International Journal of Approximate Reasoning 55 (2014) 896–907

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

A comparison of parallel large-scale knowledge acquisition using rough set theory on different MapReduce runtime systems ☆

Junbo Zhang [a,b], Jian-Syuan Wong [b], Tianrui Li [a,*], Yi Pan [b]

[a] School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China
[b] Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA

ARTICLE INFO

Article history:
Received 19 November 2012
Received in revised form 5 May 2013
Accepted 20 August 2013
Available online 17 September 2013

Keywords:
Rough sets
Knowledge acquisition
MapReduce
Large-scale

ABSTRACT

Nowadays, with the volume of data growing at an unprecedented rate, large-scale data mining and knowledge discovery have become a new challenge. Rough set theory for knowledge acquisition has been successfully applied in data mining. The recently introduced MapReduce technique has received much attention from both scientific community and industry for its applicability in big data analysis. To mine knowledge from big data, we present parallel large-scale rough set based methods for knowledge acquisition using MapReduce in this paper. We implemented them on several representative MapReduce runtime systems: Hadoop, Phoenix and Twister. Performance comparisons on these runtime systems are reported in this paper. The experimental results show that (1) The computational time is mostly minimum on Twister while employing the same cores; (2) Hadoop has the best speedup for larger data sets; (3) Phoenix has the best speedup for smaller data sets. The excellent speedups also demonstrate that the proposed parallel methods can effectively process very large data on different runtime systems. Pitfalls and advantages of these runtime systems are also illustrated through our experiments, which are helpful for users to decide which runtime system should be used in their applications.

1. Introduction

With the development of information technology, amount of data are collected from various sensors and devices in multiple formats. Such data processed by independent or connected applications will routinely cross the peta-scale threshold, which would in turn increase the computational requirements. The fast increase and update of big data brings a new challenge to quickly acquire the useful information with data mining techniques. For the purpose of processing big data, Google developed a software framework called MapReduce to support large distributed data sets on clusters of computers [8,9], which is effective to analyze large amounts of data. As one of the most important cloud computing techniques, MapReduce has been a popular computing model for cloud computing platforms [49]. To extend the MapReduce to be support for iterative programs in many applications including data mining, web ranking and graph analysis, iterative MapReduce (iMapReduce) are proposed [6,11]. In addition, many MapReduce runtime systems are developed and lots of traditional

☆ This is an extended version of the paper presented at 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, KDD2012, Beijing, 2012, China.

* Corresponding author.
E-mail addresses: JunboZhang86@163.com, jbzhang@cs.gsu.edu (J. Zhang), jwong9@student.gsu.edu (J.-S. Wong), trli@swjtu.edu.cn (T. Li), pan@cs.gsu.edu (Y. Pan).

0888-613X/$ – see front matter
http://dx.doi.org/10.1016/j.ijar.2013.08.003

methods combined with MapReduce have been presented. Here we review some implementations of MapReduce, iMapReduce, and traditional methods combined with MapReduce model.

- *Implementations of MapReduce*.
  (1) Apache Hadoop [41] was developed for data-intensive distributed applications. It is open source software framework and helps to construct the reliable, scalable, distributed systems. (2) Phoenix [37] is a shared-memory implementation of MapReduce model for data-intensive processing tasks, which can be used to program multi-core chips as well as shared-memory multiprocessors. (3) Aiming to provide a generic framework for developers to implement data- and computation-intensive tasks correctly, efficiently, and easily on the GPU, Mars [18] was developed for graphic processors (GPUs) using MapReduce framework. Mars hides the programming complexity of the GPU behind the simple and familiar MapReduce interface. Hence, the developers can write their code on the GPU without any knowledge of the graphics APIs or the GPU architecture. (4) MapReduceRoles4Azure (MR4Azure) [14] is a distributed decentralized MapReduce runtime for Windows Azure that was developed using Azure cloud infrastructure services.
- *Implementations of iMapReduce*.
  (1) Iterative MapReduce model was introduced in Twister [11], which is a lightweight MapReduce runtime system. It provides the feature for cacheable MapReduce task, which allows developer to develop iterative applications without spending much time on reading and writing large amount of data in each iteration. Another version for Windows Azure called Twister4Azure [15] also has been released. (2) HaLoop [6] is a modified version of the Hadoop MapReduce framework, designed to serve iterative applications. It does not only extend MapReduce with programming support for iterative applications, but also dramatically improves their efficiency by making the task scheduler loop-aware and by adding various caching mechanisms. (3) In addition, Microsoft also developed an iterative MapReduce runtime for Windows Azure, code-named Daytona [3], which is designed to support a wide class of data analytics and machine learning algorithms. It can scale out to hundreds of server cores for analysis of distributed data.
- *Traditional methods combined with MapReduce*.
  Apache Mahout [33] can help developers to produce implementations of scalable machine-learning algorithms on Hadoop platform. Menon et al. gave a rapid parallel genome indexing with MapReduce [31]. Blanas et al. proposed crucial implementation details of a number of well-known join strategies for log processing in MapReduce [5]. Ene et al. developed fast clustering algorithms using MapReduce with constant factor approximation guarantees [12]. Lin et al. presented three design patterns for efficient graph algorithms in MapReduce [29].

Granular computing (GrC) is an emerging information processing paradigm in computational intelligence [50]. It is a framework to create computer systems employing a human-centric view of the world [10]. Rough set theory is considered as one of the leading special cases of GrC approaches. As one of data analysis techniques, rough sets based methods have been successfully applied in data mining and knowledge discovery during last decades [13,34,45], and particularly useful for rule acquisition [22–25,40] and feature selection [19,20,27,28,35,44].

To mine knowledge from very large data sets based on rough sets, incremental techniques are employed to improve the computational efficiency [7,26,30,48]. In addition, positive approximation can be used to accelerate feature selection and rule acquisition process [21,35,36]. Susmaga presented an effective method for parallel computation of reducts with rough set theory [38]. But, to our knowledge, most of the current algorithms based on rough sets are the sequential algorithms and corresponding tools only run on a single computer to deal with small data sets. To expand the applications of rough sets in the field of data mining and knowledge discovery from big data, we proposed a parallel method for computing approximations based on rough sets and MapReduce [47]. Furthermore, a parallel method for knowledge acquisition using MapReduce was presented [46]. Based on these work, we discuss about rough set based parallel large-scale methods for knowledge acquisition in this paper. The corresponding parallel algorithms are designed for knowledge acquisition on the basis of the characteristics of the data. The proposed algorithms are implemented on several representative MapReduce runtime systems: Hadoop [41], Phoenix [39] and Twister [11]. We test the proposed algorithms on these runtime systems and compare their performance. Comprehensive experimental results demonstrate that the proposed algorithms can effectively process very large data sets.

The paper is organized as follows. Section 2 includes a background introduction to MapReduce and rough sets. Rough set based methods for knowledge acquisition with MapReduce are presented in Section 3. Experimental analysis is given in Section 4. The paper ends with conclusions in Section 5.

2. Preliminaries

In this section, we review MapReduce technique [8,9] and some basic concepts of rough sets and knowledge acquisition [30,34,40,47].

2.1. MapReduce programming model

MapReduce [8,9], by Google, is a distributed programming model for processing large-scale data, which is described as follows.

