Distributed Knowledge Map for Mining Data on Grid Platforms

Nhien An Le Khac, Lamine M. Aouad and M-Tahar Kechadi,

an.lekhac@ucd.ie lamine.aouad@ucd.ie tahar.kechadi@ucd.ie
School of Computer Science and Informatics
University College Dublin, Belfield, Dublin 4, IRELAND

Summary
Nowadays, massive amounts of data which are often geographically distributed and owned by different organizations are being mined. As consequence, a large mount of knowledge is being produced. This causes the problem of efficient knowledge management in distributed data mining (*DDM*). The main aim of *DDM* is to exploit fully the benefit of distributed data analysis while minimizing the communication overhead. Existing *DDM* techniques perform partial analysis of local data at individual sites and then generate global models by aggregating the local results. These two steps are not independent since naive approaches to local analysis may produce incorrect and ambiguous global data models.

To overcome this problem, we introduce "knowledge map" approach to represent easily and efficiently the knowledge mined in a large scale distributed platform such as Grid. This approach is developed and integrated in a DDM framework. This will also facilitate the integration/coordination of local mining processes and existing knowledge to increase the accuracy of the final model. Our "knowledge map" is being tested on real large datasets..

Key words:
distributed data mining, knowledge map, knowledge management.

1. Introduction

While massive amounts of data are being collected and stored from not only science fields but also industry and commerce fields, the efficient mining and management of useful information of this data is becoming a scientific challenge and a massive economic need. This led to the development of distributed data mining (*DDM*) techniques [16][17] to deal with huge and multi-dimensional datasets distributed over a large number of sites. This phenomenon leads to the problem of managing the mined results, so called knowledge, which becomes more complex and sophisticated. This is even more critical when the local knowledge of different sites are owned by different organizations. Existing (*DDM*) techniques is based on performing partial analysis on local data at individual sites and then generating global models by aggregating these local results. These two steps are not independent since naive approaches to local analysis may produce incorrect and ambiguous global data models. In order to take the advantage of mined knowledge at different locations, *DDM* should have a view of the knowledge that not only facilitates their integration but also minimizes the effect of the local results on the global models. Briefly, an efficient management of distributed knowledge is one of the key factors affecting the outputs of these techniques.

Recently, many research projects on knowledge management in data mining were initiated [28][13][1]. Their goals are to tackle the knowledge management issues as well as present new approaches. However, most of them propose solutions for centralized data mining and only few of them have attempted the issues of large scale *DDM*. Moreover, some recent research works [4] have just provided a manner of managing knowledge but not the integration and coordination of these results from local results.

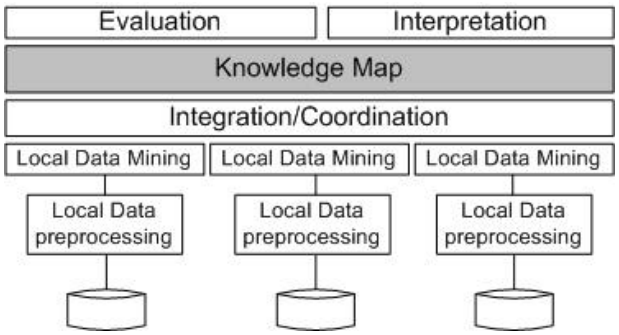


Fig. 1 ADMIRER's core architecture.

In this paper, we propose a "knowledge map", an approach for managing knowledge of (DDM) tasks on large scale distributed systems and also supporting the integration views of related knowledge. The concept of knowledge map has been efficiently exploited in managing and sharing knowledge [23] in different domains but not yet in *DDM* techniques. Our main goal is to provide a simple and efficient way to handle a large amount of knowledge built from *DDM* applications in Grid environments. This knowledge map helps to retrieve

Manuscript received October 10, 2007
Manuscript revised

quickly any results needed with a high accuracy. It will also facilitate the merging and coordination of local results to generate global models. This knowledge map is one of the key layers of ADMIRE [18] (Figure 1), a framework based on Grid platform for developing *DDM* techniques to deal with very large and distributed heterogeneous datasets.

