Journal of Database Management
Volume 29 • Issue 2 • April-June 2018

A Novel Approach to Distributed Rule Matching and Multiple Firing Based on MapReduce

Tianyang Dong, College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China
Qiang Cheng, College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China
Bin Cao, College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China
Jianwei Shi, College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China

ABSTRACT

In order to solve the poor performance problem of massive rules reasoning, as well as the inconsistency problem of working memory in distributed rule matching, this article presents the formal definition of interference relations between rules, and proposes a novel approach to distributed rule matching and multiple firing based on MapReduce. This approach adopts the way of access request control to detect and exclude interference rules, then selects several rule instantiations to perform multiple firing and concurrent execution, thus reducing the number of inference cycles effectively. By detecting the interferences between rules, this method selects and executes compatible rule sets, and avoids the inconsistency problem of system working memory. In order to verify the validity of the authors' approach, this article developes a production system based on MapReduce, and applied this approach in the master server of a distributed production system. The experimental results show that their method can promote the performance of massive rules reasoning effectively.

KEYWORDS

Access Request Control, Big Data, Cloud Computing, Mapreduce, Multiple Rule Firing, Rule Matching

INTRODUCTION

Production system, as known as rule engine, is a common way to build expert systems (Giarratano & Riley, 2005). It has been widely used in some fields such as business, science, engineering, manufacturing, and medicine. The production system can adapt to the changing requirements of enterprise information systems, and reduce almost 10% cost of the information systems for enterprises and organizations (Garnter, 2002a; Garnter, 2002b). The agility and easy-to-use of systems promote the development of business rule engine (Kaul, Storey, & Woo, 2017; Batra, 2017). The market penetration ratio of business rule technology is just 20% in 2000, and now it have reached about 80%. However, compared with other types of information systems, production system usually has the problem of poor performances. It is even worse for the massive rules reasoning. Grid and cloud computing have

DOI: 10.4018/JDM.2018040104

changed the IT landscape in the way we access and manage IT infrastructures (Li et al.,2015; Lin et al., 2015; Jianwei et al., 2015; Yuyu et al., 2016). Nowadays, both the cloud computing paradigm and MapReduce programming framework have become key enablers for running big data analytics and large-scale compute- and data intensive applications (Palanisamy et al., 2015; Lee et al., 2016; Qi et al., 2015; Eldawy et al., 2016). The way of using cluster or cloud to construct production systems (Petcu, 2005a) can flexibly expand system processing capability by increasing the cluster scale. That can effectively respond to the challenges of massive rule processing.

In order to improve the performance of production system for massive rules reasoning, as well as keeping the consistency of working memory in distributed rule matching, this paper presents the formal definitions of interference relations between rules, and proposes a novel approach to distributed rule matching and multiple firing based on MapReduce. This approach adopts the way of access request control to detect and exclude interference rules, then selects several rule instantiations to perform multiple firing and concurrent execution, thus reducing the number of inference cycle effectively. Through detecting interferences between rules, this approach can select and execute compatible rule set, and finely avoid the inconsistency problem of the system working memory. In order to verify the validity of this approach, this paper developed a distributed production system based on MapReduce, which is known as a rule engine for traffic information service. The approach to distributed rule matching and multiple firing is deployed in the master server of this distributed production system. The experimental results and successful applications in the public-travel traffic information service system show that the approach can effectively improve the performance of massive rules reasoning.

RELATED WORKS

The researches on multiple rule firings are mostly devoted to centralized production system. They try to select as many rules as possible to execute, in order to break the constraints of selecting only one rule caused by the traditional conflict resolution strategies. But there are all kinds of dependencies among the rules, which will lead to the inconsistency problem of working memory after multiple rule execution. So it needs to find out the interferences to be excluded according to the inter-rule dependencies, select compatible rule set to execute, and guarantee the consistency of working memory in the case of multiple rule firings. Ishida (1991) used data dependency graph to represent the referring and processing relations between rules and working memory elements, identified and excluded interference rules by using the corresponding selection algorithm. Schmolze et al. (1992) proposed A2 algorithm to exclude rule instantiations with interference characteristic of disabling and clashing. Kuo et al. (1991) selected compatible rules based on the RTC interference matrix.

The key of parallel rule firing in centralized production system is how to mine the association rules. Therefore, Schmolze (1991) presented a formal solution to the problem of guaranteeing serializable behavior in synchronous parallel production systems that execute many rules simultaneously. Rand et al. (1996) presented an algorithms to explore a spectrum of trade-offs between computation, communication, memory usage, synchronization, and the use of problem specific information. This method solves the problem of mining association rules on a shared nothing multiprocessor. Han et al. (2000) proposed two parallel algorithms for mining association rules: intelligent data distribution algorithm and hybrid distribution algorithm. The hybrid distribution algorithm further improves upon the intelligent data distribution algorithm by dynamically partitioning the candidate set to maintain good load balance. Thabtah et al. (2006) proposed a new associative classification method called Multi-class Classification based on Association Rules (MCAR), which takes advantage of vertical format representation and uses an efficient technique for discovering frequent items. Bădică et al. (2011) surveyed several major approaches using rules in multi-agent systems and distributed agent architectures which run rule engines at their core. Gupta et al. (1988) exploited very fine-grained parallelism to achieve significant speed-ups, which is also distinct from other parallel implementations in that they parallelize a highly optimized C-based implementation of OPS5. It is 10-20 times faster than the standard lisp implementation distributed by Carnegie Mellon University. These methods break the interference cycle in several places, which make the selected compatible rule set have a relative small amount of rule instantiations (Perraju et al., 2000; Cheng et al., 2010; Kao, 2012; Wang et al., 2011). Therefore, it is necessary to get further promotion because of the small amount of parallelism.

Recently, more and more cluster or cloud technologies are adopted to construct production systems for the sake of expanding the system processing capability. The existing cluster-based solutions can be mainly divided into two categories as follows. The first way is to partition and allocate the rule base in rule level (Petcu, 2005a; Wu et al., 2008; Cabitza et al., 2005; Wu et al., 2010; Wu, 2011), which is focused on the parallel execution of rules. The production system instances in the cluster systems of Petcu et al. (2005b) and Schmolze et al. (1992) are peer-to-peer, which don't carry out global selection and execution of rules; each engine instance uses its own rule base to reason in the local working memory, depending on the central server or parallel libraries such as Octopus (Petcu et al., 2005b), JPVM (Petcu, 2005b), and MPI (Wu et al., 2008) to realize asynchronous communications with each other. Cabitza et al. (2005) and Wu et al. (2008, 2010) adopted the master-slave structure to organize processing nodes in the clusters. The master server in the structure of Cabitza (2005) is only responsible for managing processing nodes, and each processing node performs parallel reasoning independently in the local partial copy of the global working memory; the synchronization of multiple rule execution is realized by lock mechanism combining with the ghost facts. The master server in the structure of Wu et al. (2008, 2010) is not only in charge of nodes management and load balancing, but also needs to process the intermediate results of the computing nodes according to the reduced rules (Wu et al., 2008), or select the best solution of the domain problems by using the priorities of these rules. The above approaches are mostly for specific domains. What's more, they are lack of necessary versatility and flexibility. For much larger scale of rule sets involving in multiple application areas, these systems have to be modified. Wu et al. (2010) have proposed a Grid-enabled parallel CLIPS language and a dynamic load balancing programming mode to address the problem that CLIPS suffers from long execution time because of the characteristics of rule-based language. It can parallelize the execution of a CLIPS program automatically if the data can be inferred independently. The second method is to partition based on condition element level (Bin et al., 2010). The method proposed by Cao et al. (2010) performs more fine-grained rule decomposition aiming to realize parallel rule matching based on MapReduce programming model, which makes the system obtain a more balanced load. This method is not for the specific domain problem, and has more versatility and scalability, which has some advantages compared to the above solutions. But after the reduce stage, the master server (Cao et al., 2010) uses some conflict resolution strategies to select and execute just one rule instantiation. When there is a large amount of rule instantiations in the agenda of the master, it will take a long time to complete the reasoning cycle with the continuous increase of the number of cycles. It will be a bottleneck of system performance. It is urgent to combine with multiple rule firings to solve this problem.

MapReduce is a framework for processing highly distributable problems across huge datasets. Cao et al. (2010) pointed out that MapReduce is a highly effective and efficient tool for large-scale data analysis, which has many significant advantages over parallel data-bases. Yang et al. (2007) improved MapReduce into a new model called Map-Reduce-Merge, which is a simplified relational data processing on large clusters. MapReduce can achieve better performance with the allocation of more computing nodes from the grid or cloud to speed up computation (Jiang et al., 2010). Therefore, Srirama et al. (2012) proposed the adapting scientific computing problems to clouds by using MapReduce, and showed how to adapt algorithms from each class into the MapReduce model, what affects the efficiency and scalability of algorithms in each class. Biao et al. (2012) introduced a distributed approach for performing formal concept mining. The novelty of this method is using a light-weight MapReduce runtime called Twister that is better suited to iterative algorithms than recent distributed approaches. Pallickara et al. (2009) described Granules, a lightweight, streaming-based runtime for cloud computing which incorporates support for the MapReduce framework. It can provide rich lifecycle support for developing scientific applications with support for iterative, periodic and data driven semantics for individual computations and pipelines. Xiao et al. (2011) proposed a high-integrity feature to MapReduce computation using speculative execution. The key idea of this approach is selectively replicating MapReduce tasks on a random computation node, and comparing the hash of the execution results to determine if the integrity of the task is compromised.

With the wide applications of MapReduce, some approaches to scalable reasoning by using MapReduce are proposed. Urbani et al. (2009) presented a method of scalable distributed reasoning by using MapReduce, which can consider the OWL Horst semantics and efficiently compute the closure under the RDFS semantics. Urbani et al. (2010) proposed WebPIE, a parallel approach based on MapReduce which vastly outperforms state-of-the-art approaches and scales linearly with input size. But a drawback of this system is that it does batch processing, i.e. every time we execute a rule all the input must be read. Liu et al. (2011) proposed an algorithm for fuzzy pD* reasoning by separately considering fuzzy D rules and fuzzy p rules. Hu et al. (2010) presents the design of the Better Life 2.0 framework, which facilitates implementation of large-scale social intelligence application in cloud environment. It is based on the case-based reasoning technique for its additive knowledge space growing, and MapReduce framework for its large scale processing capability on cloud.