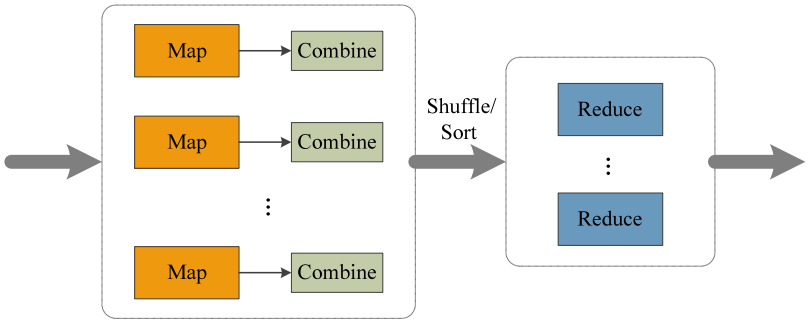

Fig. 1. The MapReduce model.

The computation takes a set of input *key/value* pairs, and produces a set of output *key/value* pairs. Users specify the necessary functions, *i.e.*, Map and Reduce and one optional function, *i.e.*, Combine. Fig. 1 shows the MapReduce model.

- **Map** takes an input pair and produces a set of intermediate *key/value* pairs. The input pair often comes from a partition of data specified by the MapReduce architecture. The MapReduce library groups together all intermediate values associated with the same intermediate key I and transforms them with the Combine/Reduce function.
- **Combine (optional)** is a local Reduce, which can do local computations to lessen the burden. It accepts a key I and a set of values for that key from the local Map. Then, it merges together these values to form a possibly smaller set of values. All of the results are shuffled, sorted, and sent to the Reduce function.
- **Reduce** accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values and produces zero or more outputs.

2.2. Rough sets

Given a pair $K = (U, R)$, where U is a non-empty finite set called the universe, and $R \subseteq U \times U$ is an equivalence relation on U. The pair $K = (U, R)$ is called an approximation space. The equivalence relation R partitions the set U into several disjoint subsets. This partition of the universe forms a quotient set induced by R, denoted by U/R. If two elements, $x, y \in U$, are indistinguishable under R, we say x and y belong to the same equivalence class. The equivalence class including x is denoted by $[x]_R$.

An approximation space $K = (U, R)$ is characterized by an information system $S = (U, A, V, f)$, where

$$\begin{cases} U & \text{is a non-empty finite set of objects, called a universe.} \\ A & \text{is a non-empty finite set of attributes (features).} \\ V & \text{equals to } \bigcup_{a \in A} V_a, V_a \text{ is a domain of the attribute } a. \\ f & \text{is an information function } U \times A \to V, \\ & \text{such that } f(x, a) \in V_a \text{ for every } x \in U, a \in A. \end{cases}$$

Specifically, $S = (U, A, V, f)$ is called a decision table if $A = C \cup D$, where C is a set of condition attributes and D is a set of decision, $C \cap D = \emptyset$.

Definition 1. Let $B = \{b_1, b_2, \ldots, b_l\} \subseteq C$ be a subset of condition attributes. The information set with respect to B for any object $x \in U$ can be denoted by the tuple

$$\vec{x}_B = \langle f(x, b_1), f(x, b_2), \ldots, f(x, b_l) \rangle. \tag{1}$$

An equivalence relation with respect to B called the indiscernibility relation, denoted by $IND(B)$, is defined as

$$IND(B) = \left\{ (x, y) \;\middle|\; (x, y) \in U \times U, \vec{x}_B = \vec{y}_B \right\}. \tag{2}$$

Two objects x, y satisfying the relation $IND(B)$ are indiscernible by attributes from B.

The equivalence relation $IND(B)$ partitions U into some equivalence classes given by:

$$U/IND(B) = \left\{ [x]_B \;\middle|\; x \in U \right\}, \tag{3}$$

where $[x]_B$ denotes the equivalence class determined by x with respect to B, $[x]_B = \{y \in U \mid (x, y) \in IND(B)\}$. For simplicity, $U/IND(B)$ will be denoted by U/B.

Table 1
A decision table S.

U	Headache	Temperature	Flu
x_1	no	normal	no
x_2	no	high	yes
x_3	yes	normal	yes
x_4	yes	veryhigh	yes
x_5	no	normal	no
x_6	yes	high	no
x_7	no	normal	no
x_8	no	high	yes
x_9	yes	high	yes
x_{10}	yes	veryhigh	yes
x_{11}	no	normal	no
x_{12}	yes	high	yes

Example 1. A decision table $S = (U, A, V, f)$ is given in Table 1, where Headache, Temperature are the condition attributes and Flu is the decision.

Let $B = \{\text{Headache}, \text{Temperature}\}$. We compute the equivalence classes. According to Definition 1, we have $U/B = \{E_1, E_2, E_3, E_4, E_5\}$, where

$$\begin{cases} E_1 = [x_1]_B = [x_5]_B = [x_7]_B = [x_{11}]_B = \{x_1, x_5, x_7, x_{11}\} \\ E_2 = [x_2]_B = [x_8]_B = \{x_2, x_8\} \\ E_3 = [x_3]_B = \{x_3\} \\ E_4 = [x_4]_B = [x_{10}]_B = \{x_4, x_{10}\} \\ E_5 = [x_6]_B = [x_9]_B = [x_{12}]_B = \{x_6, x_9, x_{12}\}. \end{cases}$$

Definition 2. Let $B_1 = \{b_{11}, \ldots, b_{1l_1}\}$, $B_2 = \{b_{21}, \ldots, b_{2l_2}\}$ be two attribute sets, where $B_1 \cap B_2 = \emptyset$. The information set with respect to $B = B_1 \cup B_2$ for any object $x \in U$ can be denoted by

$$\overrightarrow{x}_B = \overrightarrow{x}_{B_1 \cup B_2} = \langle f(x, b_{11}), \ldots, f(x, b_{1l_1}), f(x, b_{21}), \ldots, f(x, b_{2l_2}) \rangle. \tag{4}$$

Definition 3. Let $B \subseteq A$ be a subset of attributes and U/B be a partition of U defined by B. The information set with respect to B for any equivalence class $E = [x]_B \in U/B$ is denoted by

$$\overrightarrow{E}_B = \overrightarrow{x}_B. \tag{5}$$

Example 2 *(Example 1 continued).* According to Example 1, $E_2 = [x_2]_B = [x_8]_B = \{x_2, x_8\}$, hence, $\overrightarrow{E}_{2B} = \overrightarrow{x}_{2B} = \overrightarrow{x}_{8B} = \langle no, high \rangle$.

2.3. Knowledge acquisition based on rough sets

In the earlier studies, Wong et al. used the confidence and resolution factor for inductive learning [42]. Tsumoto proposed the accuracy and coverage to measure the degree of sufficiency and necessity, respectively, and acquired classification rules with high accuracy and high coverage [40]. Han et al. introduced the rule support and rule confidence to measure the degree to which a rule is interesting rule in the field of data mining [17]. Liu et al. used the three parameters: support, accuracy and coverage to discover the interesting knowledge [30].

Given a decision information system $S = (U, C \cup D, V, f)$ where $B = \{b_1, b_2, \ldots, b_l\} \subseteq C$ and $D = \{d\}$. $\forall x \in U$, we generate a decision rule in the following way. The conjunction $\wedge_{b \in B}(b = f(x, b))$ and the decision attribute d with value $f(x, d)$ are taken as the predecessor and successor of a decision rule, respectively. Therefore, the constructed decision rule for the object x is of the form: $\wedge_{b \in B}(b = f(x, b)) \to d = f(x, d)$.