The rest of this paper is organized as follows: In section 2, we give some backgrounds of knowledge representation and knowledge map concept as well as related projects. We present the architecture of our knowledge map in section 3. Section 4 presents knowledge map's operations. Implementation issues of knowledge map are presented in section 5 and an evaluation of this approach is presented in section 6. Finally, we conclude in section 7.

2. Background

In this section, we present some methods for representing knowledge in data mining. We discuss the concept of knowledge map and its use in managing the knowledge. This section will be ended by related work on knowledge map and knowledge management.

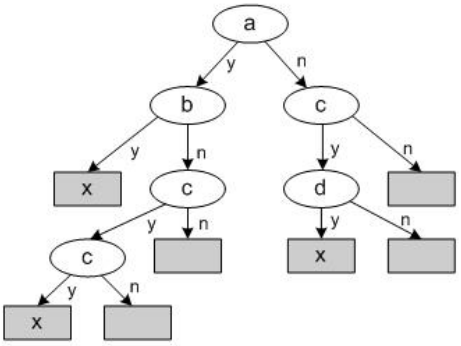
2.1 Knowledge representation

There are many different ways of representing mined knowledge in literature, such as decision tables, decision trees (Figure 2), classification rules, association rules, instance-based and clusters. *Decision table* is one of the simplest ways of representing knowledge. The columns contain set of attributes including the decisions and the rows represent the knowledge elements. This structure is simple but it can be sparse because of some unused attributes. *Decision tree* approach is based on "divide-and-conquer" concept where each node tests a particular attribute and the classification is given at the leaves level. However, it has to deal with missing value problem. *Classification rules* [9] are a popular alternative to decision tree. *Association rules* [9] are kind of classification rules except that they can predict any attribute and this gives them the flexibility to predict combinations of attributes too. Moreover, association rules are not intended to be used together as a set as classification rules are.

Classification rules as well as association rules are a kind of production rules [2] that are widely used in knowledge representation [12]. A rule is a knowledge representation technique and a structure that relates one or more causes, or a situation, to one or more effects (consequents) or actions. It is also called cause-effect relationships represented by an "IF {cause expression} THEN {conclusion expression}". The IF part of the rule is an cause expression composed of causes, and the effects are contained in the conclusion expression of THEN, so that the conclusions may be inferred from the causes when they are true. A rule may also be extended to an *uncertain rule* or a *fuzzy rule* by adding appropriate attributes. Briefly, the knowledge of an intelligent system could be represented by using a number of rules. In this case, these rules are usually grouped into sets and each set contains rules related to the same topic. In the data mining, rules can be used in the representation of knowledge learnt from classification tasks, association rules tasks, etc. It is also called rule-based classification [14] in classification problems where a set of "IF, THEN" rules including attributes such as *coverage* and *accuracy* is applied. Moreover, rules can be extracted from other kinds of model representations such as decision tree, neural network, etc. In association rule tasks, knowledge is represented by a set of rules with two attributes: confidence and support.

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
sunny	cool	normal	true	yes
...				

(a) Decision table



(b) Decision tree

Fig. 2 Knowledge representations

The instance-based knowledge representation uses the instances to represent what is mined rather than inferring a rule set and store it instead. The problem is that they do not make explicit the structures of the knowledge. In the cluster approach, the knowledge can take the form of a diagram to show how the instances fall into clusters. There are many kinds of cluster representations such as space partitioning, Venn diagram, table, tree, etc. Clustering [9] is often followed by a stage in which a decision tree or rule set is inferred allocating each instance to its cluster. Other knowledge representation approaches, such as Petri net

[25], Fuzzy Petri nets [5] and G-net [8] were also developed and used.

2.2 Knowledge map concept

A knowledge map is generally a representation of "knowledge about knowledge" rather than of knowledge itself [7][10][29]. It basically helps to detect the sources of knowledge and their structures by representing the elements and structural links of the application domains. Some kind of knowledge map structure that can be found in literature are: hierarchical/radial knowledge map, networked knowledge map, knowledge source map and knowledge flow map.