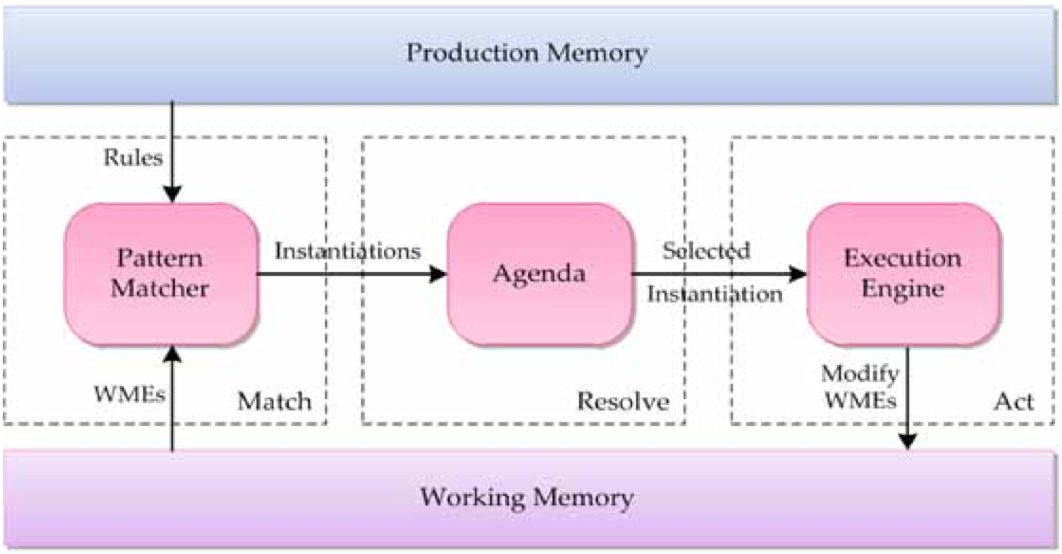
Compared with these MapReduce-based production systems, our system is different in three aspects. Firstly, we use different representations of knowledge for different purposes of knowledge manipulation. The existing rule systems (Urbani et al., 2009; Urbani et al., 2010) usually use MapReduce to realize large-scale RDF/OWL reasoning for materializing the closure of RDF/OWL. This kind of MapReduce-based rule system concerns on the semantic knowledge reasoning, which uses the existing triple rules to get implicit and new triples. In our system, the production rules are adopted to represent domain knowledge. Wu et al. (2010) and Cao et al. (2010) also used production rules to represent domain knowledge, but they just focused on distributed rule matching with large scale of facts. Obviously, the system will trigger different processing of facts according to the LHS matching situation of production rules. Secondly, the granularities of knowledge decomposition are different. We decompose and reconstruct the production rules into sub-rules according to LHSs. Wu et al. (2010) just divided production rules into mapping rules and reduce rules without any decomposition. Thirdly, the processing phases of MapReduce in these production systems are different. We use MapReduce to construct clustered-based production cycle with five phases. However, Wu et al. (2010) used the map and reduce two phases to generate the final results without any cycle. Urbani (2009, 2010) adopted four MapReduce jobs to finish the computing the RDF/OWL closure.

INTERFERENCES BETWEEN RULES

The production system mainly consists of three components. The first component is the Working Memory (WM) containing a set of assertions, which are called Working Memory Element (WME). The second component is a production rule base called Production Memory (PM). The rules in Working Memory Element have two parts: the Left-Hand-Side (LHS) containing a set of patterns, and the Right-Hand-Side (RHS) containing a series of actions. The last one is the most important part Inference Engine, which has three modules including Pattern Matcher, Agenda and Execution Engine. As shown in Figure 1, the execution engine of the production system performs the following Match-Resolve-Act Circle.

The pattern matcher uses the WMEs from WM to match the rules from PM. When the rule matching phase completes, a serial of rule instantiations will be created and added to the agenda. In resolve phase, the traditional production systems select just one suitable rule instantiation according to a certain conflict resolution strategy. After introducing multiple rule firing methods, the production system will select rule instantiations as many as possible in resolve phase, and concurrently execute them in act phase. It can increase the number of working memory changes per

Figure 1. The structure and inference cycle of the production system



inference cycle and reduce the total number of inference cycle, and thus promoting the performance of the production systems.

The parallel execution results of multiple rule instantiations can be different from any sequential execution results of the corresponding set of rule instantiations, which makes the production system reach incorrect state. The main reason is that the interference occurs between multiple firing rules. For instance, considering the following two rules in public-travel traffic information service:

Rule1: When: Vehicle.speed > 60; Road.level == Road.EXPRESS_WAY;
Then: Traffic.runingState = Traffic.VERY_SMOOTH;
Rule2: When: Traffic.runningState == Traffic.BASICALLY_SMOOTH; Road.level == Road. EXPRESS_WAY;
Then: Vechicle.speed = Math.random()*10 + 30;

Assuming that Rule1 and Rule2 are both matched successfully and only one of them can be selected to fire, two kinds of execution results will appear. If Rule1 is selected and executed, it will update the attribute c_2 of some facts with C type, which makes the matched Rule2 instantiation invalid and to be deleted, and vice versa. If the production system adopts multiple rule firing policy to realize the concurrently execution of Rule1 and Rule2, there will be no equivalent sequential execution order, which results in the inconsistencies of WMEs. It is to say that interference occurs between Rule1 and Rule2.

Interference between the rules exists in inter-rule dependencies. Therefore, it can extract the interference relation on the basis of analyzing the dependencies between rules, detect and exclude the interference, select compatible rule set to realize correct multiple rule firing. The process of identifying interference rules is called interference analysis.

The interference of rule R_i and R_j exists in the combinations of the condition part and action part of the two rules, it is necessary to firstly analyze these two composition parts of a production rule. The action part of a rule is used to add, delete, and modify WMEs. The modifying operation is equivalent to a deleting operation with an addition of an adding operation. In order to conveniently analyze the interference of the rules, this paper gives the formal definitions of pattern element and action element as follows:

Definition 1 (Positive/Negative Pattern Element): A positive pattern element in the LHS of the production rule Ri is used to check whether there is a corresponding working memory element Wk in WM, denoted as Wk ⊳ P+(Ri). A negative pattern element in the LHS of the rule is used to check whether there is no corresponding working memory element Wk in WM, denoted as Wk ⊳ P-(Ri).

Definition2 (Positive/Negative Action Element): An action element in the RHS of the production Ri which adds a working memory element Wk to the WM, is called positive action element, denoted as A+(Ri) ⊳ Wk. The action element used to delete a working memory element Wk from the WM, is called negative action element, denoted as A-(Ri) ⊳ Wk.

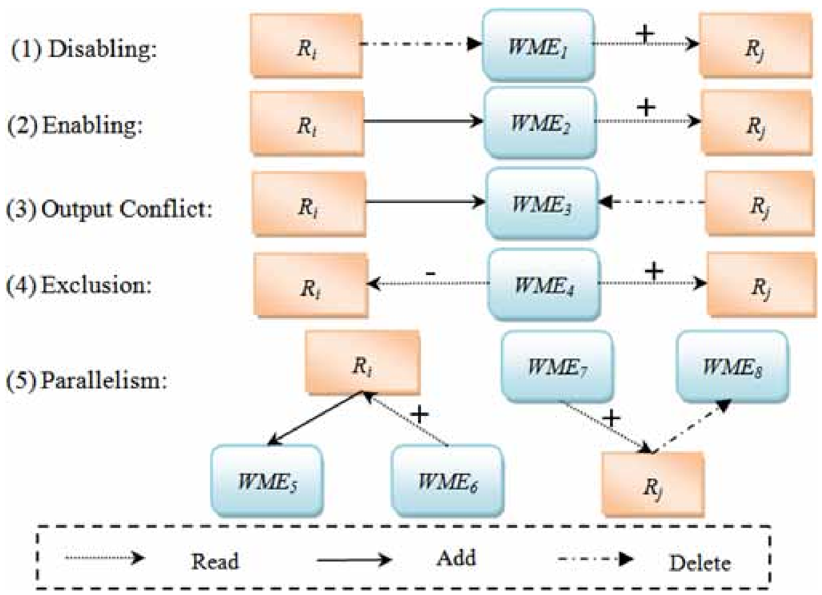
Based on Definition 1 and Definition 2, we can divide the inter-rule dependencies into five categories: disabling, enabling, output conflict, exclusion and parallelism. Accordingly, the five inter-rule relations are represented in Figure 2. The following contents will give the formal definitions of rule relations, on the basis of analyzing each corresponding diagram.

Case (1) in Figure 2 represents the disabling relation between the rules: the action part of rule R_i will delete an existing working memory element WME_1. That will make rule R_j invalid, which has matched WME_1. Apparently, the disabling indicates a relation in which the former suppresses the existing of the latter. The below is the formal definition of the disabling relation.

Definition 3 (Disabling): There is a disabling relation between rule R_i and R_j, if and only if $\exists\ W_k \in WM$ makes any of the following conditions be satisfied:

$A^+(R_i) \rhd W_k$ and $W_k \rhd P^-(R_j)$; 2)$A^-(R_i) \rhd W_k$ and $W_k \rhd P^+(R_j)$ 1)

Figure 2. Inter-rule dependencies diagram



Case (2) in Figure 2 expresses the enabling relation between the rules: the action part of rule R_i decides to add a working memory element WME_2, it will make rule R_j to be matched and instantiated. This kind of relation in which the former activates the latter, is just opposite to the disabling relation. The following is the corresponding definition.

Definition 4 (Enabling): There is an enabling relation between rule R_i and R_j, if and only if $\exists\, W_k \in WM$ makes any of the following conditions be satisfied:

$A^+(R_i) \rhd W_k$ and $W_k \rhd P^+(R_j)$; 2)$A^-(R_i) \rhd W_k$ and $W_k \rhd P^-(R_j)$ 1)

Case (3) in Figure 2 represents the output conflict relation between the rules: rule R_i decides to create a working memory element WME_3, while rule R_j wants to delete the WME_3. The action parts of the two rules will result in the opposite outputs to the WM, which is formally described by definition 5.

Definition 5 (Output Conflict): There is a output conflict relation between rule R_i and R_j, if and only if $\exists\, W_k \in WM$ makes $A^+(R_i) \rhd W_k$and$A^-(R_j) \rhd W_k$.

In the case (4) of Figure 2, the condition part of rule R_i has matched the working memory element WME_4, and rule R_j wants to match the non-existing of WME_4. This kind of opposite referring (matching) to the same WME forms the exclusion relation. Rule R_i and R_j that have exclusion relation make only one rule to be instantiated after the match phase of the production cycle, indicating they exist exclusively. Definition 6 expresses this relation formally.