Definition 4. (See [30].) Let $U/B = \{E_1, E_2, \ldots, E_M\}$ be a partition of U defined by B and $U/D = \{D_1, D_2, \ldots, D_N\}$ be a partition of U defined by D. $\forall E_i \in U/B$, $\forall D_j \in U/D$, the support, accuracy and coverage of $E_i \to D_j$ are defined respectively as follows:

$$\begin{aligned} &\text{Support of } E_i \to D_j\text{:} \quad Supp(D_j|E_i) = |E_i \cap D_j|; \\ &\text{Accuracy of } E_i \to D_j\text{:} \quad Acc(D_j|E_i) = \frac{|E_i \cap D_j|}{|E_i|}; \\ &\text{Coverage of } E_i \to D_j\text{:} \quad Cov(D_j|E_i) = \frac{|E_i \cap D_j|}{|D_j|}, \end{aligned}$$

where $|\bullet|$ denotes the cardinality of the set.

Remark. It is easy to notice that $E_i \cap D_j \in U/C \cup D$. We call $E_i \cap D_j$ an intersection class.

Here, we introduce several rule acquisition methods that all use the accuracy and the coverage.

Definition 5 *(Method 1).* (See [22].) $\forall E_i \ (i = 1, 2, \ldots, M)$, $\forall D_j \ (j = 1, 2, \ldots, N)$, if $Acc(D_j|E_i) = 1$ holds, we call the rule $E_i \to D_j$ a consistent rule with the coverage $Cov(D_j|E_i)$.

Definition 6 *(Method 2).* (See [30,40].) $\forall E_i \ (i = 1, 2, \ldots, M)$, $\forall D_j \ (j = 1, 2, \ldots, N)$, if $Acc(D_j|E_i) \geqslant \alpha$ and $Cov(D_j|E_i) \geqslant \beta$ hold, we call the rule $E_i \to D_j$ a probabilistic rule where $\alpha \in (0.5, 1)$ and $\beta \in (0, 1)$.

The above two methods for rule acquisition can only induce rules when the accuracy $\alpha = 1$ and $\alpha \in (0.5, 1)$, respectively. To induce rules when $\alpha \in (0, 0.5]$, we give the following method.

Definition 7 *(Method 3).* $\forall E_i \ (i = 1, 2, \ldots, M)$, $\forall D_j \ (j = 1, 2, \ldots, N)$, if $Acc(D_j|E_i) = \max_{k=1,2,\ldots,N}\{Acc(D_k|E_i)\} \geqslant \alpha'$ and $Cov(D_j|E_i) \geqslant \beta'$ hold, we call the rule $E_i \to D_j$ a max-accuracy probabilistic rule where $\alpha' \in (0, 1)$ and $\beta' \in (0, 1)$.

3. Parallel large-scale knowledge acquisition based on MapReduce

In this section, we give a parallel large-scale method for knowledge acquisition based on rough sets using MapReduce.

Definition 8. (See [47].) Given a decision table $S = (U, C \cup D, V, f)$. Let $S_i = (U_i, C \cup D, V, f)$, $i \in \{1, 2, \ldots, m\}$. Suppose (1) $U = \bigcup_{i=1}^{m} U_i$; (2) $U_j \cap U_k = \emptyset$, $\forall j, k \in \{1, 2, \ldots, m\}$ and $j \neq k$. This means the decision table S is divided into m decision sub-tables. Then we call S_i is a decision sub-table of S.

Lemma 1. *(See [47].) Let $B \subseteq C$ be a subset of attributes, E, F be two equivalence classes with respect to B from two different sub-tables of S. One of the following results holds*:

(1) *If $\overrightarrow{E}_B = \overrightarrow{F}_B$, then these two equivalence classes E and F can be combined as one equivalence class G with respect to B, where $G = E \cup F$ and $\overrightarrow{G}_B = \overrightarrow{E}_B = \overrightarrow{F}_B$;*
(2) *If $\overrightarrow{E}_B \neq \overrightarrow{F}_B$, then these two equivalence classes E and F cannot be combined as one equivalence class with respect to B.*

Zhang et al. proposed a parallel strategy to compute equivalence classes and decision classes based on MapReduce [47]. Here, we give the following corollary.

Corollary 1. *Given a decision table $S = (U, C \cup D, V, f)$ and its decision sub-table $S_i = (U_i, C \cup D, V, f)$, $i \in \{1, 2, \ldots, m\}$. For any subset $B \subseteq C$, $U/B = \{E_1, E_2, \ldots, E_t\}$ and $\forall i \in \{1, 2, \ldots, m\}$, $U_i/B = \{E_{i1}, E_{i2}, \ldots, E_{ip_i}\}$. Let $E_{all} = \bigcup_{i=1}^{m} U_i/B = \{E_{11}, E_{12}, \ldots, E_{1p_1}, \ldots, E_{m1}, E_{m2}, \ldots, E_{mp_m}\}$. Therefore, for any $E_j \in U/B$, $j \in \{1, 2, \ldots, t\}$, we have*

$$|E_j| = \sum_{F \in E_{all}, \overrightarrow{F}_B = \overrightarrow{E_j}_B} |F|. \tag{6}$$

According to Corollary 1, each sub-decision table can compute numbers of elements in equivalence classes, decision classes and intersection classes independently. At the same time, the classes of different sub-decision tables can combine together if their information sets are the same. Hence, it could be changed to a MapReduce problem and we design three parallel methods based on rough set theory for knowledge acquisition according to Definitions 5–7. Fig. 2 shows our parallel models for three kinds of knowledge acquisition methods, which all contain two steps.

- **Step 1:** By the above analysis, the computation of the cardinality of the equivalence classes $|E|$, the cardinality of decision classes $|D|$ and the cardinality of intersection classes $|E \cap D|$ can be executed in parallel. In detail, we present the corresponding algorithms based on MapReduce/Combine, which are outlined in Algorithms 1, 2 and 3, respectively.
- **Step 2:** After computing the cardinalities of the equivalence classes, decision classes and intersection classes, by Definition 4, the accuracy $Acc(D|E)$ and the coverage $Cov(D|E)$ are computed. Then, three kinds of rule sets are generated according to Definitions 5–7, respectively.

Remark. Step 1 of these three methods are same and are executed in parallel. Step 2 of these three methods due to Definitions 5–7 and are executed in sequential.