Hierarchical knowledge map, so-called concept map [23], provides one model for the hierarchical organization of knowledge: top-level concepts are abstractions with few characteristics. Concepts of the level below have detailed traits of the super concept. The link between concepts can represent any type of relations as "is part of", "influences", "can determine", etc. A similar approach is radial knowledge map or mind map [3], which consists of concepts that are linked through propositions. However, it is radially organized. Networked knowledge map is also called causal map which is defined as a technique "for linking strategic thinking and acting, making sense of complex problems, and communicating with others what might be done about them" [3]. This approach is normally used for systematizing knowledge about causes and effects. Knowledge source map [10] is a kind of organizational charts that does not describe functions, responsibility and hierarchy, but expertise. It helps experts in a specific knowledge domain. The knowledge flow map [10] represents the order in which knowledge resources should be used rather than a map of knowledge.

2.3 Related works

Little research work on knowledge map is given in [11][21]. However, these few projects were not in the context of *DDM*.

The Knowledge Grid project [4] proposed an approach to manage the knowledge by using Knowledge Discovery Service. This module is responsible for handling meta-data of not only knowledge obtained from mining tasks but also all kinds of resources such as hosts, data repositories, used tools and algorithms, etc. All metadata information is stored in a Knowledge Metadata Repository. However, this approach does not provide a management of meta-data of knowledge in their relationships to support the integration view of knowledge as well as the coordination of local the mining process. There is moreover no distinct separation between resource, data, and knowledge.

Until now, to the best of our knowledge, in spite of the popularity of *DDM* applications, there is only our system [19] that provides knowledge map layer for *DDM* applications on a Grid type platforms. This constitutes one of the motivations of our research to provide a fully integrated view of knowledge to facilitate the coordination of local mining processes and increase the accuracy of the final models.

3. Architecture of knowledge map

The knowledge map (*KM*) does not attempt to systematize the knowledge itself but rather to codify "knowledge about knowledge". In our context, it facilitates (*DDM*) by supporting users coordination and interpretation of the results. The objectives of our (*KM*) architecture are: provide an efficient way to handle a large amount of data collected and stored in large scale distributed system; retrieve easily, quickly, and accurately the knowledge; and support the integration process of the knowledge. We propose an architecture of the (*KM)* system as shown in Figure 3, 4, 5 and 6 to achieve these goals. *KM* consists of the following components: knowledge navigator, knowledge map core, knowledge retrieval, local knowledge map and knowledge map manager (Figure 3). From now on, we use the term "mined knowledge" to represent for knowledge built from applications.

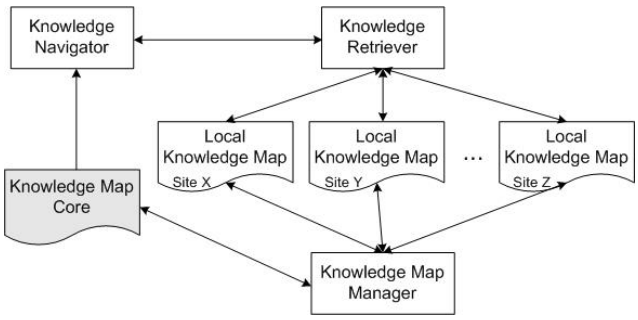


Fig. 3 Knowledge map systems

3.1 Knowledge navigator

Usually, users may not exactly know the mined knowledge they are looking for. Thus, knowledge navigator component is responsible for guiding users to explore the *KM* and for determining the knowledge of interest. The result of this task is not the knowledge but its meta-data, called ***meta-knowledge***, which includes related

information such as data mining task used, data type, and a brief description of this knowledge and its location. For example, a user may want to retrieve some knowledge about tropical cyclone. The application domain "meteorology" allows the user to navigate through tropical cyclone area and then a list of knowledge related to it will be extracted. Next, based on this meta-knowledge and its application domain, the users will decide which knowledge and its location are to be retrieved. It will interact with knowledge retrieval component to collect all mined knowledge from chosen locations.

3.2 Knowledge map core

This component (Figure 4) is composed of two main parts: *concept tree repository* and *meta-knowledge repository*. The former is a repository storing a set of application domains. Each application domain is represented by a *concept tree* that has a hierarchical structure such as a concept map [23]. A node of this tree, so called *concept node* represents a sub-application domain and each *concept node* includes a unique identity, called *concept identity*, in the whole *concept tree* repository and a name of its sub-application domain. The content of each *concept tree* is defined by the administrator before using the *KM* system. The concept tree repository could also be updated during the runtime. In our approach, a mined knowledge is assigned to only one sub-application domain and this assignment is given by the users.

