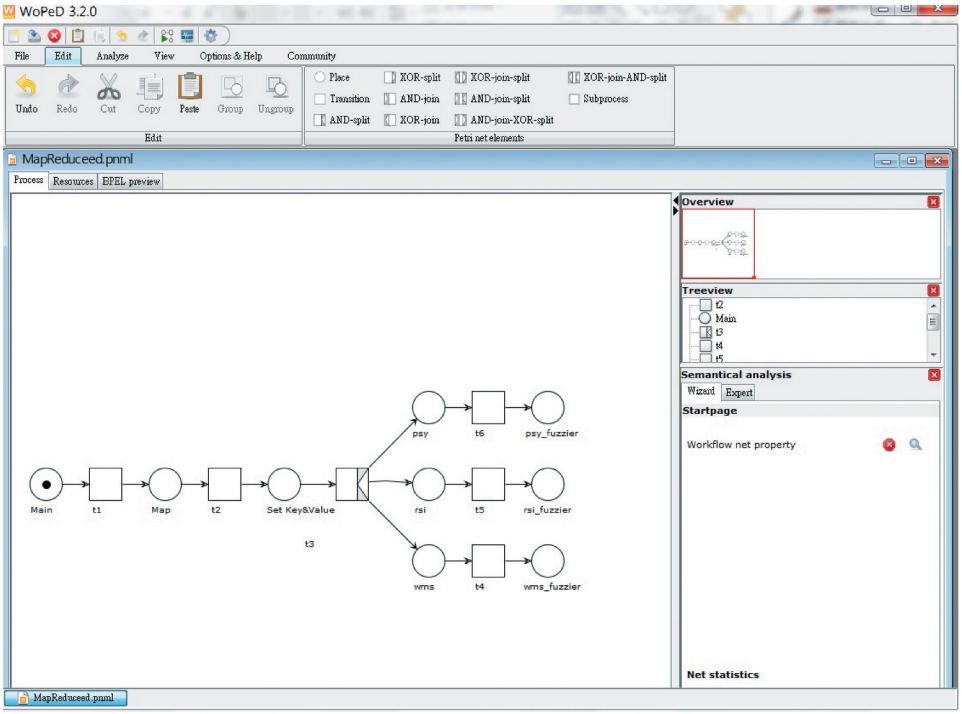

Figure 4. Screenshot of WoPeD.

The Proposed Approach

This study is affiliated with the research project sponsored by the Ministry of Science and Technology (MOST), Taiwan; and entitled Cloud-Based Learning and Analysis System for Big-Data. Below, we apply the Petri nets to model and analyze this project.

The purpose of the project is to enable users to customarily analyze the selected datasets. This includes rewriting the methods of <key, value> pairs generation and data distribution as well as adding check mechanisms to ensure that the results can be the same as anticipated. The system operating procedure is listed as follows: input file; check file; check rules; conduct mapping; perform the fuzzification, fuzzy inference, defuzzification, and reduction of the integrated data in the various nodes; produce results; and output results. We further explain the system development processes to illustrate the benefits of the proposed models.

Parallel MapReduce System for Big-Data Analysis

We conduct more thorough system development using the basic MapReduce framework in Figure 3. During the analysis, the data sets must be fuzzified before the analysis calculations are performed by using fuzzy inference. The

defuzzified data sets are then sent to the Reduce function for integration, which then yields the final results. Below is the algorithm, and the Petri net model is depicted in Figure 5.

Algorithm: MapReduce	
INPUT:	The data a user wants to analyze.
OUTPUT:	The data analysis results after MapReduce.
PROCEDURE:	
Step 1:	Start MapReduce program.
Step 2:	The user inputs the data.
Step 3:	The mapper function begins the map function to generate <key, value> pairs.
Step 4:	According to <key, value> pairs, cut the job into many small sub-jobs, and then split to every node.
Step 5:	The nodes fuzzify the data.
Step 6:	The nodes analyze the data using fuzzy inference.
Step 7:	The nodes defuzzify the data.
Step 8:	Merge all data from every node to the Reducer doing reduce function.
Step 9:	Output the data after being reduced.

Detailed Procedure for Parallel MapReduce Framework

This system requires the customized methods for <key, value> pairs generation and data distribution, thereby necessitating a thorough analysis of the MapReduce framework. The algorithm presented below is converted into the Petri net model for the MOST-project as depicted in Figure 6.

As can be seen, the Map framework includes an input format function to normalize the input data, a record reader to set the <key, value> pairs generation method, and a partitioner to determine the means of data distribution.

The Reduce framework contains the following steps: a shuffle function to compile the data, a sorting function to arrange the compiled data in alphabetic order, a reduce function for integration and simplification, and an output format function to normalize the data sets and to yield the final results.

Algorithm: Map	
INPUT:	The data that a user wants to analyze
OUTPUT:	Separate sub-jobs distributed to the various nodes
PROCEDURE:	
Step 1:	The mapper function begins the input format function to normalize the input data sets.
Step 2:	The formatted data sets are sent to the record reader to generate <key, value> pairs.
Step 3:	The partitioner divides the data sets and distributes them to the nodes in the system.