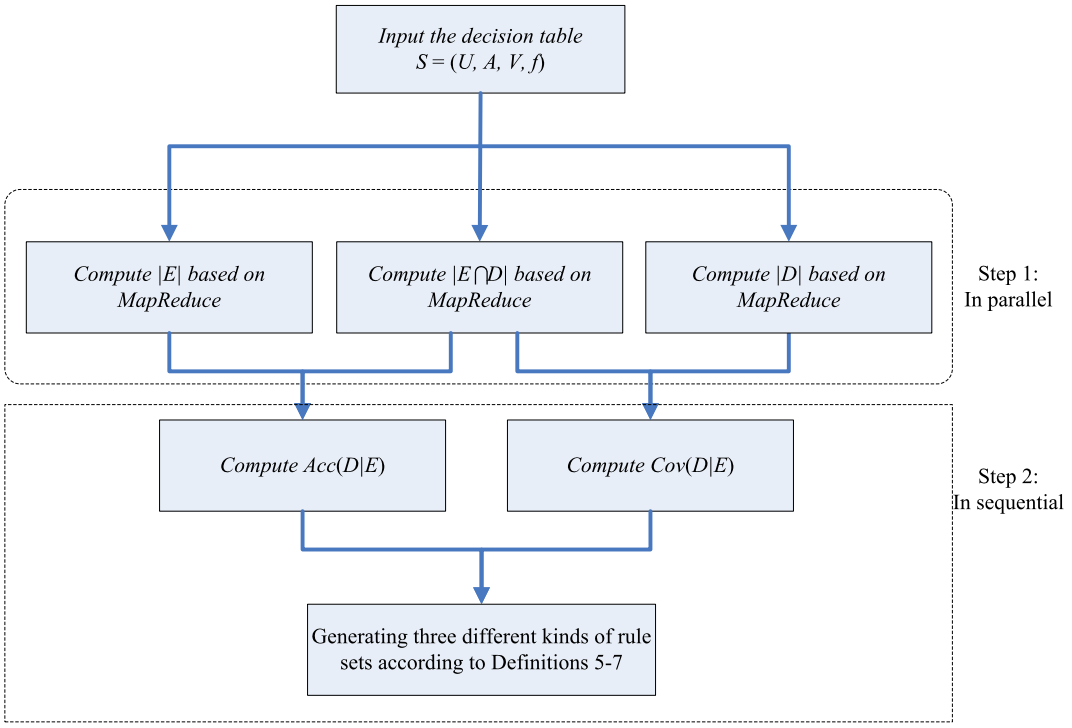

Fig. 2. Parallel method for knowledge acquisition.

Algorithm 1: Map(*key*, *value*).

Input:
//*key*: document name
//*value*: $S_i = \{U_i, C \cup D, V, f\}$
//*Global variable*: $B \subseteq C$
Output:
//*key'*: the information set of the object with respect to the sets B, D and $B \cup D$
//*value'*: the count

```
1  begin
2      for each x ∈ U_i do
3          let key' = 'E'+ x⃗_B; //Here, 'E' is flag, which means the equivalence class
4          output.collect(key', 1);
5          let key' = 'D'+ x⃗_D; //Here, 'D' is flag, which means the decision class
6          output.collect(key', 1);
7          let key' = 'F'+ x⃗_{B∪D}; //Here, 'F' is flag, which means the association between the equivalence class and decision class
8          output.collect(key', 1);
9      end
10 end
```

Example 3. In Table 1, let $S_1 = (U_1, C \cup D, V, f)$ and $S_2 = (U_2, C \cup D, V, f)$ be two sub-decision tables of S, $U = U_1 \cup U_2$, where $U_1 = \{x_1, x_2, \ldots, x_6\}$ and $U_1 = \{x_7, x_8, \ldots, x_{12}\}$. We set thresholds $\alpha = 0.6$, $\beta = 0.2$ for Method 2 and $\alpha' = 0.1$, $\beta' = 0.1$ for Method 3.

Table 3 shows Map, Combine and Reduce phases in Step 1 of three parallel methods. Then, the accuracy and the coverage of rules are computed and rule sets are generated as shown in Table 3, where $\surd$ and $\times$ means it is a rule or not, respectively. Obviously, the number of rules are 4, 4, 5 for these three methods.

4. Experimental analysis

We implemented our proposed parallel large-scale knowledge acquisition on several representative MapReduce runtime systems: Hadoop [41], Phoenix [39] and Twister [11]. The experiments on Hadoop and Twister were carried out on Cheetah cluster [2] using 9 nodes, which contains 1 head node and 8 compute nodes. 4 of compute nodes have 16 GB main memory and use AMD Opteron(TM) Processor 2376 CPUs (8 cores in all, each has a clock frequency of 2.3 GHz). Another 4 nodes

Algorithm 2: Combine(*key*, *V*).

Input:
//*key:* the information set of the object with respect to the sets B, D and $B \cup D$
//V: a list of counts
Output:
//*key'*: the information set of the object with respect to the sets B, D and $B \cup D$
//*value'*: the count.

```
1 begin
2   let value' = 0 and key' = key;
3   for each v ∈ V do
4     value' = value' + v;
5   end
6   output.collect(key', value');
7 end
```

Algorithm 3: Reduce(*key*, *V*)

Input:
//*key:* the information set of the object with respect to the sets B, D and $B \cup D$
//V: a list of counts
Output:
//*key'*: the information set of the object with respect to the sets B, D and $B \cup D$
//*value'*: the count.

```
1 begin
2   let value' = 0 and key' = key;
3   for each v ∈ V do
4     value' = value' + v;
5   end
6   output.collect(key', value');
7 end
```

Table 2
Step 1 – Map, Combine, Reduce.

	S_1			S_2		
	$\|E\|$	$\|D\|$	$\|E \cap D\|$	$\|E\|$	$\|D\|$	$\|E \cap D\|$
Map	⟨*no*, *normal*⟩,1 ⟨*no*, *high*⟩,1 ⟨*yes*, *normal*⟩,1 ⟨*yes*, *veryhigh*⟩,1 ⟨*no*, *normal*⟩,1 ⟨*yes*, *high*⟩,1	⟨*no*⟩,1 ⟨*yes*⟩,1 ⟨*yes*⟩,1 ⟨*yes*⟩,1 ⟨*no*⟩,1 ⟨*no*⟩,1	⟨*no*, *normal*, *no*⟩,1 ⟨*no*, *high*, *yes*⟩,1 ⟨*yes*, *normal*, *yes*⟩,1 ⟨*yes*, *veryhigh*, *yes*⟩,1 ⟨*no*, *normal*, *no*⟩,1 ⟨*yes*, *high*, *no*⟩,1	⟨*no*, *normal*⟩,1 ⟨*no*, *high*⟩,1 ⟨*yes*, *high*⟩,1 ⟨*yes*, *veryhigh*⟩,1 ⟨*no*, *normal*⟩,1 ⟨*yes*, *high*⟩,1	⟨*no*⟩,1 ⟨*yes*⟩,1 ⟨*yes*⟩,1 ⟨*yes*⟩,1 ⟨*no*⟩,1 ⟨*yes*⟩,1	⟨*no*, *normal*, *no*⟩,1 ⟨*no*, *high*, *yes*⟩,1 ⟨*yes*, *high*, *yes*⟩,1 ⟨*yes*, *veryhigh*, *yes*⟩,1 ⟨*no*, *normal*, *no*⟩,1 ⟨*yes*, *high*, *yes*⟩,1
Combine	⟨*no*, *normal*⟩,2 ⟨*no*, *high*⟩,1 ⟨*yes*, *normal*⟩,1 ⟨*yes*, *veryhigh*⟩,1 ⟨*yes*, *high*⟩,1	⟨*no*⟩,3 ⟨*yes*⟩,3	⟨*no*, *normal*, *no*⟩,2 ⟨*no*, *high*, *yes*⟩,1 ⟨*yes*, *normal*, *yes*⟩,1 ⟨*yes*, *veryhigh*, *yes*⟩,1 ⟨*yes*, *high*, *no*⟩,1	⟨*no*, *normal*⟩,2 ⟨*no*, *high*⟩,1 ⟨*yes*, *high*⟩,2 ⟨*yes*, *veryhigh*⟩,1	⟨*no*⟩,2 ⟨*yes*⟩,4	⟨*no*, *normal*, *no*⟩,2 ⟨*no*, *high*, *yes*⟩,1 ⟨*yes*, *high*, *yes*⟩,2 ⟨*yes*, *veryhigh*, *yes*⟩,1