Definition 6 (Exclusion): There is a exclusion relation between rule R_i and R_j, if and only if $\exists$ $W_k \in WM$ makes $W_k \rhd P^+(R_i)$ and $W_k \rhd P^-(R_j)$.

The working memory elements respectively matched and operated by rule R_i and $R_j$of case (5) in Figure 2 have no intersection, expressing the two rules have no relevance and there is a inherent parallelism between them. The corresponding definition is as follows:

Definition 7 (Parallelism): There is a parallelism relation between rule R_i and R_j, if and only if there is no relation of disabling, enabling, output conflict or exclusion.

We can find the interference relation to be detected and excluded by analyzing the above five inter-rule dependencies. The enabling relation expressed by definition 4 indicates there is a relation between the two rules, in which the former activates the latter. This partial order reveals that these two rules will not fire at the same time. The exclusion relation given by definition 6 shows there are opposite references to the same WME from the two rules. However, there is only one of the two rules to be matched in runtime, which means these two rules will not be instantiated simultaneously. The parallelism relation in definition 7 indicates the two rules having no relevance can execute in any order. It can be seen that the execution result of the two rules, which satisfy Definition 4, 6 or 7, will not lead to the inconsistency problem of WM, so these three relations are not in the considering scope of the objective interference. However, after the two rules with disabling relation execute sequentially, the execution result of the former will make the latter invalid; and the parallel execution of two rules with output conflict relation can cause the inconsistency of WMEs. So the parallel execution result of the two rules with these two relations, which are the objective interferences, will not be equivalent to any sequential execution results, and result in error execution results of multiple rule firing.

Definition 8 (Interference): There is a interference relation between rule R_i and R_j, if and only if there is enabling or output conflict relation between the two rules, that is if and only if $\exists\ W_k \in WM$ makes any of the following conditions to be satisfied:

$$A^+(R_i) \rhd W_k \text{ and} W_k \rhd P^-(R_j) \tag{1}$$

$$A^-(R_i) \rhd W_k \text{ and} W_k \rhd P^+(R_j) \tag{2}$$

$$A^+(R_i) \rhd W_k \text{ and } A^-(R_j) \rhd W_k \tag{3}$$

After clearing the types of interferences shown in Definition 8, the production system can exclude rule instantiations with such characteristics in resolve phase, select compatible rule set, and concurrently execute them in the act phase, in order to achieve the performance promotion of production system. It needs to research multiple rule firing algorithms to realize the above functions.

MULTIPLE RULE FIRING BASED ON ACCESS REQUEST CONTROL

This section will introduce multiple rule firing method used in this paper. Firstly, the workflow of multiple rule firing method is introduced; secondly, the related data structures that correspond to the distributed production system based on MapReduce is presented; thirdly, the request phase and firing phase of this algorithm are described in detail; and finally the correctness of the serialization of the compatible rules is validated.

Workflow of Multiple Rule Firing

Interference analysis can be done in compile time and run time. Compile-time interference analysis is a space-intensive static analysis task, which needs aided data structure such as data dependency graph (Ishida, 1991) or parallel matrix (Schmolze et al., 1992) to describe the rule level dependencies. While analyzing at run time, it spends a large amount of computation time and has to process in rule instantiation level (Ishida., 1991). The pattern element of the production rule is a partial match of the WMEs, so the two rules are determined as interference ones at compile time, and the corresponding rule instantiations may be judged as none-interference because of matching more specific WMEs at runtime. The difference in the analysis results is just because that the more conservative compile-time analysis cannot recognize and obtain all the parallelism in rule instantiation level. Despite a certain amount of computational overhead, the runtime analysis is more specific, and can acquire a higher level parallelism than compile-time analysis. So we adopt runtime analysis in the agenda of the Master to break interference cycle and select compatible rules.

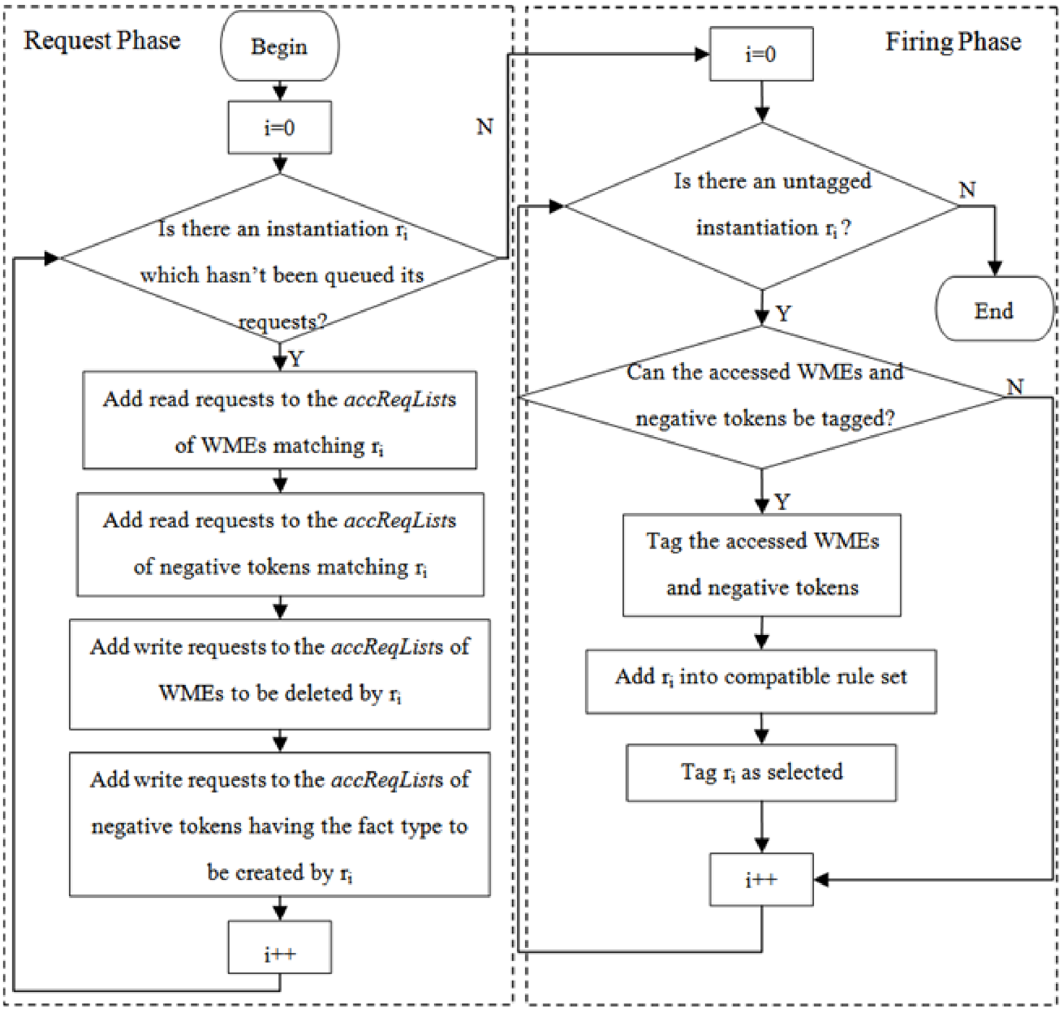
Using multiple rule firing method in resolve phase, it needs to select compatible rules which do not form an interference cycle, and guarantee the consistency of WM when firing and executing them. This situation is similar to the concurrent control in database to keep the consistency of the data. Locking is an important technology to realize the concurrent access control for DB, but it may result in deadlock when multiple transactions lock the required resources directly. Any interference case in Definition 8 is represented as the opposite access requests to the same entity from the two rules. So these access requests can be added to the access request list of the corresponding access entities, which is the request phase of the multiple rule firing method in this paper. And then after the corresponding analysis, the access requests forming an interference cycle with the characteristic of

Definition 8 are excluded, which is the firing phase of the proposed method. These compatible access operations are actually executed at last. It can be seen that this method does not judge the interference characteristics in Definition 8 one by one between the rules, but uses aided data structure to uniform the references and operations to WMEs from the rules as the 'read' and 'write' access requests, and excludes the interference cycle between multiple rule instantiations based on the analysis and control of these requests. This method can realize the compatible access of these rule instantiations to the WM, guarantee the consistency of the WM, and prevent the deadlock from occurring.

The multiple rule firing method introduced in this paper is based on the idea of request access control of Perraju et al. (2000), and only needs to process WMEs and negative tokens which are to be accessed directly without considering the concept of weak pattern (Perraju et al., 2000). Besides, this method uses data structure *NegTokenRef* to store the related information of the negative tokens in the master server which is different from directly accessing the alpha memories of the local Rete network to deal with the access requests of the negative tokens.

The overall workflow of the multiple rule firing method proposed in this paper is shown in Figure 3. The processing loop in the left part of Figure 3 corresponds to the request phase, where

Figure 3. The overall processing workflow of the proposed multiple rule firing method


Request Phase
Begin
i=0
N
Is there an instantiation r_i which hasn't been queued its requests?
Y
Add read requests to the accReqLists of WMEs matching r_i
Add read requests to the accReqLists of negative tokens matching r_i
Add write requests to the accReqLists of WMEs to be deleted by r_i
Add write requests to the accReqLists of negative tokens having the fact type to be created by r_i
i++
Firing Phase
i=0
Is there an untagged instantiation r_i ?
N
End
Y
Can the accessed WMEs and negative tokens be tagged?
N
Y
Tag the accessed WMEs and negative tokens
Add r_i into compatible rule set
Tag r_i as selected
i++

the instantiations in the agenda are treated one by one. The 'read' access requests and 'write' access requests of these instantiations are added to the request lists of the access entities. The loop in the right part of Figure 3 corresponds to the firing phase, in which the access entities of each instantiation will be judged whether can be marked or not, the appropriate entities will be tagged the corresponding access flag, and then the current rule instantiation will be marked as selected and added to the compatible rule set. After the above processing phases, there is no interference cycle between these compatible rules.