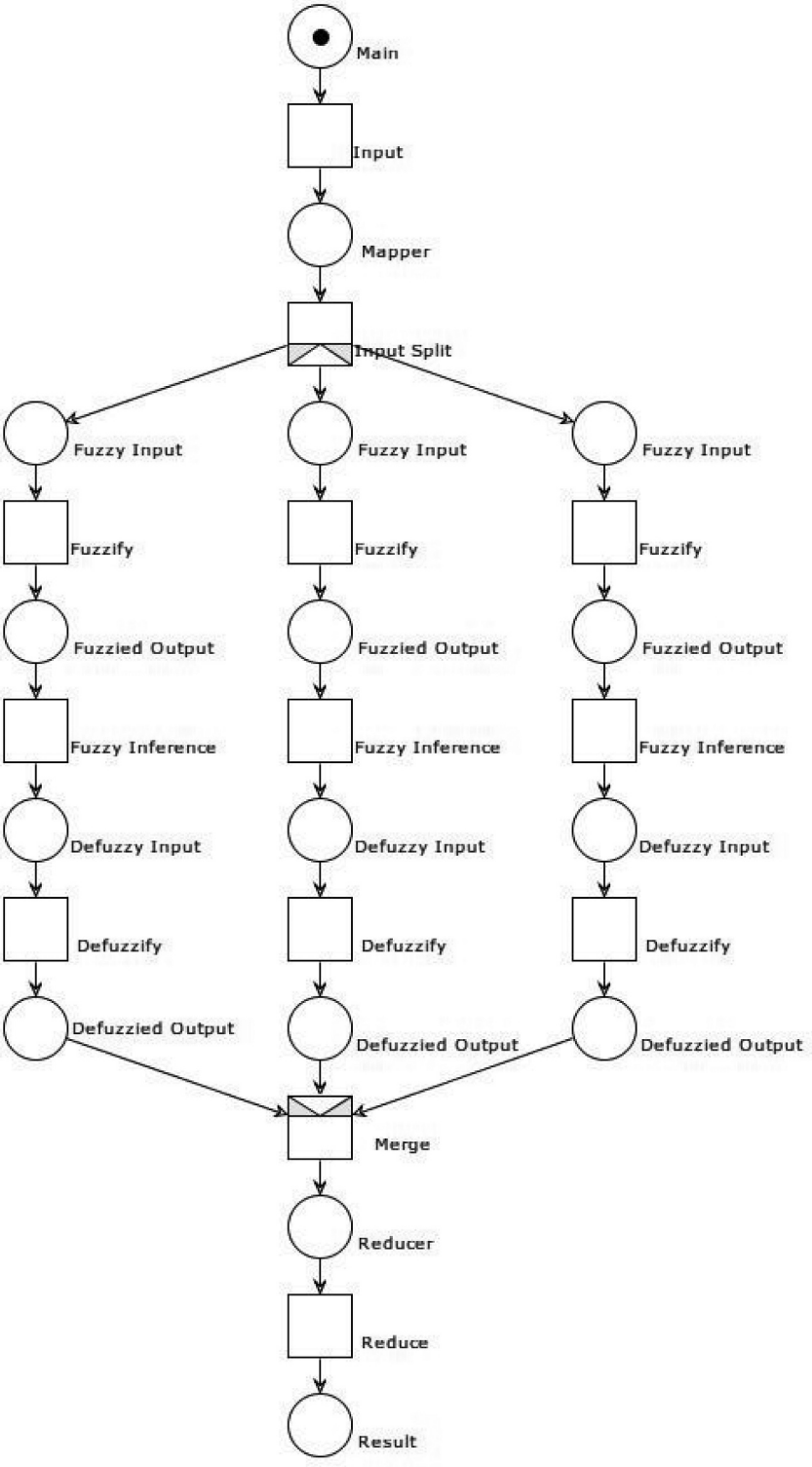

Figure 5. Petri net model of parallel MapReduce framework.

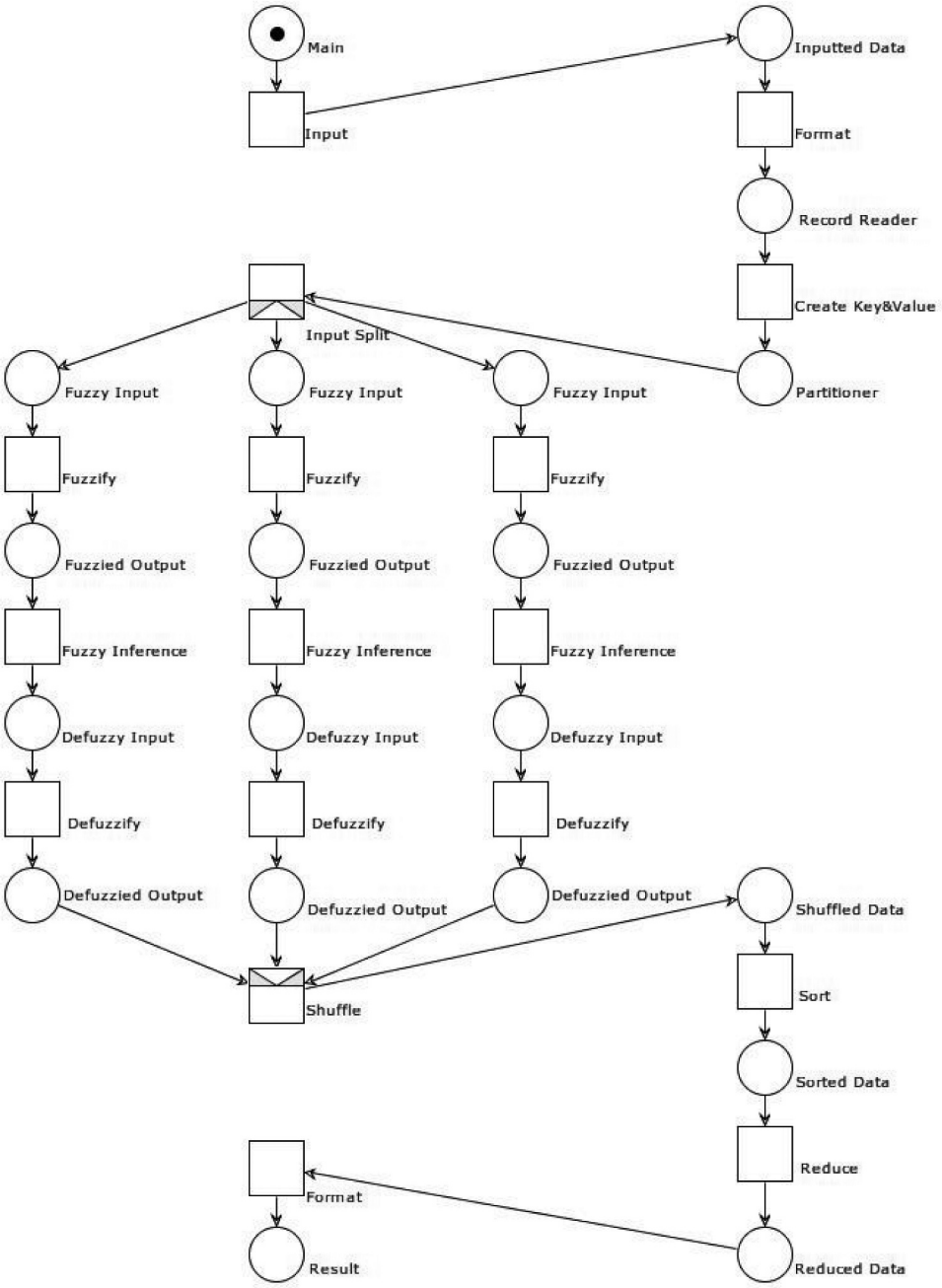

Figure 6. Internal structure of parallel MapReduce framework.