	$\|E\|$	$\|D\|$	$\|E \cap D\|$
Reduce	⟨*no*, *normal*⟩,4 ⟨*no*, *high*⟩,2 ⟨*yes*, *normal*⟩,1 ⟨*yes*, *veryhigh*⟩,2 ⟨*yes*, *high*⟩,3	⟨*no*⟩,5 ⟨*yes*⟩,7	⟨*no*, *normal*, *no*⟩,4 ⟨*no*, *high*, *yes*⟩,2 ⟨*yes*, *normal*, *yes*⟩,1 ⟨*yes*, *veryhigh*, *yes*⟩,2 ⟨*yes*, *high*, *no*⟩,1 ⟨*yes*, *high*, *yes*⟩,2

have 8 GB main memory and use Intel Xeon CPU E5410, comprising two Quad-Core CPUs (8 cores in all, each has a clock frequency of 2.33 GHz). Since Phoenix is based on multi-core, the experiments on Phoenix were conducted on single compute node, which has 64 GB main memory and use AMD Opteron(TM) Processor 6272 CPUs (64 cores in all, each has a clock frequency of 2.1 GHz). The operating system in these nodes is Linux CentOS 6.2 Kernel 2.6.32. Hadoop-1.0.1, phoenix++-1.0, twister-0.9 are used as MapReduce runtime systems. The detailed description is listed in Table 4. In the following subsections, we compare performance using 1, 2, 4, 8, 16 and 32 cores.

Table 3
Step 2 – Accuracy, Coverage, Rules.

$E \to D$	$Acc(D\|E)$	$Cov(D\|E)$	Method 1	Method 2	Method 3
$\langle no, normal\rangle \to \langle no\rangle$	$4/4 = 1.00$	$4/5 = 0.80$	$\surd$	$\surd$	$\surd$
$\langle no, high\rangle \to \langle yes\rangle$	$2/2 = 1.00$	$2/7 = 0.29$	$\surd$	$\surd$	$\surd$
$\langle yes, normal\rangle \to \langle yes\rangle$	$1/1 = 1.00$	$1/7 = 0.14$	$\surd$	$\times$	$\surd$
$\langle yes, veryhigh\rangle \to \langle yes\rangle$	$2/2 = 1.00$	$2/7 = 0.29$	$\surd$	$\surd$	$\surd$
$\langle yes, high\rangle \to \langle no\rangle$	$1/3 = 0.33$	$1/5 = 0.20$	$\times$	$\times$	$\times$
$\langle yes, high\rangle \to \langle yes\rangle$	$2/3 = 0.67$	$2/7 = 0.29$	$\times$	$\surd$	$\surd$

Table 4
A description of MapReduce runtime systems.

	Runtime	Version	Programming Language	Run on
1	Hadoop	hadoop-1.0.1	Java, jdk1.6.0.12	Cheetah cluster
2	Phoenix	phoenix++-1.0	C++, g++4.4.6	64-core computer
3	Twister	twister-0.9	Java, jdk1.6.0.12	Cheetah cluster

Table 5
A description of data sets.

	Data sets	Samples	Features	Classes	Size
1	KDD99	4,898,421	41	23	0.48 GB
2	Weka-1.8G	32,000,000	10	35	1.80 GB
3	Weka-3.2G	40,000,000	15	45	3.20 GB
4	Weka-6.4G	80,000,000	15	58	6.40 GB

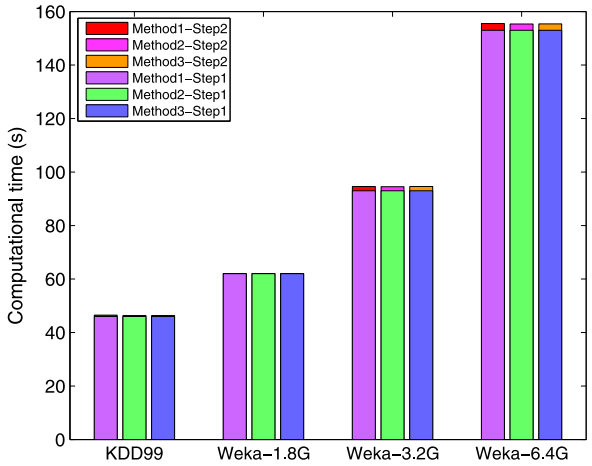

Fig. 3. Computational time of 32 cores.

4.1. Data sets

We utilized the large data set KDD99 from the machine learning data repository, University of California at Irvine [32], which consists of approximately five million records. Each records consists of 1 decision attribute and 41 condition attributes, where 6 are categorical and 35 are numeric. Since our method can only process categorical attributes, we discretize the 35 numeric attributes firstly. In addition, three synthetic data sets have been generated by means of the WEKA data generator [16]. The data sets are outlined in Table 5. The data sets KDD99, Weka-1.8G, Weka-3.2G and Weka-6.4G are split into 64, 64, 128 and 256 blocks, respectively, in Hadoop. All the data sets are split into 64 blocks in Twister and not split in Phoenix.

4.2. Performance comparison on different MapReduce runtime systems

4.2.1. A comparison of computational time

We executed the aforementioned three kinds of knowledge acquisition methods on Hadoop MapReduce system. Fig. 3 shows the computational times of Step 1, Step 2 and total over the three parallel methods on 32 cores. Here, Methods 1–3 correspond to Definitions 5–7. It shows that Step 2 only costs very little time. Moreover, three parallel methods almost cost