Related Data Structures

The multiple rule firing method proposed in this paper is applied in a MapReduce-based distributed production system, and the interference analysis is done in the master server. The master server dispatches sub-rule sets based on pattern elements to the Map Worker clusters. The Rete networks are built in each Map Worker, preparing to be used in the match phase of the production inference cycle. The Rete algorithm accepts the changes of the working memory as the inputs of the Rete network in pattern matching. So these changes can be encapsulated as *Token*, its formal definition is as follow:

Definition 9 (Token): The changes of the working memory include the adding, deleting and modifying of WMEs, and the modifying operation is equivalent to a combination of one deleting and one adding. So these changes can be expressed by Token, denoted as *Token<flag, wme>*, $Token.flag \in \{+,-\}$ indicates the corresponding WME is added to the WM, or is removed from the WM. Token Set can be used to express a set of the WM changes, denoted as:

$$TokenSet = \{Token_i | Token_i.wme \in WM\} \quad (1)$$

After the MapReduce-based distributed matching, a serial of rule instantiations are put into the agenda of the master server. The rule instantiation contains the id and action part, as well as the *TokenSet* which matches this rule. The action part of a rule does adding or deleting operation to the WM, which is the 'write' access request of the rule to the corresponding WME. The *TokenSet* matching with the rule instantiation is the references of the WME changes from the rule, which is the 'read' access request. All the access requests are queued in the *accReqList*of the corresponding access entities.

According to Definition 9, the Tokens can be divided into two categories: Positive Token and Negative Token. The existing of Positive Tokens indicates that in the act phase of the previous inference cycle, the corresponding WMEs are added into the WM, and they can be accessed in the WM. So the access request to the positive token *t* can be directly added into the access request list *accReqListoft.wme*. The existing of Negative Token *t* indicates that in the act phase of the previous inference cycle, there is a corresponding WME *t*.wme having been removed from the WM. It means that *t.wme* does not exist in the WM now, and it needs some data structure to store the related information of Negative Tokens, which is the function of *negTokenRef*. Each item in *negTokenRef* contains a Negative Token and its corresponding *accReqList*. Therefore the access request to a Negative Token can be added to the *accReqList* of some item in the *negTokenRef*.

The way of adding access requests to *accReqList*, does not guarantee that the access request of the corresponding rule instantiation will be allowed, but indicates the possibility of this request to be permitted. It is obvious that the 'read' request from one rule instantiation to the same entity (WME or Negative Token) will conflict with the 'write' request from the other rule instantiation, vice versa.

In addition to the access request list, which holds the access requests of the rule instantiations to the WMEs and Negative Tokens, these access entities also need to display their current access states. So each WME in the WM of the master server contains the fact type and the information of attribute-value pairs, as well as a flag with the value of 0, 1 or 2, which indicate the access state of the corresponding WME is read, write or both. Each item in the *negTokenRef* also has a flag like that.

Processing Phases

Request Phase

The request phase of the proposed multiple rule firing method adds the 'read' and 'write' access requests of the rule instantiations to the *accReqList* of the corresponding accessing entities. After the master server adds a rule instantiation into the agenda, for any matching $token^+ \in TokenSet$, $token^+.wme \in WM$, it needs to add a *readRequest* to the *accReqList*of $token^+.wme$; for any matching $token^- \in TokenSet$, $token^-.wme \notin WM$, it needs to add a *readRequest* to the *accReqList*of this negative token in the *negTokenRef*.

For the negative action elements of the rule instantiations, the master server has to acquire the references of the target WMEs according to the binding information of reserved variables, and add a *writeRequest* to the *accReqList*of the corresponding WME. For the positive action elements of the rule instantiations, it has to judge whether the WME_i to be added can match some negative token in the *negTokenRef* or not. If a successful match occurs, a *writeRequest* has to be added to the *accReqList*of this negative token.

The acquisition process of accessing requests from rule instantiations is shown in Figure 4. A series of rule instantiations generated in current inference cycle constitute a conflict set. They match some WMEs and Negative Tokens, and also need to add or delete some WMEs. These 'read' and 'write' requests are added to the *accReqList*of the corresponding WMEs and Negative Tokens, which is the request phase of the proposed method. The corresponding pseudo-code is shown in Figure 5.

Firing Phase

After completing the request phase, each untagged rule instantiation R_i in the conflict set is to be processed individually. If all the WMEs and Negative Tokens to be accessed by R_i haven't been tagged, R_i will be set as 'selected', and the access entities will be tagged according to the corresponding access

Figure 4. The acquisition process of the accessing requests from the rule instantiations

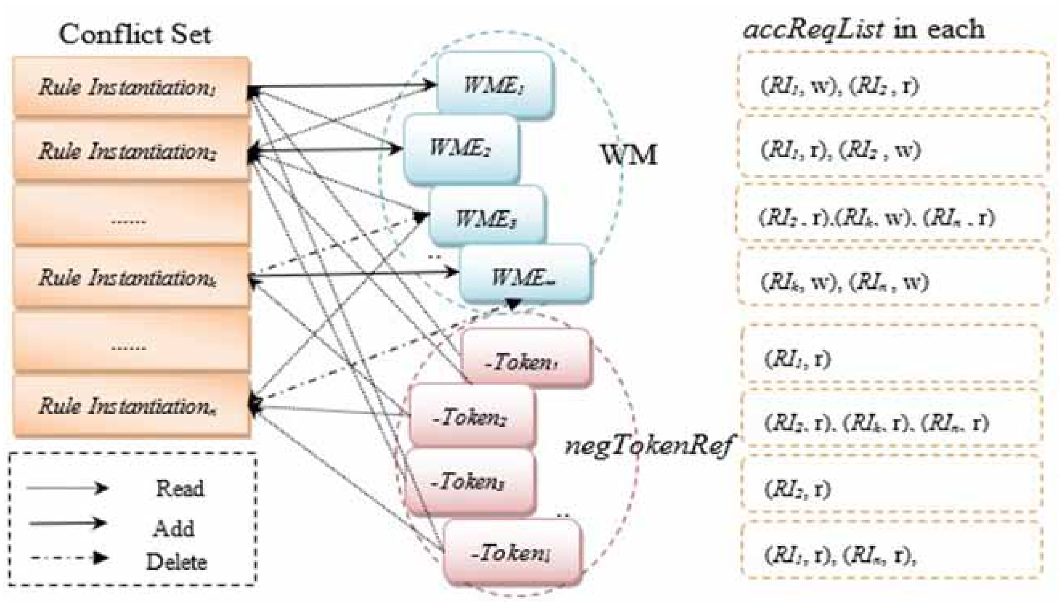
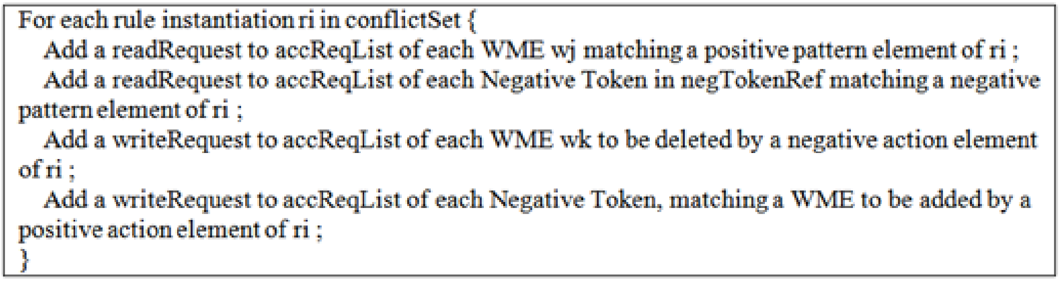


Figure 5. The pseudo-code of the request phase in the proposed method

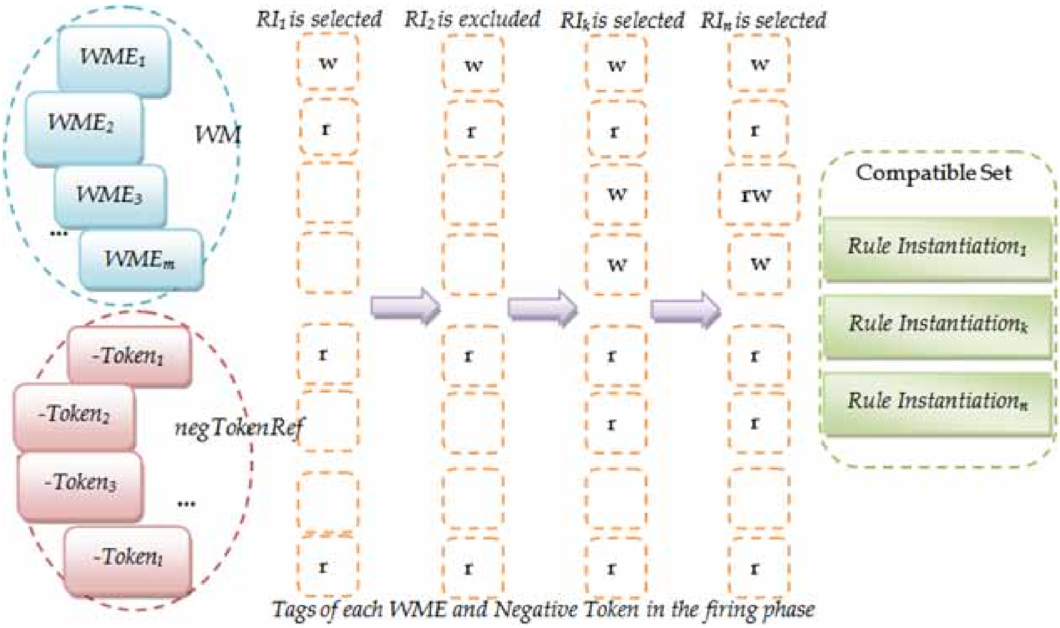


```
For each rule instantiation ri in conflictSet {
    Add a readRequest to accReqList of each WME wj matching a positive pattern element of ri ;
    Add a readRequest to accReqList of each Negative Token in negTokenRef matching a negative
pattern element of ri ;
    Add a writeRequest to accReqList of each WME wk to be deleted by a negative action element
of ri ;
    Add a writeRequest to accReqList of each Negative Token, matching a WME to be added by a
positive action element of ri ;
}
```

requests. Or if the entities accessed by R_i with the ‘read’ (‘write’) access request haven’t been marked as ‘read’ (‘write’), R_i will also be set as ‘selected’. The master server deals with the rule instantiations in the *conflictSet* one by one according to the above way, and makes no interference cycle with the interfering characteristics expressed by Definition 8 between the selected rule instantiations. Finally, the rule instantiations in the conflict set marked as ‘selected’ constitute the required *compatibleSet*.