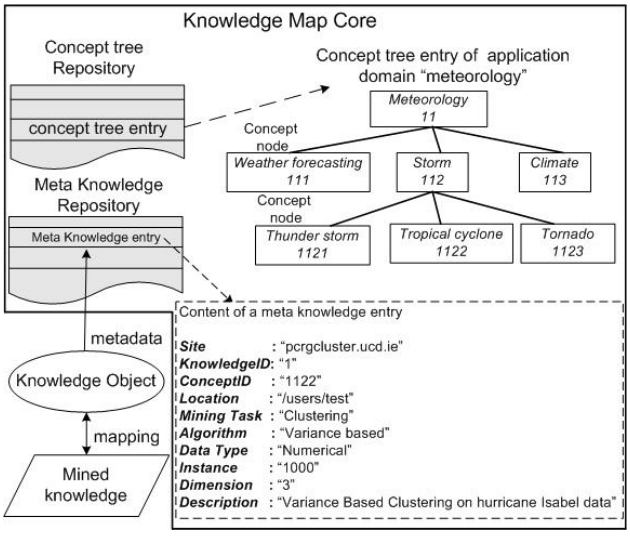


Fig. 4 Knowledge map core structure

As shown in Figure 4 for example, the ***concept tree repository*** contains an application domain named "meteorology" which includes sub-application domains such as "weather forecasting", "storm" and "climate". And then, "thunder storm", "tropical cyclone" and "tornado" are parts of "storm". By using *concept tree*, we can deal with the problem of knowledge context. For instance, given the distributed nature of the knowledge, some of them may have variations depending on the context in which it is presented locally. Moreover, we can also extend the concept tree of each application domain to an ontology of this domain in order to increase the accuracy in retrieving knowledge in different contexts. At that moment, the concept tree will become a taxonomy tree and a list of term as well as slots [12] will be added. The ontology-based architecture of this repository will be applied in the next version of our *KM*.

Meta-Knowledge repository (Figure 4): this handles meta-data of the mined knowledge from different sites. A knowledge is mapped to a ***knowledge object*** and its meta-data is represented by a meta-knowledge entry in this repository. Figure 4 also shows an example of a meta-knowledge entry in XML format. In this example, this knowledge is built from *"pcrgcluster.ucd.ie"* (*knowledge location*) and its local identity (*knowledge identity KID*) is 1; its concept identity (*CID*) is 1122 (sub-application domain is *tropical cyclone*); the location of datasets is "/users/test/"; the used mining task is "clustering" and its algorithm is "variance-based" [20]. Other related information are data type of mined datasets, number of instances, dimension of data and a brief description about this knowledge. Based on this information, users could determine which mined knowledge they want to extract.

The goal of *KM* core, is not only to detect the sources of knowledge and information but also represent their relationships with concepts of application domains. The location of this component depends on the topology of the system. It could be, for example, implemented in a master site assigned to a group of sites. The creation and maintenance of this component as well as its operations such as retrieving knowledge will be presented in section 4.

3.3 Knowledge retrieval

The role of this component is to seek the knowledge that is potentially relevant. This task depends on the information provided by the users after navigating through application domains and getting the meta-knowledge needed. This component is similar to a search engine which interacts with each site and returns knowledge acquired.

3.4 Local knowledge map

This component (Figure 5) is located in each site of the system where knowledge are built from datasets. *Local knowledge map* is a repository of knowledge entries. Each

entry, which is a knowledge object, represents a mined knowledge and contains two parts: *meta-knowledge* and a *representative*. *Meta-knowledge* includes information such as the identity of its mined knowledge that is unique in this site, its properties, and its description. Theses attributes are already explained in the section ***Knowledge map core*** above. This *meta-knowledge* is also submitted to the ***Knowledge map core*** and will be used in *meta-knowledge entry* of its repository at the global level. The *representative* of a knowledge entry depends on a given mining task. *KM* supports two kinds of representatives: one for knowledge mined from clustering tasks and another for mined knowledge represented by production rule. Our system has however the capacity of adding more representative types for other mining tasks.