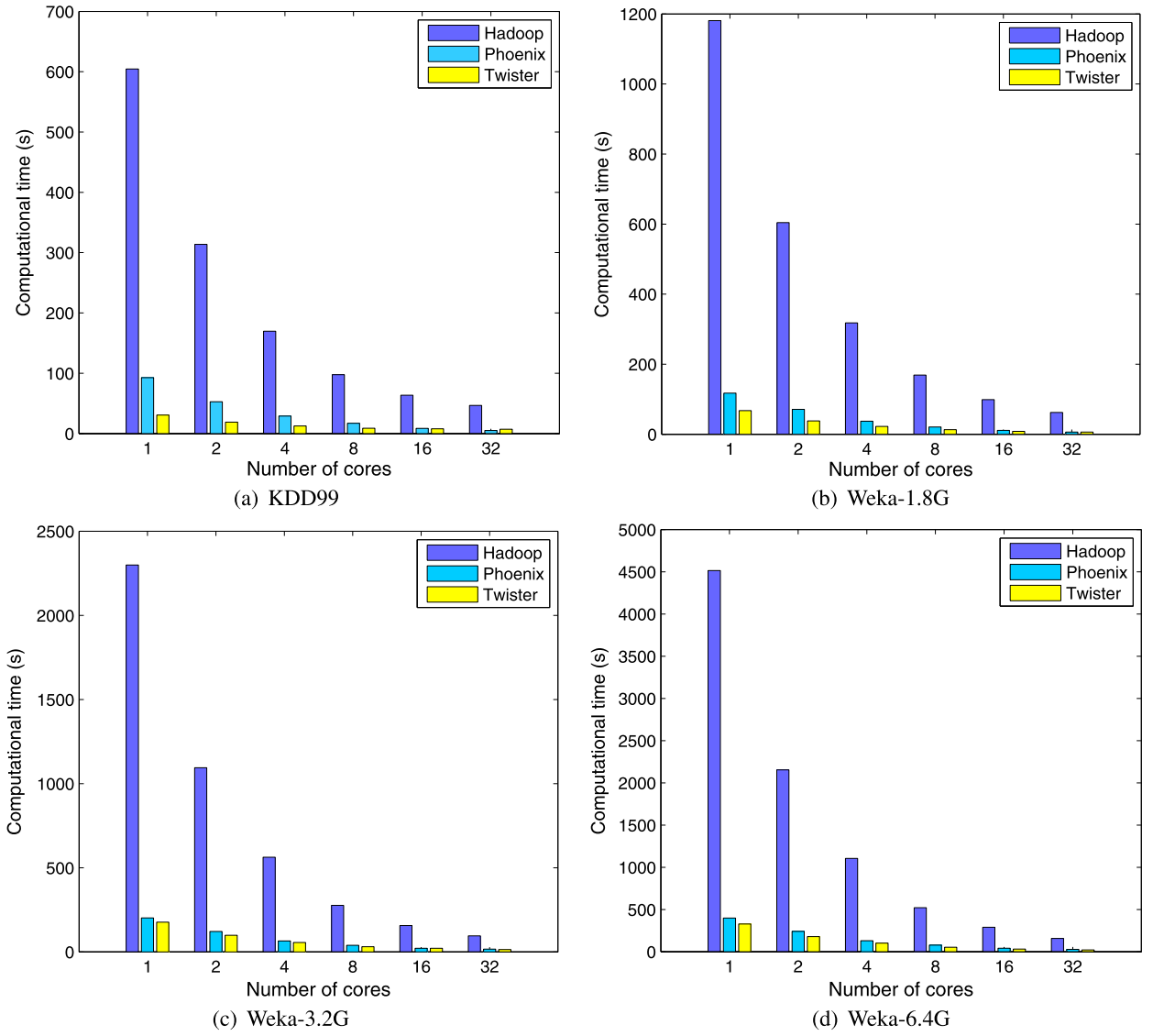

Fig. 4. Computational time on different MapReduce runtime systems.

the same time in Step 1, Step 2 and total, respectively. It is also demonstrated in another two MapReduce systems Phoenix and Twister. Because of that, we merely discuss about the performance of one method in the following evaluations.

Fig. 4 shows the computational time of the parallel method with different cores on different runtime systems. As the number of the cores increases, the computational time of the parallel methods becomes smaller.

It is easy to see that Hadoop always takes the most time to finish. The computational time of the parallel method on Hadoop is 5.x–11.x times and 6.x–19.x times than that on Phoenix and Twister, respectively. Moreover, in most cases, Twister is faster than Phoenix.

4.2.2. Speedup comparison

Further, we examined the speedup characteristic of the proposed parallel methods. To measure the speedup, we kept the data set constant and increased the number of cores in the system. Speedup given by the larger system is defined by the following formula [43]:

$$Speedup(p) = \frac{T_1}{T_p}$$

where p is the number of cores, T_1 is the execution time on a single core, and T_p is the execution time on p cores.

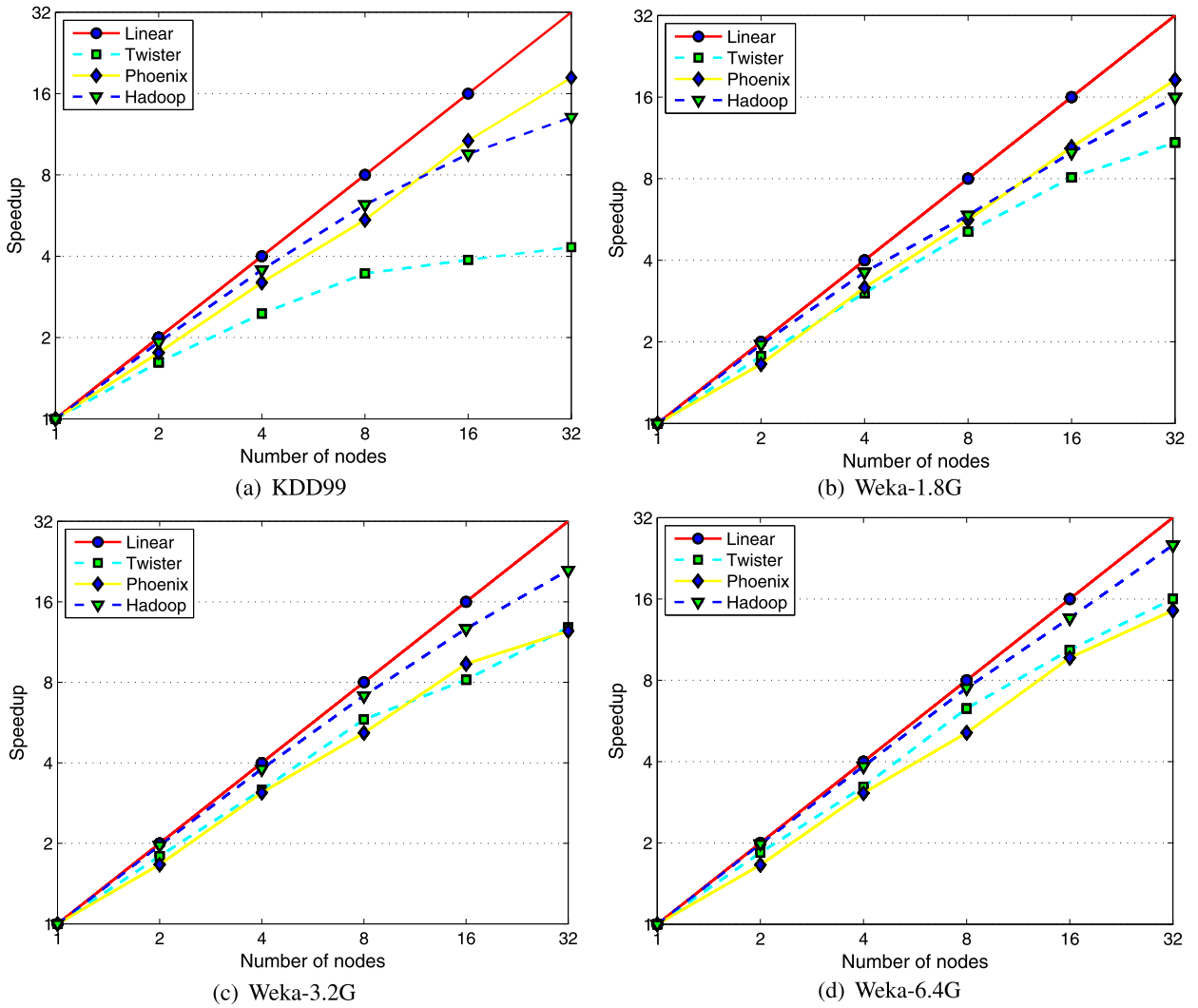

Fig. 5. Speedup of different MapReduce runtime systems.