For instance, the master server has to process the conflict set containing RI_1, RI_2, RI_k and RI_n, and obeys the order of RI_1->RI_2->RI_k->RI_n whose corresponding process is shown in Figure 6. Firstly, the entities to be accessed by RI_1 haven’t been tagged, so RI_1 is set as ‘selected’, and these accessing entities are marked as the corresponding state flag. Secondly, because WME_1 to be ‘read’ accessed by RI_2 and WME_2 to be ‘write’ accessed byRI_2 have been tagged as the opposite accessing state, then RI_2 is marked as ‘excluded’. Thirdly, the entities accessed by RI_k haven’t been tagged, so RI_k is set as ‘selected’ and these entities are marked as the corresponding accessing tag. Finally, WME_3, WME_m, -*token*$_2$ and -*token*$_l$ accessed by RI_n are tagged according to the access requests of RI_n, and RI_n is selected. The selected RI_1, RI_k and RI_n constitute the compatible rule set, in which the rule instantiations can be concurrently fired and executed, always guaranteeing the consistency of the WM.

Figure 6. The firing phase of the compatible rules



The pseudo-code of the firing phase is shown in Figure 7. Combining with the request phase in Figure 5, it constitutes the multiple rule firing method presented in this paper. The input is the rule instantiations set *conflictSet* in the agenda, and the output is the compatible rule instantiation set *compatibleSet*. The instantiations in the *compatibleSet* don't form any interference cycle, which guarantees the parallel execution result of these rule instantiations can be equivalent to a kind of sequential execution result.

After completing the request phase presented in Figure 5 and the firing phase expressed in Figure 7 individually, it enters into the act phase of the production system. In the act phase, the instantiations in *compatibleSet* can be executed concurrently, and the information in *accReqList*s and state flags will be cleared. After that, the change of the WM caused by the compatible rule set will be encapsulated as *TokenSet*, and the system will begin a new inference cycle.

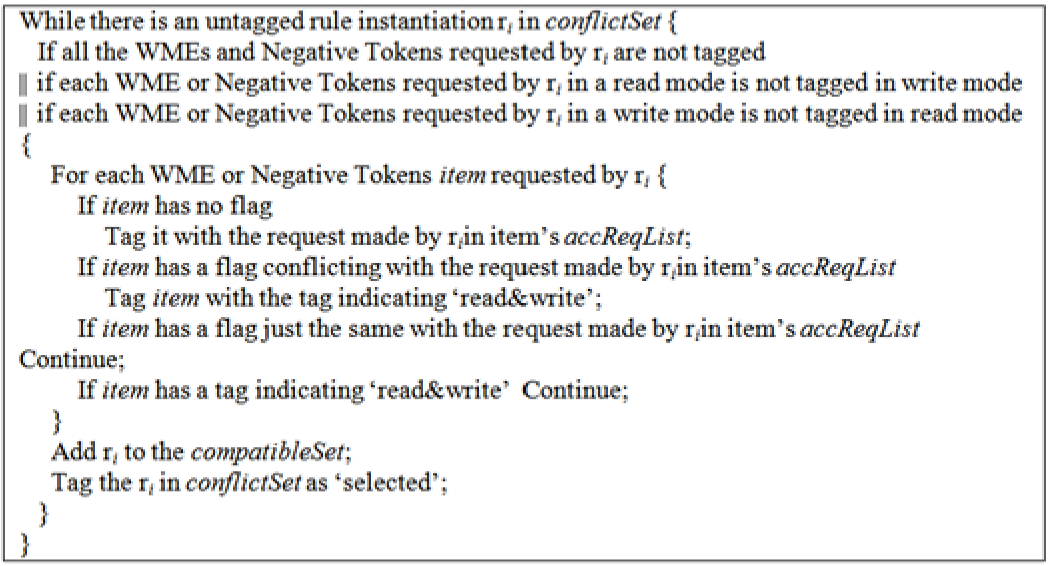
Serialization Validation

C.M et al (1991) have proved that the multiple firing of a rule instantiation set is serializable, as long as there is no interference cycle in this set. So in order to prove the parallel execution result of the compatible instantiation set selected by the method proposed in this paper can be equivalent to a sequential execution result of the same rule set, it has to prove there is no interference cycle between the compatible instantiations selected by this multiple rule firing method.

For example, there is only rule instantiation r_1 in the *compatibleSet*, and no interference cycle exists. It has to consider whether another rule instantiation r_2 in the *conflictSet* can be added into the *compatibleSet* or not. There is an interference cycle between r_1 and r_2, if any of the following conditions satisfies:

Existing $A^+(r_1) \triangleright w_i$ *and* $w_i \triangleright P^-(r_2)$, meanwhile existing $w_j \triangleright P^+(r_1)andA^-(r_2) \triangleright w_j$
Existing $A^+(r_1) \triangleright w_i$ *and* $w_i \triangleright P^-(r_2)$, meanwhile existing $w_j \triangleright P^-(r_1)andA^+(r_2) \triangleright w_j$
Existing $A^-(r_1) \triangleright w_i$ *and* $w_i \triangleright P^+(r_2)$, meanwhile existing $w_j \triangleright P^+(r_1)andA^-(r_2) \triangleright w_j$
Existing $A^-(r_1) \triangleright w_i$ *and* $w_i \triangleright P^+(r_2)$,meanwhile existing $w_j \triangleright P^-(r_1)andA^+(r_2) \triangleright w_j$

Figure 7. The pseudo-code of the firing phase in the proposed method



```
While there is an untagged rule instantiation r_i in conflictSet {
  If all the WMEs and Negative Tokens requested by r_i are not tagged
|| if each WME or Negative Tokens requested by r_i in a read mode is not tagged in write mode
|| if each WME or Negative Tokens requested by r_i in a write mode is not tagged in read mode
{
    For each WME or Negative Tokens item requested by r_i {
       If item has no flag
          Tag it with the request made by r_i in item's accReqList;
       If item has a flag conflicting with the request made by r_i in item's accReqList
          Tag item with the tag indicating 'read&write';
       If item has a flag just the same with the request made by r_i in item's accReqList
Continue;
       If item has a tag indicating 'read&write'  Continue;
    }
    Add r_i to the compatibleSet;
    Tag the r_i in conflictSet as 'selected';
  }
}
```

For case 1) in the request phase, the *accReqList*of a negative token corresponding to w_i is added into (r_1, w) and (r_2, r), and the *accReqList*of w_j is added into (r_1, r) and (r_2, w). r_1 is in the *compatibleSet*, so the access state of the negative token corresponding to w_i has been tagged as 'write', and w_j has been tagged as 'read'. The entities to be accessed by r_2 in the firing phase have been marked with the opposite access state, so r_2 is not selected.

In the request phase, for case 2) the *accReqList* of some negative token corresponding to w_i is added into (r_1, w) and (r_2, r), and the *accReqList* of a negative token corresponding to w_j contains (r_1, r) and (r_2, w); for case 3), the *accReqList*of w_i is added into (r_1, w) and (r_2, r), and the *accReqList* of w_j includes (r_1, r) and (r_2, w); for case 4), the *accReqList* of w_i is added into (r_1, w) and (r_2, r), and the *accReqList* of a negative token corresponding to w_j contains (r_1, r) and (r_2, w). Because r_1 has been added into *compatibleSet*, the entities to be accessed by r_2 in the case 2), 3) and 4) have all been tagged with the opposite access states, r_2 has to be excluded. Therefore when there is only r_1 in the *compatibleSet*, any r_2 which can form the above four possible interference cycles will not been added into the *compatibleSet*.

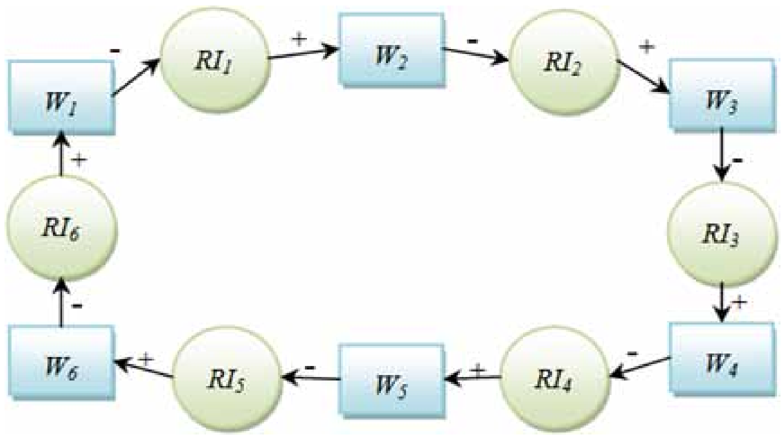
Assuming that there are rule instantiation set $\{r_1, r_2, \ldots, r_k\}$ in the *compatibleSet* after several selections, and there is no interference cycle between them, it has to consider whether r_{k+1} can been added into the *compatibleSet* or not. If the WMEs or negative tokens matched by r_{k+1} have been tagged as 'write' by some rule instantiation in the *compatibleSet*, and the WMEs or negative tokens to be set as 'write' by r_{k+1} have been marked as 'read' by some instantiation in the *compatibleSet*, r_{k+1} will be excluded. r_{k+1} can be added into the *compatibleSet*, which means that it will not form any interference cycle with the instantiations in the *compatibleSet*.

After the validation of the above mathematical induction, there is no interference cycle in the compatible rule set selected by the multiple rule firing method proposed in this paper, which meets the serializability requirement of the rule set proved by Kuo et al.(1991). So the parallel execution result of this compatible rule set will be equivalent to some sequential execution result of the same rule set.

COMPARISON OF PARALLELISM

Generally speaking, we can measure the parallelism degree allowed by interference analysis methods with the number of compatible rule instantiations. The example presented in Figure 8 shows an interference cycle between rule instantiations, which is illustrated by using data dependency graph(Ishida,1991;Kuo et al.,1991). In Figure 8, circles stand for rule instantiations, squares represent WMEs; the directed edge from a circle to a square and the symbol above expresses that an instantiation

Figure 8. An example containing inter-instantiation interference cycle



will add (+) or delete (-) some WME; and the directed edge from a square to a circle and the symbol above shows that an instantiation match a WME (+) or a negative token (-). Based on this example, we can analyze the parallelism degree acquired by several different multiple rule firing methods.

The interference analysis selection algorithm proposed by Ishida (1991) is based on data dependency graph. Initially the *compatibleSet* is empty, this method adds the instantiation RI_i into the *compatibleSet* which satisfies the following conditions: 1) the WME '+' referred by RI_i is not '-' changed (deleted) by the instantiations in the *compatibleSet*; 2)the WME '-' referred by RI_i is not '+' changed (added) by the instantiations in the *compatibleSet*. So based on the example shown in Figure 8, RI_1 is firstly selected and added. W_2 '-' referred byRI_2 will be '+' changed by RI_1, so RI_2 is secondly excluded. Thirdly RI_3 satisfies the two conditions of the algorithm and is selected. Fourthly W_4 '-' referred by RI_4 will be '+' changed by RI_3, so RI_4 is excluded. SimilarlyRI_5 is selected and RI_6 is excluded. Finally the *compatibleSet* is {RI_1, RI_3, RI_5}.