Inclusion of Check Procedure in the System

To prevent the system errors and to ensure that users have not input interdependent or contradictory rules' analysis, which will lead to errors in <key, value> pairs generation method, we add a file checker and a rule checker to the system.

Algorithm: Reduce	
INPUT:	The data resulting from the Map operation
OUTPUT:	The data analysis results yielded by the Reduce function
PROCEDURE:	
Step 1:	The nodes execute the shuffle function to compile the defuzzified datasets.
Step 2:	The sorting function rearranges the compiled data sets.
Step 3:	The data are sent to the reduce function for integration and simplification.
Step 4:	The reduced data are sent to the output format function for normalization.
Step 5:	The nodes execute the shuffle function to compile the defuzzified datasets.

File Checker

The purpose of the file checker is to ascertain whether the system contains the data uploaded by the user or not. During this execution, it ensures the following two cases:

Users have uploaded the data sets.

An output folder does not exist before the execution of the program. If it exists, the HDFS will not know where to put the results, and the Hadoop system will not run the program.

Below is the file-checker algorithm, and the Petri net model is depicted in Figure 7.

Algorithm: File Checker	
INPUT:	The data that a user wants to analyze
OUTPUT:	The data check results
PROCEDURE:	
Step 1:	The file checker is executed.
Step 2:	The system is searching for the uploaded data that the user wants to analyze.
Step 3:	If the data exist, then the next task begins; if not, then the system returns to its initial state.

Rule Checker

The rule checker examines each of the rules input by users for interdependence or contradiction. For example, consider two rules below:

(1) If *a* is *b* then *c* is *a*.
(2) If *a* is *b* then *c* is not *a*.

These rules are conflicting and will lead to abnormalities in the system analysis and therefore to the unanticipated results. They might also cause unnecessary operations that will waste system resources and increase execution time.

If the rules are independent but placed in the wrong order, errors will arise in the results. For example, consider three rules below:

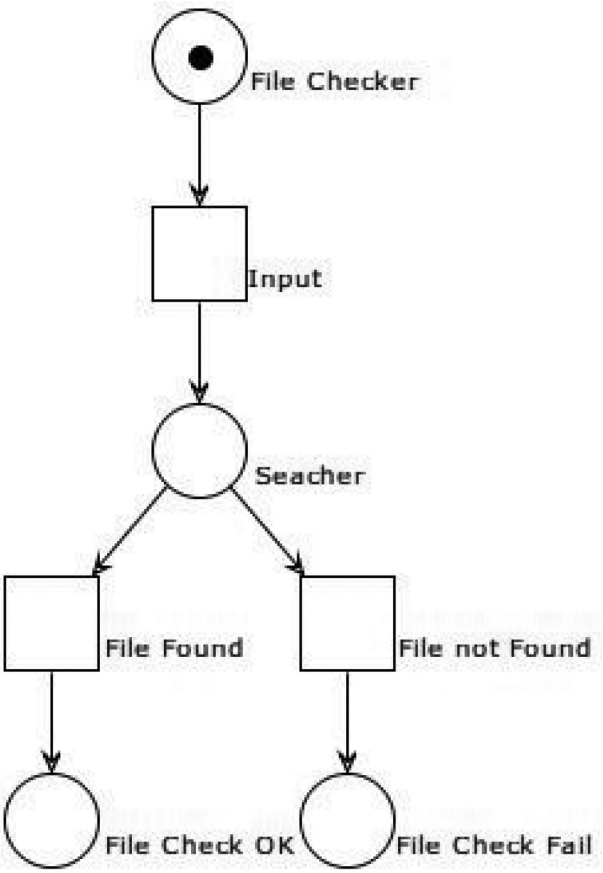

Figure 7. File checker.

(1) If c is d then e is f.
(2) If e is f then g is h.
(3) If a is b then c is d.

Users intend to obtain the following rule: "if e is f then g is h." However, problems in the order of execution may lead to problems in the analysis or the results. Below is the rule-checker algorithm, and the Petri net model is depicted in Figure 8.

Algorithm: Rule Checker	
INPUT:	The analysis rules input by the user
OUTPUT:	The rule analysis results
PROCEDURE:	
Step 1:	The rule checker is executed.
Step 2:	Each of the rules input by the user is read in.
Step 3:	The rules are checked for dependence or contradiction.
Step 4:	If neither of these exists, then the next task begins; if dependence or contradictions exist, then the system returns to its initial state.