The ideal parallel algorithm demonstrates linear speedup: a system with p times the number of cores yields a speedup of p. However, linear speedup is difficult to achieve because the communication cost increases with the increasing number of cores.

We performed the speedup evaluation on different data sets. The number of processors varies from 1 to 32. Fig. 5 shows the speedup for over all data sets for different processing cores on different runtime systems. As the result shows, in the smallest data set KDD99, Phoenix has a best speedup as the number of cores increases. Specially, the speedup of Twister on KDD99 performs worst, and it is only 4.3 times on 32 cores. In the median size data set Weka-1.8G, the speedup of Phoenix performs still the best. But that of Hadoop is much closer to it. Moreover, the speedup of Twister has a good ascension. In the larger data sets Weka-3.2G and Weka-6.4G, Hadoop has a best speedup, and that is growing closer to the linear speedup as the size of the data set increases. Contrast to this, the speedup of Phoenix becomes a little smaller. Twister performs better than Phoenix on Weka-6.4G.

In fact, since our cluster is heterogeneous with different kinds of compute nodes and we always run experiments on 1 core using the better compute node, the proposed parallel methods would have a better speedup in a homogeneous setting.

4.3. Discussion

According to the above performance comparison, we find that the computational time of the same application on different runtime systems are different. Take Hadoop [41] as example, the input data is firstly uploaded to the Hadoop Distributed File System (HDFS), which is partitioned by a specified size (64 MB at default). HDFS is designed to reliably store very large files across machines in a large cluster and the data is replicated for fault tolerance. Hadoop provides a pluggable MapReduce

scheduler that provides a way to share large clusters. The default scheduling mode in Hadoop is First-In-First-Out (FIFO), which is used to schedule the jobs in our experiments. Normally, a Hadoop application consists of one or more Hadoop jobs. Our application consists of only one job. When running a Hadoop job, it allocates resource on the basis of the Hadoop scheduler. After that, Map workers and Reduce workers are launched and start to work. The intermediate data produced by Map is written to a local file system first. These processes increase the running time on Hadoop. Indeed, it would cost a few seconds even if the input data is empty on Hadoop. Therefore, the application on Hadoop is always slow if the data is not enough big.

To the contrary, Twister and Phoenix have not the distributed file system. In Twister [11], users partition the data manually or automatically by a specified script, and send them to different compute nodes. Then, it generates a configuration file to tell the compute nodes to process the local data in Map phase. Unlike Hadoop, to achieve better performance, Twister handles the intermediate data in the distributed memory of the worker nodes. Meanwhile, the use of publish/subscribe messaging infrastructure improves the efficiency of Twister runtime and Twister schedules Map and Reduce tasks statically. Twister provides fault tolerance support only for iterative MapReduce computations rather than non-iterative MapReduce computations. These handling methods in Twister make it perform better than Hadoop but with worse fault tolerance.

For Phoenix [39], it is a MapReduce framework for shared-memory CMPs and SMPs. It does not need to send data to other nodes because it is designed for a single node. Naturally, there is no communication cost. Therefore, it always performs the best if the input data can be loaded into the memory. In our experiment, it performs a little worse than Twister. The possible reason includes two aspects: (a) The clock frequency of CPUs for Phoenix is lower than that for Twister. (b) The input data for Phoenix does not need to be partitioned before testing, which means Phoenix does split the data itself. The input data for Twister needs to be partitioned before testing. It may reduce the running time for Twister.

According to the above speedup comparison, Hadoop has a better speedup as the size of data set increases. Combined with its good fault tolerance, Hadoop can help to process very large-scale data. Because of its very good support, users can easily deploy Hadoop on local cluster and Public Cloud, such as Amazon EC2 and Microsoft Azure both support Hadoop by using Amazon Elastic MapReduce [1] and HadoopOnAzure [4].

The application on Phoenix is fast, and has an excellent speedup when the size of data size is not large. Because of its characteristic, it is more suitable for smaller data sets on single multi-cores compute node. If the size of data set is larger than memory, Phoenix would be failed with an out of memory error.

The application on Twister has only a mediocre speedup in our experiment. Because Twister is suitable for iterative applications by using iMapReduce, but our application here is not iterative. And yet for all that, compared with Hadoop and Phoenix, the program on Twister is mostly fastest. It also has a Public Cloud version: Twister4Azure [15].

5. Conclusions

Large-scale data mining has been a new challenge in recent years. Traditional rough set based methods for knowledge discovery fail to process the enlarging data in applications. In this paper, we designed the rough set based method for knowledge acquisition using MapReduce, implemented it on different runtime systems, and compared their performance. The computational time comparison showed that the parallel method on Twister almost costed the least time, and the computational time on Hadoop was always multiple times than that on Phoenix and Twister. The speedup comparison showed that the parallel method on Phoenix over smaller data sets and that on Hadoop over larger ones had a better speedup. It both demonstrated that our proposed parallel large-scale methods could effectively process very large data sets. One of our future research work to develop a rough set based large-scale knowledge discovery system. Another is to focus on unstructured data processing by using rough set theory and MapReduce techniques.

Acknowledgements

This work is supported by the National Science Foundation of China (Nos. 60873108, 61175047, 61100117) and NSAF (No. U1230117), the Fundamental Research Funds for the Central Universities (No. SWJTU11ZT08), the Research Fund of Traction Power State Key Laboratory, Southwest Jiaotong University (No. 2012TPL_T15), the Science and Technology Planning Project of Sichuan Province (No. 2012RZ0009), China and the Fostering Foundation for the Excellent Ph.D. Dissertation of Southwest Jiaotong University (No. 2012ZJB), China.

References

[1] Amazon Elastic MapReduce, http://aws.amazon.com/elasticmapreduce/.
[2] Cheetah cluster in Georgia State University, http://cs.gsu.edu/?q=cheetah.
[3] Daytona, http://research.microsoft.com/en-us/projects/daytona/.
[4] HadoopOnAzure, https://www.hadooponazure.com/.
[5] S. Blanas, J.M. Patel, V. Ercegovac, J. Rao, E.J. Shekita, Y. Tian, A comparison of join algorithms for log processing in mapreduce, in: Proceedings of the 2010 International Conference on Management of Data, SIGMOD '10, ACM, New York, NY, USA, 2010, pp. 975–986.
[6] Y. Bu, B. Howe, M. Balazinska, M.D. Ernst, Haloop: efficient iterative data processing on large clusters, Proc. VLDB Endow. 3 (2010) 285–296.
[7] Y. Cheng, The incremental method for fast computing the rough fuzzy approximations, Data Knowl. Eng. 70 (2011) 84–100.
[8] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, in: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, vol. 6, OSDI'04, USENIX Association, Berkeley, CA, USA, 2004, p. 10.