Schmolze et al. (1992) proposed A2 algorithm to asynchronously and individually exclude instantiations forming the interference characteristics of disabling and clashing with other ones in the *conflictSet*, and selected compatible rules. The detection and selection process of this method is similar to Ishida's, but it does not use data dependency graph, and the final acquired *compatibleSet* is also {RI_1, RI_3, RI_5}.

The CREL approach adopted by Kuo et al. (1991) needs to construct a so called RTC inference matrix to aid the selection of compatible rules. The number of lines and columns is equal to the number of rule instantiations in the *conflictSet*. RTC is an adjacent matrix about data dependency graph. If there are a '-' reference and a '+' change about some WME between the two rule instantiations RI_i and RI_j, then RTC[i][j]=RTC[j][i]=1, and the RTC matrix corresponding to the example in Figure 8 can be built.

Then the instantiation nodes are sorted with the ascending order according to the sum of in and out degrees in each line of the RTC. Because the degrees of all the nodes are the same, they are still sorted with the sequence from RI_1 to RI_6. RI_1 is added into the *compatibleSet*, and RI_2 is excluded because of being neighbor node with RI_1. After that, RI_3 is added and RI_4 is excluded. Then this method selects RI_5 and precludesRI_6. The final *compatibleSet* is {RI_1, RI_3, RI_5}.

For the method described in section 3, the access requests from rule instantiations in the *conflictSet* are added to the *accReqList* of the corresponding access entities. So there are *accReqList*(W_1)={ (RI_1, r), (RI_6, w)}, *accReqList*(W_2)={ (RI_1, w), (RI_2, r)}, *accReqList*(W_3)={ (RI_2,w), (RI_3, r)}, *accReqList*(W_4)={ (RI_3,w), (RI_4, r)}, *accReqList*(W_5)={ (RI_4,w), (RI_5, r)}, and *accReqList*(W_6)={ (RI_5,w), (RI_6, r)}. The instantiations in the *conflictSet* are processed individually, and the firing phase is shown as Figure 9. Firstly, RI_1 is added into the *compatibleSet*, WME_1 is set as 'read', and WME_2 is tagged as 'write'. Secondly because the WME_3 to be 'write' accessed by RI_2 hasn't been marked as 'read', RI_2 is added into the *compatibleSet*, WME_2 is tagged as 'read&write', and WME_3 is set as 'write'. Then because of the same reason, RI_3, RI_4 and RI_5 are added, the corresponding access entities are tagged as well. FinallyWME_6 to be 'read' accessed by RI_6 has been tagged as 'write', and WME_1 to be 'write' accessed has been marked as 'read', so RI_6 is excluded. The generating *compatibleSet* is {RI_1, RI_2, RI_3, RI_4, RI_5}. It can be seen from the proof of section 3.4 that the parallel execution result of these instantiations is equivalent to some sequential execution result of the same rule set.

As to the interference cycle shown in Figure 8 involving 6~60 instantiations, it can acquire different maximum sizes of compatible rule sets for these four methods. The maximum size of compatible rule set means the number of rules that can be fired in parallel. The comparison of maximum parallelism for different methods is shown in Figure 10. The processing of the methods proposed by Ishida (1991), Schmolze et al. (1992) and Kuo et al. (1991) are different, but they all need to perform interference detection and exclusion between paired instantiations, and break interference cycle in several places. The method proposed in this paper does not detect interference in pairs, but processes the 'read' and 'write' requests to WMEs uniformly based on the access request list. On the basis of guaranteeing serializability, our method breaks interference cycle in one place, selects and adds more instantiations into the compatible rule set than the other methods, thus acquiring more parallelism.

Figure 9. The firing phase of the proposed method based on the example in Figure 8

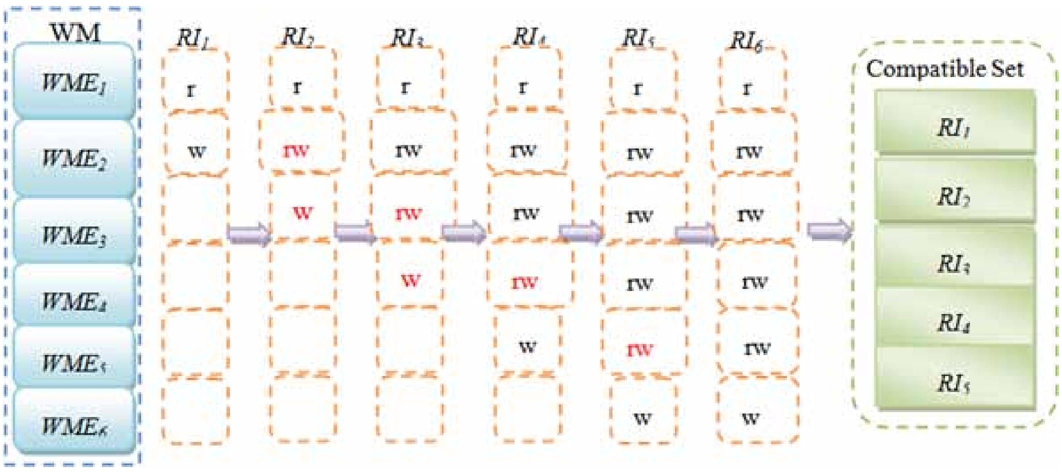
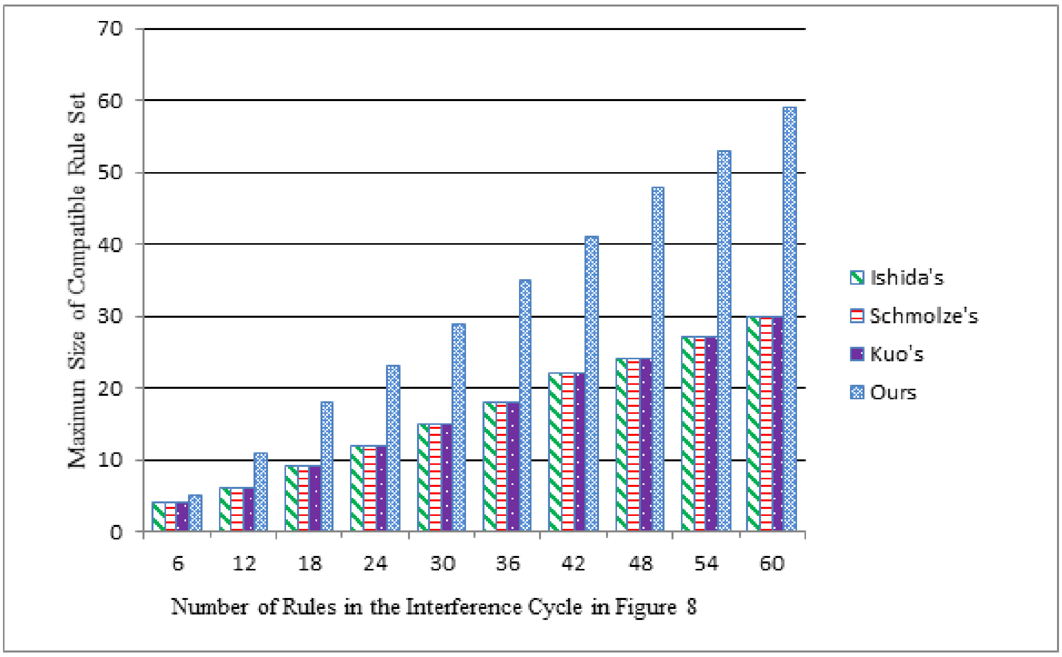


Figure 10. Maximum parallelism of four methods



APPLICATION AND PERFORMANCE EVALUATION

This paper aims at combining the multiple rule firing method based on access control with the MapReduce-based distributed rule matching, and developed a production system with functions of distributed matching and multiple firing. This system is constructed by the master-salve structure with a master server and several workers. The workers can be divided into two categories: Map Worker and Reduce Worker. To prove the effectiveness of the approach to distributed rule matching and multiple firing based on MapReduce, a rule engine for public-travel traffic information service was developed, which adopts the architecture of distributed production system. This rule engine for public-travel traffic information service combines multiple rule firing based on access request control with distributed rule matching and multiple firing based on MapReduce. This public-travel traffic information service system provides all kinds of traffic information for portable navigation devices, smart phones and Web systems. The Master is responsible for monitoring and managing the Workers, as well as preprocessing the business rule sets when the traffic information service system is running. In addition, it assigns matching tasks in traffic information service to the clusters, and uses the multi-trigger algorithm to select a compatible set of rule instance for traffic information service. Map Worker is responsible for the constructing of the local Rete network and executing the specific distributed matching tasks of traffic information service. Reduce Worker is applied to merge the intermediate match results and pass them to the Master, thus completing the process of traffic information service.

This paper uses the cluster consisting of some PCs to construct the rule engine system, which contains Master server, Map Workers and Reduce Workers. The configurations of Master server and Workers are shown in Table 1. In order to guarantee the validation of the data, the statistic of the experimental data is acquired by calculating the average values of several tests. The experiments are divided into two groups. The first group is under the condition of giving 600 initial facts, and evaluates the execution of the MapReduce-based distributed production system, which is implemented based on the idea of Wu et al.(2010). In the first test, the system uses some conflict resolution strategy to select just one rule instantiation to execute, and does the operations similar to those described above. The second group of the test evaluates the execution of the system by using the multiple rule firing method proposed in this paper, under the same conditions.

According to our experimental results, we found that each Map Worker of the distributed production system can compile about 10000 rules. Under the condition of 600 initial fact instances, the number of the sub-rules is increased from 2000 to 20000, the cluster is successively set as 2, 4, 6 and 8 Map Workers with 1 Reduce Worker. We evaluated the execution of the distributed production system without the multiple rule firing method described in Section 4. To be sure, more Map Workers and Reduce Workers can be used in this experiment to promote the system performance. The corresponding experimental results are shown in Figure 11 and Figure 12.