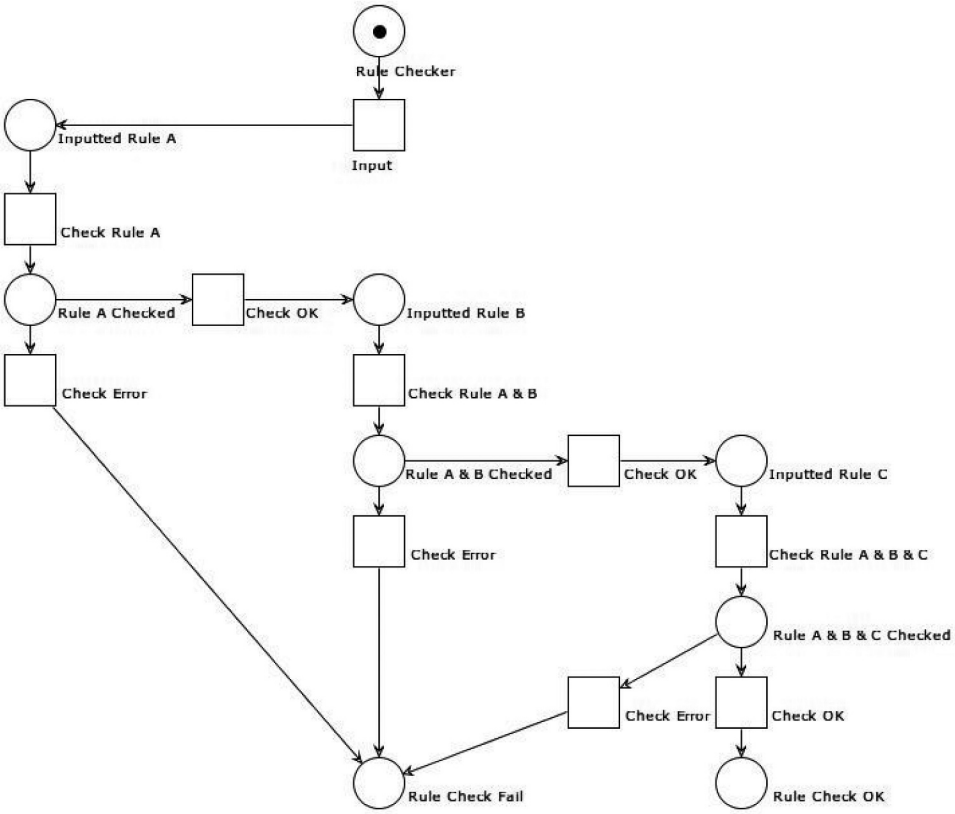

Figure 8. Rule checker.

Big Data Analysis System by Inclusion of Checking Mechanism

The addition of the file checker and the rule checker in the system model builds a big-data analysis system with checking mechanism. The algorithm is presented below, and the Petri net model is depicted in Figure 9. This complete MapReduce system analysis ensures that the model fulfills the requirements of reachability and has liveness, i.e. no deadlocks. The file checker and the rule checker prevent errors and the unexpected results from happening. At the same time, this complete model presents guidelines for the system developer so that fewer errors are made due to minor details, which increases the efficiency of system development.

Algorithm: Big-data analysis system with checking mechanism	
INPUT:	The data that a user wants to analyze
OUTPUT:	The data analysis results after MapReduce
PROCEDURE:	
Step 1:	Start MapReduce program.
Step 2:	The user inputs the data.
Step 3:	The file checker is executed.

(*Continued*)

(Continued).

Algorithm: Big-data analysis system with checking mechanism	
Step 4:	The system is searching for the uploaded data that the user wants to analyze.
Step 5:	If the data exist, then the next task begins; if not, then the system returns to its initial state.
Step 6:	The rule checker is executed.
Step 7:	Each of the rules input by the user is read in.
Step 8:	The rules are checked for dependence or contradiction.
Step 9:	If neither of these exists, then the next task begins; if dependence or contradictions exist, then the system returns to its initial state.
Step 10:	The mapper function begins the input format function to normalize the input data.
Step 11:	The formatted data are sent to the record reader to generate <key, value> pairs.
Step 12:	The partitioner divides the data and distributes them to the nodes in the system.
Step 13:	The nodes fuzzify the data.
Step 14:	The nodes analyze the data using fuzzy inference.
Step 15:	The nodes defuzzify the data.
Step 16:	The nodes execute the shuffle function to compile the defuzzified data.
Step 17:	The sorting function rearranges the compiled data.
Step 18:	The data are sent to the reduce function for integration and simplification.
Step 19:	The reduced data are sent to the output format function for normalization.
Step 20:	The normalized data yield the results.

Main Results

In Section 4, we first list some common errors made by the system developer. Then the property analysis is discussed. Finally, our proposed approach is compared with other methods such as CTL (Guo et al., 2013) and LTL (Mukund, 1996; Naldurg et al., 2004).

Common Errors Detection

This sub-section lists the common errors made by the system developer. Many of these errors are based on the internal details that are easily overlooked by the system developer but can potentially lead to the unexpected results. The file checker and the rule checker presented in Section 3 can solve these problems.

File Error Detection

In Hadoop, the data sets uploaded by users are stored in the HDFS and serve as the input data needed for system operation. The first step is to read the data sets in the HDFS. If the system cannot find the files or an output file existing, then a system error will occur. Such an error is depicted in Figure 10.

Rule Error Detection

This is a big-data analysis system that allows users to upload files and analyze rules. However, the system developer must consider various situations in which incorrect rules have been input. They may be contradictory, or interdependent