[9] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (2008) 107–113.
[10] S. Dick, A. Tappenden, C. Badke, O. Olarewaju, A granular neural network: Performance analysis and application to re-granulation, Int. J. Approx. Reason. 54 (2013) 1149–1167.
[11] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.H. Bae, J. Qiu, G. Fox, Twister: a runtime for iterative mapreduce, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, ACM, New York, NY, USA, 2010, pp. 810–818.
[12] A. Ene, S. Im, B. Moseley, Fast clustering using mapreduce, in: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '11, ACM, New York, NY, USA, 2011, pp. 681–689.
[13] J.W. Grzymala-Busse, W. Ziarko, Data mining and rough set theory, Commun. ACM 43 (2000) 108–109.
[14] T. Gunarathne, T.L. Wu, J. Qiu, G. Fox, Mapreduce in the clouds for science, in: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, CLOUDCOM '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 565–572.
[15] T. Gunarathne, B. Zhang, T.L. Wu, J. Qiu, Portable parallel programming on cloud and hpc: Scientific applications of twister4azure, in: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, pp. 97–104.
[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: an update, ACM SIGKDD Explor. Newsl. 11 (2009) 10–18.
[17] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, 3rd edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.
[18] B. He, W. Fang, Q. Luo, N.K. Govindaraju, T. Wang, Mars: a mapreduce framework on graphics processors, in: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT '08, ACM, New York, NY, USA, 2008, pp. 260–269.
[19] Q. Hu, W. Pedrycz, D. Yu, J. Lang, Selecting discrete and continuous features based on neighborhood decision error minimization, IEEE Trans. Syst. Man Cybern. Part B. Cybern. 40 (2010) 137–150.
[20] Q. Hu, Z. Xie, D. Yu, Hybrid attribute reduction based on a novel fuzzy-rough model and information granulation, Pattern Recognit. 40 (2007) 3509–3521.
[21] Q. Hu, D. Yu, J. Liu, C. Wu, Neighborhood rough set based heterogeneous feature subset selection, Inf. Sci. 178 (2008) 3577–3594.
[22] K. Kaneiwa, A rough set approach to mining connections from information systems, in: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10, ACM, New York, NY, USA, 2010, pp. 990–996.
[23] Y. Leung, M.M. Fischer, W.Z. Wu, J.S. Mi, A rough set approach for the discovery of classification rules in interval-valued information systems, Int. J. Approx. Reason. 47 (2008) 233–246.
[24] Y. Leung, W.Z. Wu, W.X. Zhang, Knowledge acquisition in incomplete information systems: A rough set approach, Eur. J. Oper. Res. 168 (2006) 164–180.
[25] H. Li, M. Wang, X. Zhou, J. Zhao, An interval set model for learning rules from incomplete information table, Int. J. Approx. Reason. 53 (2012) 24–37.
[26] T. Li, D. Ruan, W. Geert, J. Song, Y. Xu, A rough sets based characteristic relation approach for dynamic attribute generalization in data mining, Knowl.-Based Syst. 20 (2007) 485–494.
[27] J. Liang, F. Wang, C. Dang, Y. Qian, An efficient rough feature selection algorithm with a multi-granulation view, Int. J. Approx. Reason. 53 (2012) 912–926.
[28] G. Lin, Y. Qian, J. Li, Nmgrs: Neighborhood-based multigranulation rough sets, in: Selected Papers – Uncertain Reasoning at FLAIRS 2010, Int. J. Approx. Reason. 53 (2012) 1080–1093.
[29] J. Lin, M. Schatz, Design patterns for efficient graph algorithms in mapreduce, in: Proceedings of the Eighth Workshop on Mining and Learning with Graphs, MLG '10, ACM, New York, NY, USA, 2010, pp. 78–85.
[30] D. Liu, T. Li, D. Ruan, J. Zhang, Incremental learning optimization on knowledge discovery in dynamic business intelligent systems, J. Glob. Optim. 51 (2011) 325–344, http://dx.doi.org/10.1007/s10898-010-9607-8.
[31] R.K. Menon, G.P. Bhat, M.C. Schatz, Rapid parallel genome indexing with mapreduce, in: Proceedings of the second international workshop on MapReduce and its applications, MapReduce '11, ACM, New York, NY, USA, 2011, pp. 51–58.
[32] D. Newman, S. Hettich, C. Blake, C. Merz, UCI Repository of Machine Learning Databases, University of California, Department of Information and Computer Science, Irvine, CA, 1998, http://archive.ics.uci.edu/ml/.
[33] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, Manning Publications Co., Greenwich, CT, USA, 2011.
[34] Z. Pawlak, J. Grzymala-Busse, R. Slowinski, W. Ziarko, Rough sets, Commun. ACM 38 (1995) 88–95.
[35] Y. Qian, J. Liang, W. Pedrycz, C. Dang, Positive approximation: An accelerator for attribute reduction in rough set theory, Artif. Intell. 174 (2010) 597–618.
[36] Y. Qian, J. Liang, W. Pedrycz, C. Dang, An efficient accelerator for attribute reduction from incomplete data in rough set framework, Pattern Recognit. 44 (2011) 1658–1670.
[37] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis, Evaluating mapreduce for multi-core and multiprocessor systems, in: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, HPCA '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 13–24.
[38] R. Susmaga, Parallel computation of reducts, in: L. Polkowski, A. Skowron (Eds.), Rough Sets and Current Trends in Computing, in: Lect. Notes Comput. Sci., vol. 1424, Springer, Berlin, Heidelberg, 1998, pp. 450–458.
[39] J. Talbot, R.M. Yoo, C. Kozyrakis, Phoenix++: modular mapreduce for shared-memory systems, in: Proceedings of the second international workshop on MapReduce and its applications, MapReduce '11, ACM, New York, NY, USA, 2011, pp. 9–16.
[40] S. Tsumoto, Automated extraction of medical expert system rules from clinical databases based on rough set theory, Inf. Sci. 112 (1998) 67–84.
[41] T. White, Hadoop: The Definitive Guide, 2nd edition, O'Reilly Media/Yahoo Press, 2010.
[42] S. Wong, W. Ziarko, R. Ye, Comparison of rough-set and statistical methods in inductive learning, Int. J. Man-Mach. Stud. 25 (1986) 53–72.
[43] X. Xu, J. Jäger, H.P. Kriegel, A fast parallel clustering algorithm for large spatial databases, Data Min. Knowl. Disc. 3 (1999) 263–290, http://dx.doi.org/10.1023/A:1009884809343.
[44] T. Yang, Q. Li, Reduction about approximation spaces of covering generalized rough sets, Int. J. Approx. Reason. 51 (2010) 335–345.
[45] Y. Yao, Probabilistic rough set approximations, Int. J. Approx. Reason. 49 (2008) 255–271.
[46] J. Zhang, T. Li, Y. Pan, Parallel rough set based knowledge acquisition using mapreduce from big data, in: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine '12, ACM, New York, NY, USA, 2012, pp. 20–27.
[47] J. Zhang, T. Li, D. Ruan, Z. Gao, C. Zhao, A parallel method for computing rough set approximations, Inf. Sci. 194 (2012) 209–223.
[48] J. Zhang, T. Li, D. Ruan, D. Liu, Rough sets based matrix approaches with dynamic attribute variation in set-valued information systems, Int. J. Approx. Reason. 53 (2012) 620–635.
[49] J. Zhang, D. Xiang, T. Li, Y. Pan, M2M: A simple Matlab-to-MapReduce translator for cloud computing, Tsinghua Sci. Technol. 18 (2013) 1–9.
[50] X. Zhang, D. Miao, Two basic double-quantitative rough set models of precision and grade and their investigation using granular computing, Int. J. Approx. Reason. (2013) 1130–1148.