Under the condition of different numbers of Map Workers and different sizes of sub-rules, the system execution time without using the proposed multiple rule firing method is shown in Figure 11, and each curve represents a scale of cluster. It can be seen from Figure 11 that the execution time drops obviously with the increase of Map Workers; this trend becomes more obvious when the number of sub-rules increases continuously. With the increase of sub-rules, the execution time of the cluster with 2 Map Workers increases from 1733ms to 22315ms; while that of 8 Map Workers increases from 814ms to 8317ms. The execution time of the latter increases slowly owing to more

Table 1. The system configuration of the cluster

Configuration	Configuration of Master	Configuration of Worker
CPU	Intel Pentium Duo E7400@2.8GHz	Intel Pentium Duo E7200@2.53GHz
Memory	4GB	2GB
Hard Disk	SATA 250GB	SATA 250GB
LAN	100Mbps	100Mbps
VM	VMware-workstation-7	VMware-workstation-7
OS of VM	Ubuntu 10.10	Ubuntu 10.10
JRE	Jdk_1.6.21	Jdk_1.6.21

Figure 11. System execution time only with the MapReduce-based distributed rule matching

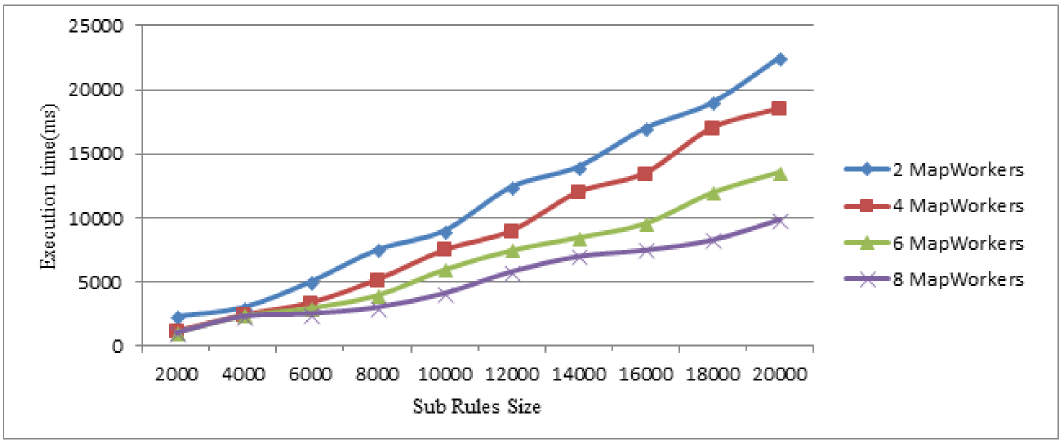
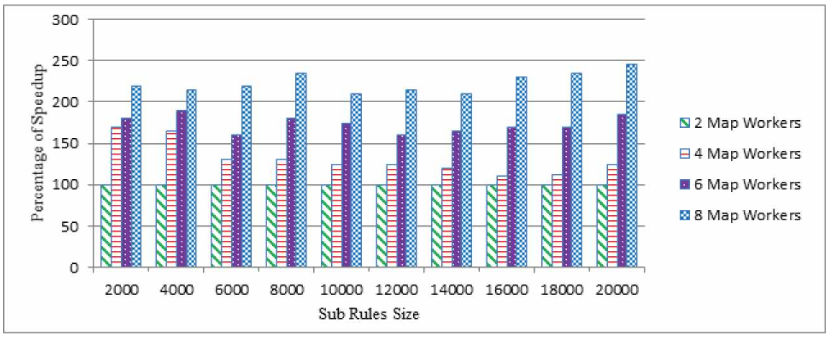


Figure 12. Speedup comparison of distributed rule matching with different cluster scale

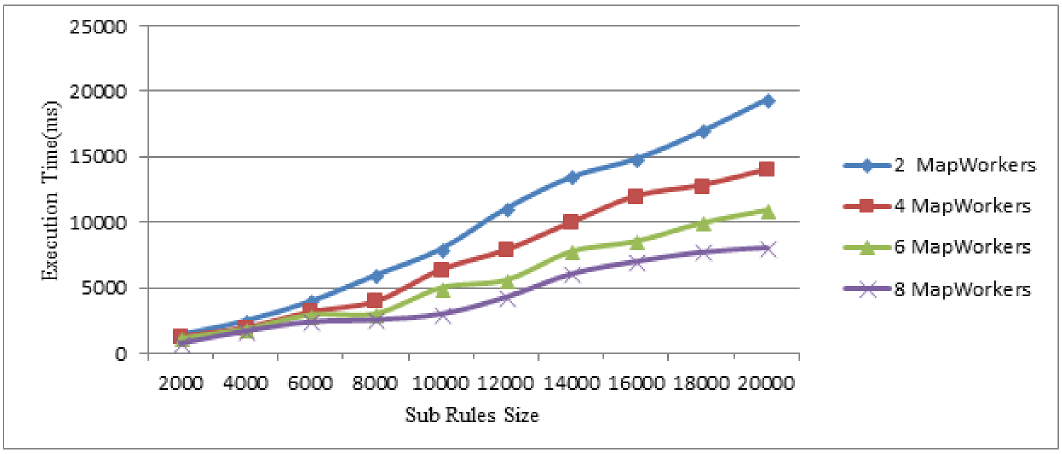


Map Workers. The distributed parallel matching on more Map Workers offsets the execution time brought by the increase of rules.

Based on the data of execution time in Figure 11, under the same scale of the sub-rules, the percentage of speedup presented in Figure 12 is derived from dividing the system execution time of 2 Map Workers by that of other scale of Map Workers. Under the condition of processing the same scale of sub-rules, there is a trend that the more Map Workers there are, the more speedup it can get. However, when the number of sub-rules reaches some scale, the speedup of the system will slow in local. There is more communication cost due to more Map Workers in the cluster, which gradually offsets some speedup brought by more Map Workers. Combining with Figure 11, it can be seen that this drop in speedup is not so important. After all, the overall execution time of the system decreases obviously with the increase of workers. Under the condition of inputting moderate scale of initial facts, the system can meet the processing challenge brought by the increase of rules through enlarging the number of Map Workers.

The second experiment introduces the multiple rule firing method to the MapReduce-based production system. The initial fact size is also 600, and the number of sub-rules expands from 2000 to 20000. The experimental result is shown in Figure 13. With the increase of sub-rules, the execution time of the cluster with 2 Map Workers increases from 1431ms to 19441ms, while that of 8 Map Workers only increases from 753ms to 7933ms.

Figure 13. System execution time with distributed rule matching and multiple rule firing

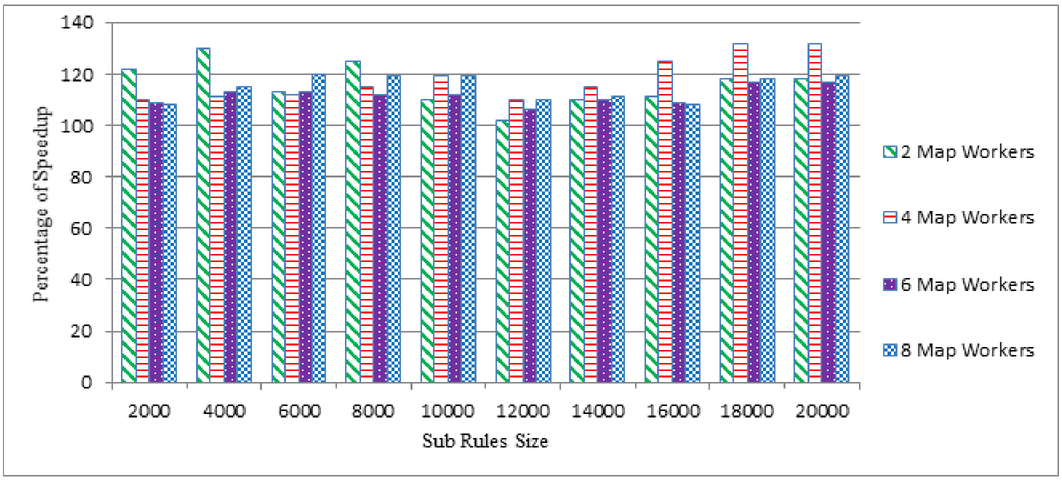


Based on the data of the execution time shown in Figure 11 and Figure 13, under the same scale of the sub-rule (ranging from 2000 to 20000), the percentage of speedup presented in Figure 14 is acquired by dividing the execution time without using the proposed multiple rule firing method by the execution time of using the proposed multiple rule firing method. That means the speedup percentage of the method without multiple rule firing is 100%. Generally speaking, the overall performance of the system is improved about 8% to 33%, and the overall execution time obtains a larger reduction.

CONCLUSION

It is very important for distributed production system to perform massive rules reasoning efficiently, as well as keep the inconsistency problem of working memory. This paper presents a method of distributed rule matching and multiple firing based on MapReduce for efficient scalable rule reasoning. Firstly, the formal definitions are presented according to the analysis of inter-rule relations. Next, a multiple rules firing method based on access request control is proposed, and an example of processing

Figure 14. Speedup comparisons of using and without using multiple rule firing



interference cycle is provided to show the multiple rules firing method of our paper can acquire more parallelism than other methods. Finally, the architecture of distributed production system and the running phases are described. This distributed production system can combine the method of multiple rule firing with the MapReduce-based distributed rule matching. In addition, this paper developed a production system based on MapReduce, and applied the method of distributed rule matching and multiple firing in the master server of the rule engine for public-travel traffic information service. The application shows that the approach proposed in this paper can promote the performance of massive rules reasoning effectively.

ACKNOWLEDGMENT

This work is supported by following foundations: National Natural Science Foundation of China (No.61672464) and Key Research and Development Project of Zhejiang Province (No.2017C01013).

REFERENCES

Bădică, C., Braubach, L., & Paschke, A. (2011). Rule-based distributed and agent systems. *Lecture Notes in Computer Science*, *6826*, 3–28. doi:10.1007/978-3-642-22546-8_3

Batra, D. (2017). Adapting agile practices for data warehousing, business intelligence, and analytics. *Journal of Database Management*, *28*(4), 1–23. doi:10.4018/JDM.2017100101

Cabitza, F., & Seno, B. D. (2005). DJess-A Knowledge-Sharing Middleware to Deploy Distributed Inference Systems. *Proceedings of World Academy of Science*, *Engineering and Technology*, *4*, 66–69.

Cao, B., Yin, J., Zhang, Q., & Ye, Y. (2010). A MapReduce-based Architecture for Rule Matching in Production System. In *IEEE Second International Conference on Cloud Computing Technology and Science* (pp. 790-795). doi:10.1109/CloudCom.2010.11

Cheng, C. J., Cui, F., Le-Fei, F., Xiong, G., & Zou, Y. M. (2010). Parallel Management Systems for Complex Productions Systems: Methods and Cases. *Fuza Xitong Yu Fuzaxing Kexue, 7(1),* 24–32.