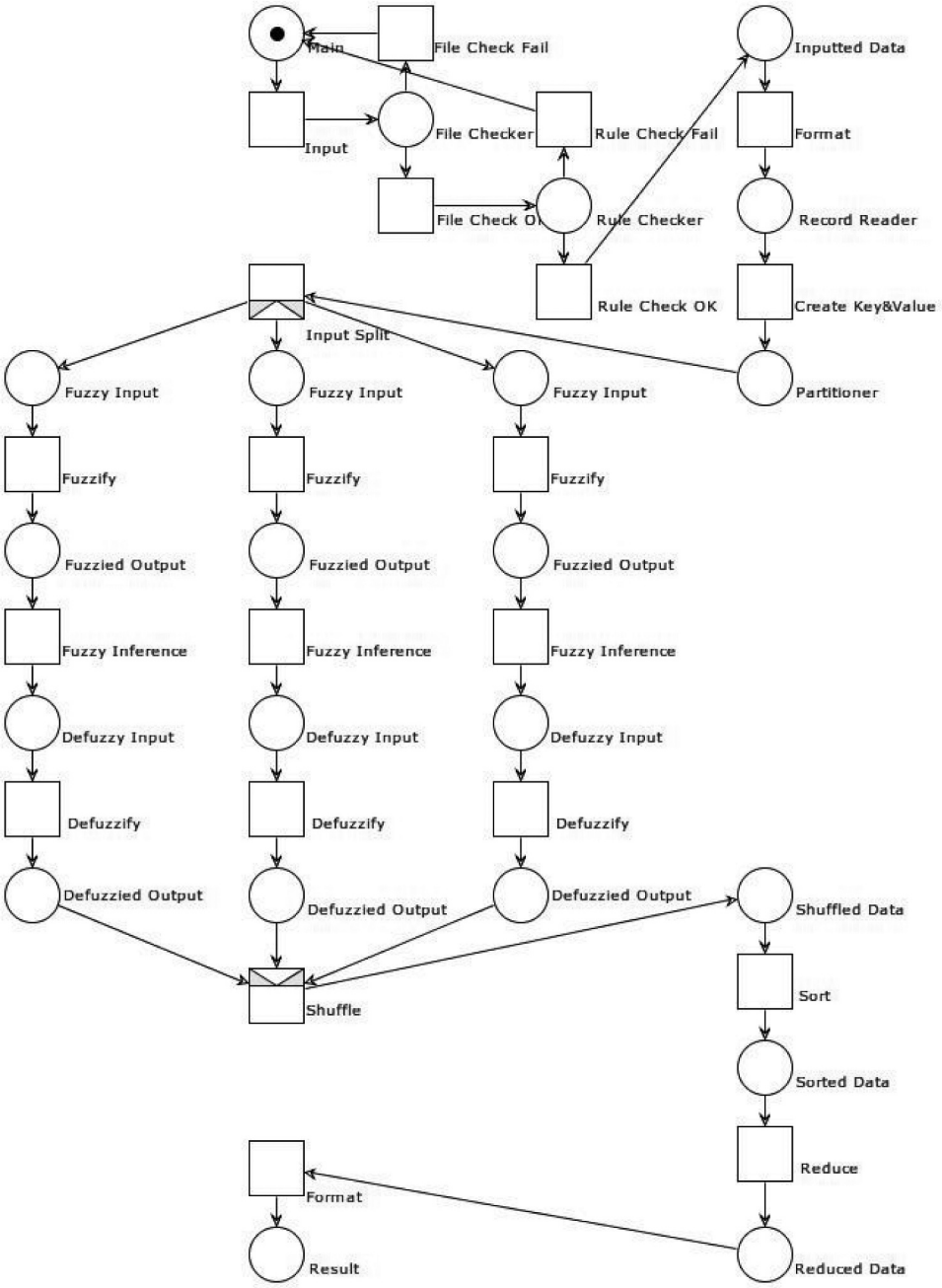

Figure 9. Big data analysis system by inclusion of checking mechanisms.

but in the wrong order. The rule checker described in Sub-section 3.3 prevents such errors, an example of which is shown in Figure 11.

The rule checker must be activated before MapReduce is executed because the rules determine the methods of <key, value> pairs generation and data distribution in the Map function. The rule checker must ensure that all of the

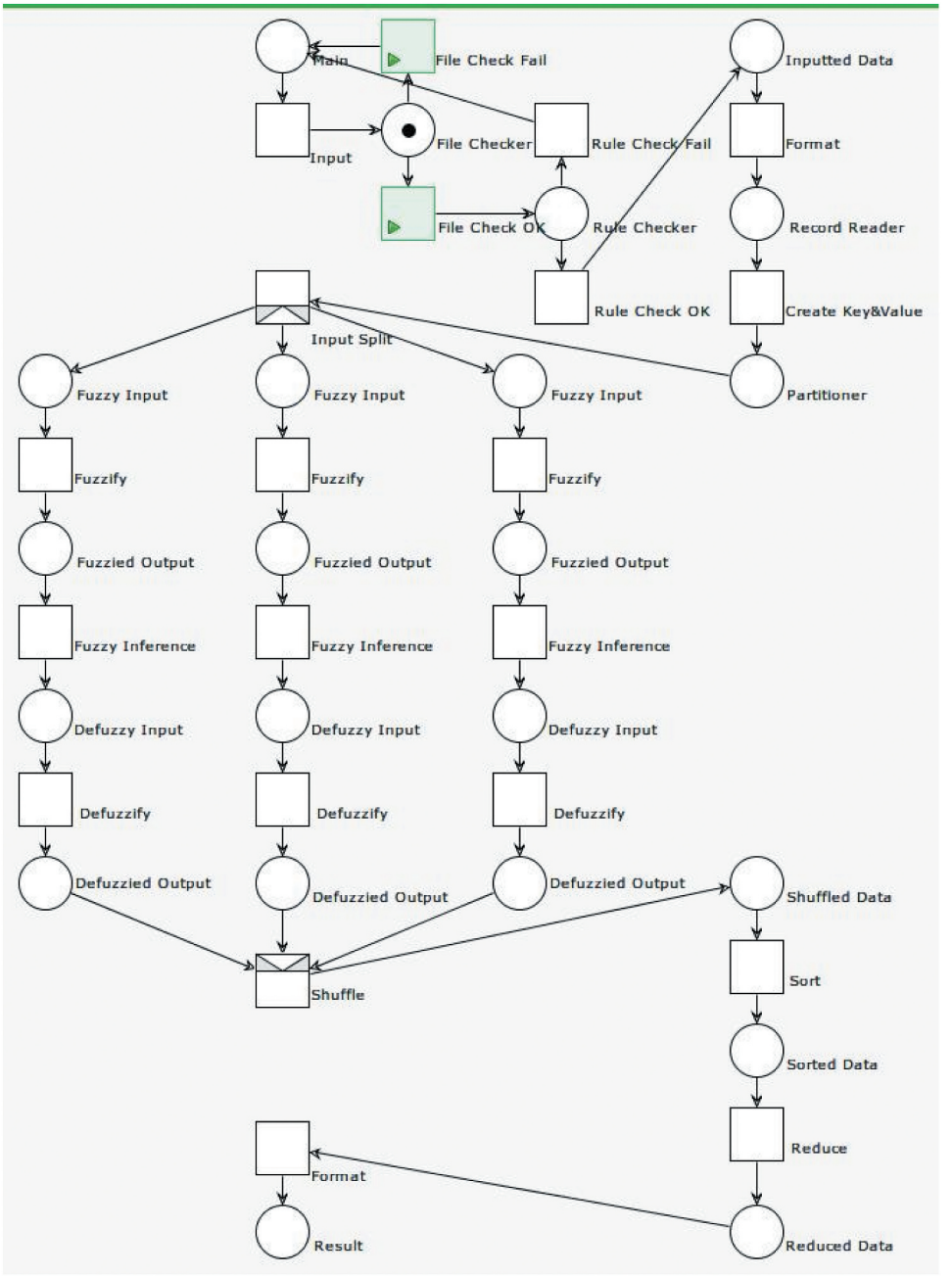

Figure 10. File error.

rules input by users fulfill the following requirements before executing the next step of the procedure below:

(1) The rules must conform to the required format of the system.

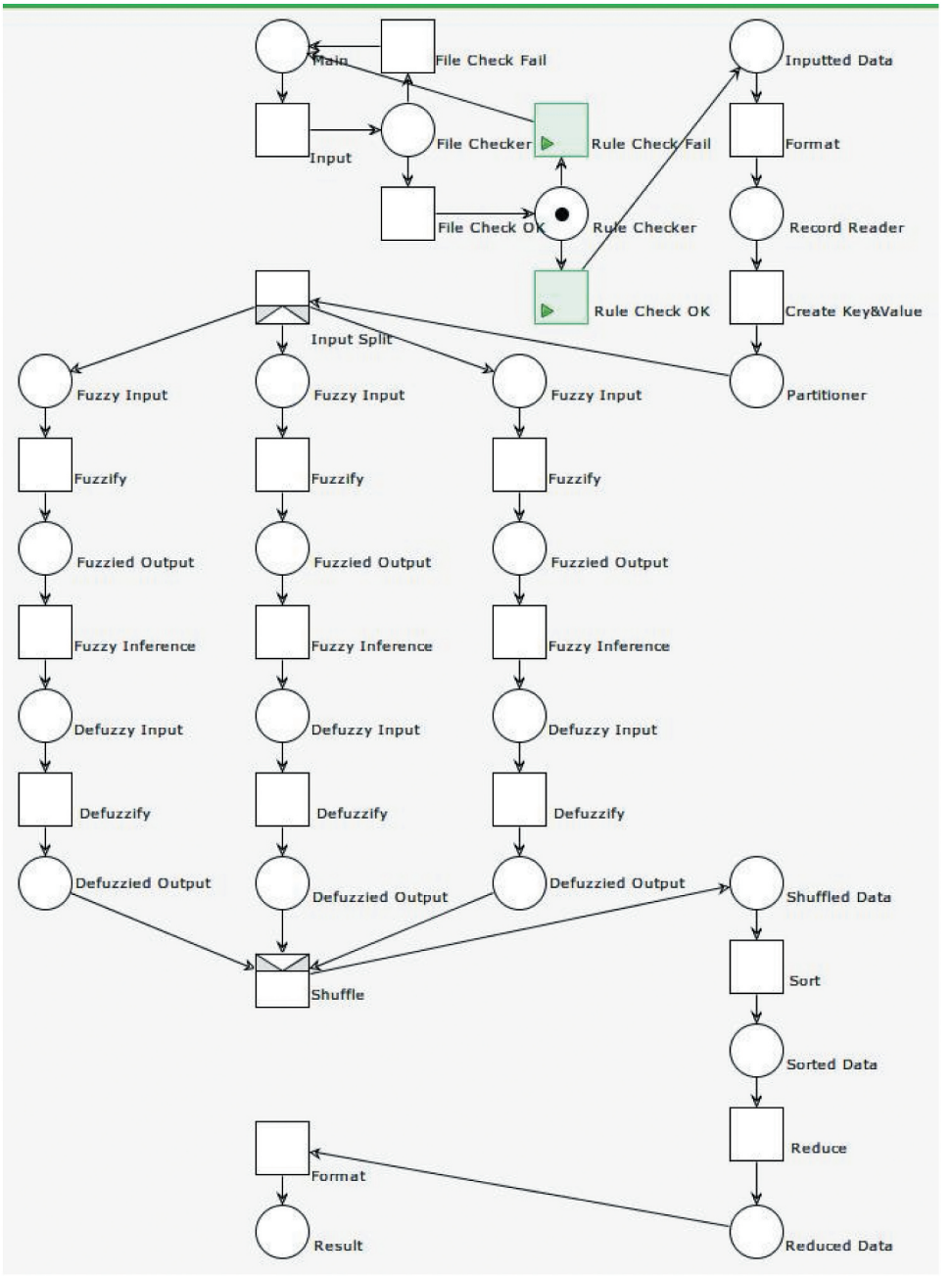

Figure 11. Rule error.

(2) The rules cannot contradict one another.
(3) The interdependent rules must be in the right order.

<Key, Value> Pairs Error Detection

Once the rules input by users have been checked, the Map function is executed. Users must carefully check the accuracy of <key, value> pairs generation method. However, as this portion is hidden in the Map framework,