Wu, C. C., Lai, L. F., Ke, J. Y., Jhan, S. S., & Chang, Y. S. (2010). Designing a Parallel Fuzzy Expert System Programming Model with Adaptive Load Balancing Capability for Cloud Computing. *Journal of Computers*, *21*(1), 38–48.

Eldawy, A., Mokbel, M., & Jonathan, C. (2016). HadoopViz: A MapReduce Framework For Extensible Visualization of Big Spatial data. In *IEEE International Conference on Data Engineering* (pp. 601-612). doi:10.1109/ICDE.2016.7498274

Gartner. (2002a). Rules: Adding Intelligence to the Enterprise Architecture. Gartner Group.

Gartner. (2002b). The 2002 Business Rule Engine Market Magic Quadrant. M-15-2087.

Dean, J., Ghemawat, S. (2010). MapReduce: A flexible Data Processing Tool. *Communications of the ACM*, *53*(1), 72–77. doi:10.1145/1629175.1629198

Giarratano, J. C., & Riley, G. D. (2005). *Expert System Principles and Programming*. Boston: Boyd & Fraser.

Gupta, A., Forgy, C. L., Kalp, D., Newell, A., & Tambe, M. (1988). Parallel OPS5 on the Encore Multimax. In *Proceedings of the International Conference on Parallel Processing*, Bristol, UK (Vol. 1, pp.271-280).

Hu, D. H., Wang, Y., & Wang, C. L. (2010). BetterLife 2.0: Large-scale Social Intelligence Reasoning on Cloud. In *Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science,* Indianapolis, IN (pp. 529-536). doi:10.1109/CloudCom.2010.108

Ishida, T. (1991). Parallel Firing of Production System Programs. *IEEE Transactions on Knowledge and Data Engineering*, *3*(1), 11–17. doi:10.1109/69.75883

Jiang, D., Ooi, B. C., Shi, L., & Wu, S. (2010). The Performance of MapReduce: An In-depth Study. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *3*(1), 472–483. doi:10.14778/1920841.1920903

Yin, J., Lu, X., Pu, C., Wu, Z., & Chen, H. (2015). JTangCSB: A Cloud Service Bus for Cloud and Enterprise Application Integration. *IEEE Internet Computing*, *19*(1), 35–43. doi:10.1109/MIC.2014.62

Kao, C. (2012). Efficiency Decomposition for Parallel Production Systems. *The Journal of the Operational Research Society*, *63*(1), 64–71. doi:10.1057/jors.2011.16

Kaul, M., Storey, V. C., & Woo, C. (2017). A framework for managing complexity in information systems. *Journal of Database Management*, *28*(1), 31–42. doi:10.4018/JDM.2017010103

Kuo, C. M., Miranker, D. P., & Browne, J. C. (1991). On the Performance of the CREL System. *Journal of Parallel and Distributed Computing*, *13*(4), 424–441. doi:10.1016/0743-7315(91)90101-E

Lee, M. C., Lin, J. C., & Yahyapour, R. (2016). Hybrid Job-Driven Scheduling For Virtual MapReduce Clusters. *IEEE Transactions on Parallel and Distributed Systems*, *27*(6), 1687–1699. doi:10.1109/TPDS.2015.2463817

Li, S., & Hu, S. (2015). Abdelzaher T. The Packing Server for Real-Time Scheduling of Mapreduce Workflows. In *IEEE Real-Time and Embedded Technology and Applications Symposium* (pp.51-62).

Lin, J. C., Leu, F. Y., & Chen, Y. (2015). Impact of Mapreduce Policies On Job Completion Reliability and Job Energy Consumption. *IEEE Transactions on Parallel and Distributed Systems*, *26*(5), 1364–1378. doi:10.1109/TPDS.2014.2374600

Liu, C., Qi, G., Wang, H., & Yu, Y. (2011). Large Scale Fuzzy pD* Reasoning Using MapReduce. In *International Conference on the Semantic Web* (pp. 405-420). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-25073-6_26

Palanisamy, B., Singh, A., & Liu, L. (2015). Cost-Effective Resource Provisioning for MapReduce in a Cloud. *IEEE Transactions on Parallel and Distributed Systems*, *26*(5), 1265–1279. doi:10.1109/TPDS.2014.2320498

Pallickara, S., Ekanayake, J., & Fox, G. (2009). Granules: A Lightweight, Streaming Runtime for Cloud Computing With Support, for Map-Reduce. *IEEE International Conference on CLUSTER Computing and Workshops* (pp. 1-10). IEEE. doi:10.1109/CLUSTR.2009.5289160

Pallickara, S., Ekanayake, J., & Fox, G. Granules. A Lightweight, Streaming Runtime for Cloud Computing With Support for Map-Reduce. In *IEEE International Conference on Cluster Computing and Workshops*, New Orleans, LA (pp.1-10). doi:10.1109/CLUSTR.2009.5289160

Agrawal, R., & Shafer, J. C. (1996). Parallel Mining of Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, *8*(6), 962–969. doi:10.1109/69.553164

Perraju, T. S., & Prasad, B. E. (2000). An Algorithm for Maintaining Working Memory Consistency in Multiple Rule Firing Systems. *Data & Knowledge Engineering*, *32*(2), 181–198. doi:10.1016/S0169-023X(99)00038-5

Petcu, D. (2005a). Parallel Jess. In *The International Symposium on Parallel and Distributed Computing* (pp. 307-316). Lille, France: IEEE Computer Society.

Petcu, D., & Petcu, M. (2005b). Distributed Jess on a Condor pool. In *Proceedings of the 9th WSEAS International Conference on Computers* (pp. 11), Wisconsin, WI: Stevens Point.

Zhang, Q., Zhani, M. F., Yang, Y., Boutaba, R., & Wong, B. (2015). PRISM: Fine-Grained Resource-Aware Scheduling for MapReduce. *IEEE Transactions on Cloud Computing*, *3*(2), 182–194. doi:10.1109/TCC.2014.2379096

Yin, Y., Aihua, S., Min, G., Yueshen, X., & Shuoping, W. (2016). QoS Prediction for Web Service Recommendation with Network Location-Aware Neighbor Selection. *International Journal of Software Engineering and Knowledge Engineering*, *26*(4), 611–632. doi:10.1142/S0218194016400040

Eui-Hong Han., Karypis, G., & Kumar, V. (2000). Scalable Parallel Data Mining for Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, *12*(3), 337–352. doi:10.1109/69.846289

Schmolze, J. G. (1991). Guaranteeing Serializable Results in Synchronous Parallel Production Systems. *Journal of Parallel and Distributed Computing*, *13*(4), 348–365. doi:10.1016/0743-7315(91)90096-R

Schmolze, J. G., & Neiman, D. E. (1992). Comparison of Three Algorithms for Ensuring Serializable Executions in Parallel Production Systems. In *Proceedings of the 10th National Conference on Artificial Intelligence* (pp. 492-492).

Schmolze, J. G., & Neiman, D. E. (1992). Comparison of three algorithms for ensuring serializable executions in parallel production systems. In *Proceedings of the 10th National Conference on Artificial Intelligence,* San Jose, CA (pp. 492-499). DBLP.

Srirama, S. N., Jakovits, P., & Vainikko, E. (2012). Adapting scientific computing problems to clouds using mapreduce. *Future Generation Computer Systems*, *28*(1), 184–192. doi:10.1016/j.future.2011.05.025

Thabtah, F., Cowling, P., & Hammoud, S. (2006). Improving Rule Sorting, Predictive Accuracy and Training Time in Associative Classification. *Expert Systems with Applications*, *31*(2), 414–426. doi:10.1016/j.eswa.2005.09.039

Urbani, J., Kotoulas, S., & Maassen. (2010). WebPIE: AWeb-scale Parallel Inference Engine. In *Third IEEE International Scalable Computing Challenge*, Melbourne: Australia.

Urbani, J., Kotoulas, S., Oren, E., & Harmelen, F. V. (2009). *Scalable Distributed Reasoning Using MapReduce. The Semantic Web - ISWC 2009*. Berlin, Germany: Springer.

Wang, K., & Shen, Z. (2011). Artificial Societies and GPU Based Cloud Computing for Intelligent Transportation Management. *IEEE Intelligent Systems*, *26*(4), 22–28. doi:10.1109/MIS.2011.65

Wu, C. C. (2011). Parallelizing a CLIPS-based Course Timetabling Xxpert System. *Expert Systems with Applications*, *38*(6), 7517–7525. doi:10.1016/j.eswa.2010.12.116

Wu, C. C., Lai, L. F., & Chang, Y. S. (2008). Using MPI to Execute a FuzzyCLIPS Application in Parallel in Heterogeneous Computing Systems. In *Proceedings of the IEEE 8th International Conference on Computer and Information Technology*, Sydney, Australia (pp. 279-284).

Xiao, J., & Xiao, Z. (2011). High-Integrity MapReduce Computation in Cloud with Speculative Execution. In Theoretical and Mathematical Foundations of Computer Science (pp. 397-404).

Xu, B., de Fréin, R., Robson, E., & Foghlú, M. Ó. (2012, May). Distributed formal concept analysis algorithms based on an iterative MapReduce framework. In *International Conference on Formal Concept Analysis* (pp. 292-308). Springer.

Yang, H. C., Dasdan, A., Hsiao, R. L., & Parker, D. S. (2007). Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* Beijing, China (pp.1029-1040). doi:10.1145/1247480.1247602

Tianyang Dong was born in Dongyang City, Zhejiang Province, China in 1977 and received his M.S. in Mechanical Engineering and Ph.D. in Computer Science from Zhejiang University, Hangzhou, China in 2002 and 2005, respectively. He also went to the University of Queensland, Australia, as a visiting scholar from March to September of 2011. He works for Zhejiang University of Technology since 2005. His research interest includes computer graphics and distributed computing.

Qiang Cheng was born in Quzhou City, Zhejiang Province, China in 1990 and received his B.S. in 2014 and is currently completing his M.S. coursework in College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China. His research interests are in service computing and software component technology.

Bin Cao received his Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2013. He was sponsored by the China Scholarship Council to pursue his research as a joint Ph.D. student with the Department of Computer Science, University of Minnesota, Minneapolis, MN, USA, in 2012. He then worked as a research associate in Hongkong University of Science and Technology and Noah's Ark Lab, Huawei for half a year. He joined Zhejiang University of Technology, Hangzhou, China in 2014, and is now an Associate Professor in College of Computer Science and Technology. His research interests include service computing and data management.

Jianwei Shi was born in Jiaxing City, Zhejiang Province, China in 1986 and received her B.S. in 2009 and is currently completing his M.S. coursework in College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China. His research interests are in distributed computing and software component technology.