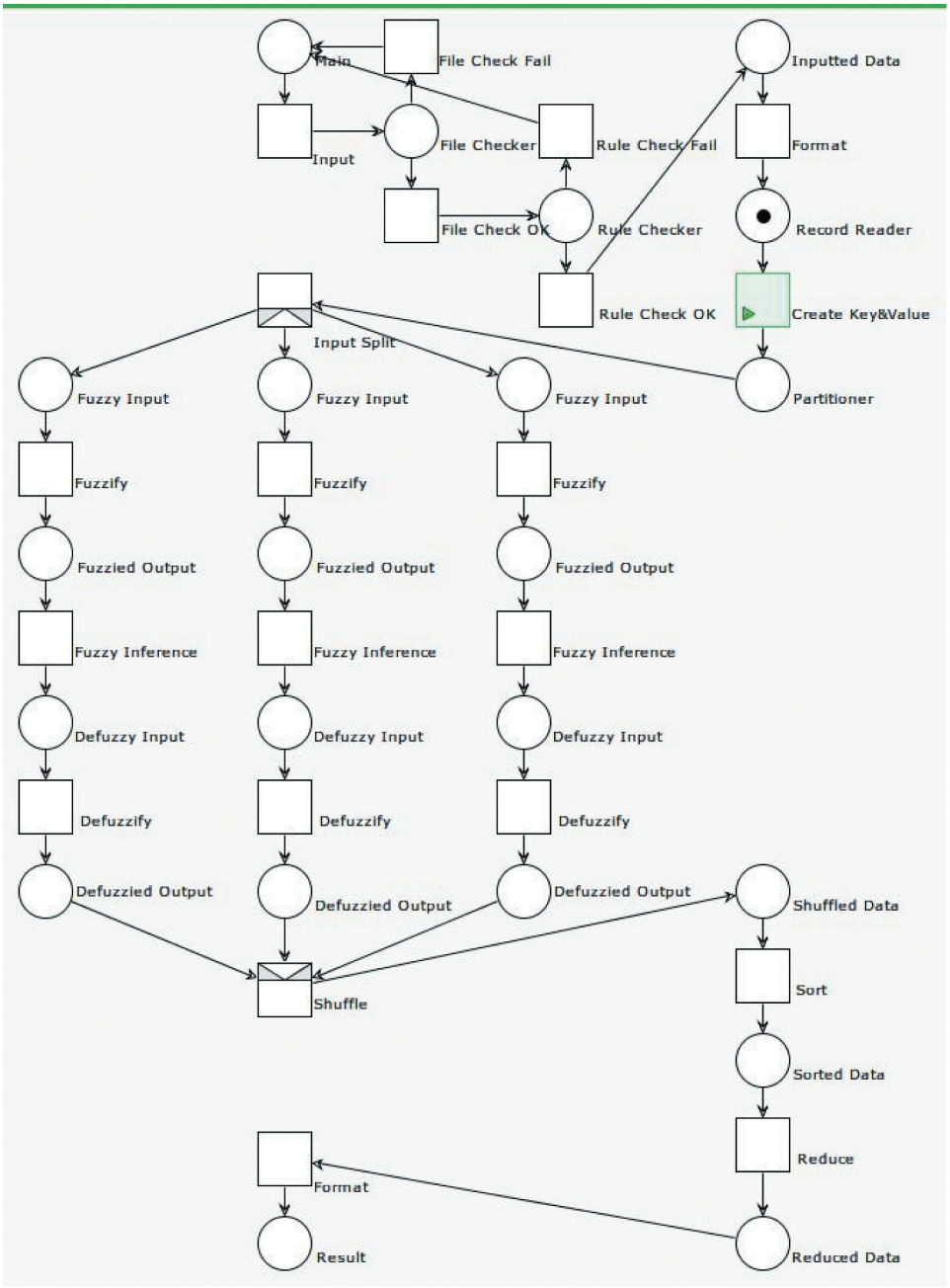

Figure 12. <key, value> pairs error

it is often overlooked. If the system developer does not customize this portion, the MapReduce framework will use the default settings. However, the default approach may be different from what the system developer envisioned and lead to the unanticipated analysis procedure and results. Such an error is depicted in Figure 12.

Property Analysis

In this Sub-section, we discuss the properties of the proposed MapReduce-based Petri net models. Note that, all MapReduce details and solutions have been depicted in Figure 9.

Error Resolution

Errors appear in the Petri net model as shown in Figure 6. If users input data with an error, then the MapReduce system may have an error in the *input* place. We can add a file checker after the *input* place, and therefore avoid the file error as shown in Sub-section 4-1-1. If users input rules with conflict, then the system may have an error in the result. We can add a rule checker after the file checker, and avoid the rule error as shown in Sub-section 4-1-2. If the system developer does not rewrite the record reader for defining a new way to create <key, value> pairs, then the analysis result may be wrong; but the system still has the reachability as shown in Sub-section 4-1-3.

Reachability Analysis

We verify the properties and correctness of the MapReduce-based Petri net model through reachability analysis. Similar to those cases in Figures 1 and 6, we can construct the reachability graph of the MapReduce-based Petri net model in Figure 9, and the reachability graph of a composition model of the ones in Figures 7 and 8 to avoid some possible errors in the system. Their analyses suggest that the MapReduce-based Petri net models are live and reversible.

Comparison of the Proposed Approach with Other Methods

In this sub-section, we compare the proposed Petri net approach with computation tree logic (CTL) and linear temporal logic (LTL). Both of these methods can be used to verify systems, but they lack the visual appeal and ease of system simulation of the MapReduce framework.

CTL is a branching-time logic which models the system evolution as a tree-like structure where each state can evolve in several ways. Future directions cannot be confirmed, and any branch could become reality. CTL is often used to analyze and verify software and hardware systems (Camilli et al. 2014).

LTL is a modal temporal logic. In LTL, the future is viewed as a sequence of states, so the future is viewed as a path. For example, a condition will eventually be true, a condition will be true until another fact becomes true, etc. LTL can use mathematical models to track the state changes of a system for verification and employ various formal languages to describe different types of constraints, e.g. *Boolean connectives* (Hallé and Soucy-Boivin 2015).

Both CTL and LTL use the distributed search techniques to increase intrinsic availability and reduce the overall calculation time of system verification processes.

CTL and LTL are both subsets of CTL*, but are not equivalent to each other. For example:

- There are things we can write in CTL but not in LTL, things like "on all paths is always true that exists a path such that . . .".
- There are things we can write in LTL but not in CTL, like "If it is infinitely often the case that p is then at some point q".

Petri net models can be used to describe parallel systems, so Petri nets fully express the parallel computing of the MapReduce framework and present it as a visual model, which can also be converted into a mathematical model for further analysis.

This study uses Petri nets to analyze the MapReduce framework and to verify its reachability, which acts as guidelines for the system developer, making a detailed description of the internal procedure of the MapReduce framework so as to increase its development efficiency.

The characteristics of the proposed approach compared with other two commonly used methods are shown in Table 1.

Conclusion

This study has examined the parallel procedure of the MapReduce framework in detail and used Petri nets for visualized modeling and analysis. We applied the proposed approach to a real-world example of big data analysis system to demonstrate its feasibility. To prevent common errors, we have included a file checker and a rule checker in the framework. The guidelines and algorithms

Table 1. Comparison of the proposed approach with CTL and LTL.

Analysis Method	Proposed Petri Nets	Computation Tree Logic (CTL)	Linear Temporal Logic (LTL)
Presentation	Visual or Mathematical Model	Mathematical Model	Mathematical Model
Ease of Simulation	Easy	Difficult	Difficult
Target of Analysis	Verification of reachability in detailed procedures of MapReduce using visual Petri net models and facilitation of tracking and error prevention for system developers	The temporal model of CTL is a dendritic structure in which future directions cannot be confirmed, and any branch could become reality.	In LTL, the future is seen as a sequence of states, so the future is viewed as a path. It is capable of using mathematical models to track the state changes of systems.

provided by this study can help the system developer increase the efficiency in the system development.

This study has made the following contributions:

- Analysis of the MapReduce framework using visual Petri net models;
- Verification of the reachability in the MapReduce framework using Petri nets;
- Application of the proposed approach to verify the real parallel MapReduce system, listing common errors as well as including check mechanism to prevent errors;
- Detailed guidelines useful for the system developer.

In the future work, we will extend the Petri net model proposed in this study to other parallel MapReduce systems and provide more real-world examples to assist the system engineer in the development of parallel MapReduce systems.

Acknowledgments

The authors are very grateful to the anonymous reviewers for their constructive comments which have improved the quality of this paper. Also, this work was supported by the Ministry of Science and Technology, Taiwan, under grants MOST 107-2221-E-845- 001-MY3 and MOST 107-2221-E-845- 002-MY3.

Funding

Financial support for the completion of annual AI project.

References

Birzhandi, P., K. TaeKim, B. Lee, and H. Y. Youn. 2019. Reduction of training data using parallel hyperplane for support vector machine. *Applied Artificial Intelligence* 33 (6):497–516. doi:10.1080/08839514.2019.1583449.

Camilli, M. 2012. Petri nets state space analysis in the cloud. Proceedings of the 2012 International Conference on Software Engineering, Zurich, Switzerland, pp. 1638–40.

Camilli, M., C. Bellettini, L. Capra, and M. Monga. 2014. CTL model checking in the cloud using MapReduce. *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. 333–40.

Freytag, T., and M. Sänger. 2014. WoPeD – An educational tool for workflow nets," Proceedings of the BPM Demo Sessions, Eindhoven, Haifa, Israel, September. pp. 31–35.

Guo, F., G. Wei, M. Deng, and W. Shi. 2013. CTL model checking algorithm using MapReduce. *Emerging Technologies for Information Systems, Computing, and Managemen, LNEE* 236:341–48.

Hallé, S., and M. Soucy-Boivin. 2015. MapReduce for parallel trace validation of LTL properties. *Journal of Cloud Computing: Advances, Systems and Applications* 4(1). doi:10.1186/s13677-015-0032-x.

Jadhav, A., D. Pramod, and K. Ramanathan. 2019. Comparison of performance of data imputation methods for numeric dataset. *Applied Artificial Intelligence* 33(10):913–33. doi: 10.1080/08839514.2019.1637138.

Karun Kala, A., and K. Chitharanjan. 2013. A review on Hadoop – HDFS infrastructure extensions. 2013 IEEE Conference on Information and Communication Technologies (ICT 2013), Thuckalay, Tamil Nadu, India.

Matsuzaki, K. 2017. Functional models of Hadoop MapReduce with application to scan. *International Journal of Parallel Programming* 45(2):362–81. doi: 10.1007/s10766-016-0414-9.

Mazhar Rathore, M., H. Son, A. Ahmad, A. Paul, and G. Jeon. 2018. Real-time big data stream processing using GPU with spark over Hadoop ecosystem. *International Journal of Parallel Programming* 46(3):630–46. doi: 10.1007/s10766-017-0513-2.

Mukund, M. 1996. *Linear-time temporal logic and Büchi automata*. India: SPIC Mathematical Institute.

Naldurg, P., K. Sen, and P. Thati. 2004. A temporal logic based framework for intrusion detection. *FORTE, of Lecture Notes in Computer Science* 3235:359–76.

Natesan, P., R. R. Rajalaxmi, G. Gowrison, and P. Balasubramanie. 2017. Hadoop based parallel binary bat algorithm for network intrusion detection. *International Journal of Parallel Programming* 45(5):1194–213. doi: 10.1007/s10766-016-0456-z.

Neil, J., P. P. Gunther, and K. Tomasette. 2014. Hadoop superlinear scalability. *Communications of the ACM* 58:46–55.

Saisai, M., L. Jiuyong, L. Liu, and T. D. Le. 2016. Mining combined causes in large data sets. *Knowledge-Based Systems* 92:104–11. doi: 10.1016/j.knosys.2015.10.018.

Valero, V. 2018. Strong behavioral similarities in time-arc Petri nets. *Applied Mathematics and Computation* 333:401–15. doi: 10.1016/j.amc.2018.03.073.

White, T. 2009. *Hadoop: the definitive guide*. USA: O'Reilly Media.

Wu, N. Q., and M. C. Zhou. 2010. *System modeling and control with resource-oriented petri nets*. New York: CRC Press.

Xiong, P. C., Y. S. Fan, and M. C. Zhou. 2010. A Petri net approach to analysis and composition of web services. *IEEE Transactions on System, Man, and Cybernetics—Part A: Systems and Humans* 40(2):376–87. doi: 10.1109/TSMCA.2009.2037018.

Zhang, S., C. Zhang, and Q. Yang. 2010. Data preparation for data mining. *Applied Artificial Intelligence* 17(5–6):375–81. doi: 10.1080/713827180.