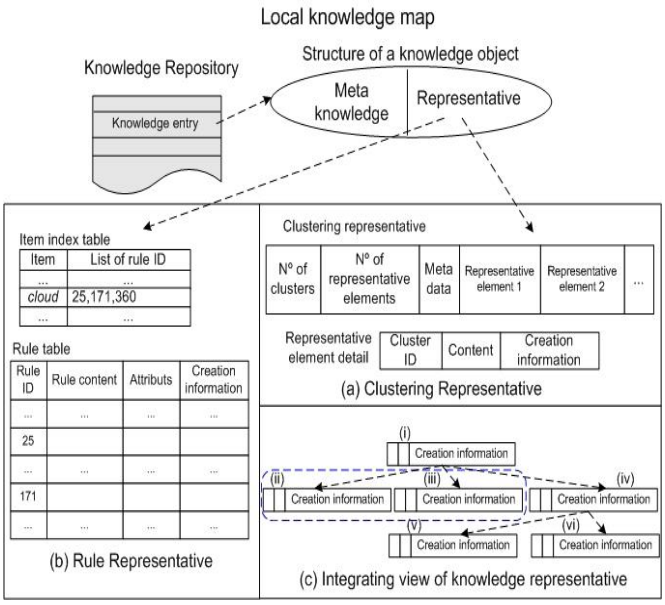


Fig. 5 Organization of local knowledge map

In the clustering case (Figure 5a), a representative of a mined knowledge stands in one or many clusters. A cluster has one or more representative elements and each element consists of fields filled by the user. The number of fields as well as data type of each field, which is also defined by the user, depends on the clustering algorithm used. The meta-data of these fields is also included in each representative. *KM* allows the user to define this meta-data with both scalar and vector data type. A cluster also contains information about its creation. This information shows how this cluster was created: by clustering or integration process. In the former case, the information is a tuple of (*hostname, cluster filename, cluster identity*) and in the latter, it is a tuple of (*hostname, knowledge identity, cluster identity*). *hostname* is the location where clustering results are stored in files called cluster files with their *cluster filenames*. Each cluster has a *cluster identity* and it is unique in its knowledge entry. For example, a knowledge entry which is created by a variance-based clustering algorithm [20] on test datasets, has its representative in XML format as shown in Figure 7. In this example, there are three clusters, each one has only one representative. A cluster representative consists of three fields: *cluster identity*, *counts*, *centres* and *variances* with their data types which are *integer*, *long*, *vector 3 of doubles* and *matrix 3x3 of doubles* respectively. The content of a cluster representative is presented after its meta-data. Besides, another important information of cluster representative is the creation type which shows how this cluster was created: by either a clustering process or an integration process which merges sub-clusters from different sources to build this cluster. In the integration case, the cluster representative shows its integration link representing all information needed to build this cluster. Figure 5c and Figure 6b show an example of integration link. In this figure, the cluster at the root level is integrated from three other sub-clusters where the last one is also integrated from two others. Note that in Figure 5c, representatives (ii) and (iii) belong to the same knowledge.

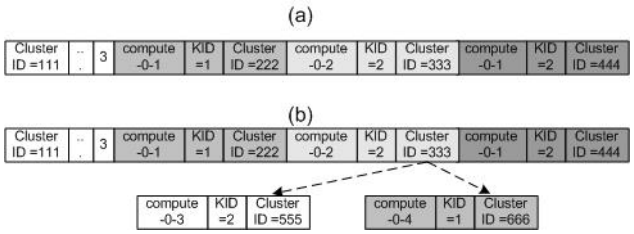


Fig. 6 An example of integration link of creation information: (a) one level; (b) multi-level

In the rule case (Figure 5b), the mined knowledge is represented as a set of the production rules [2]. As mentioned above, a rule is of the form "IF cause expression THEN conclusion expression" and an expression (cause or conclusion) contains a set of items. A rule also includes its attributes such as *support* and *confidence* [9] in association rules task or *coverage* and *accuracy* [14] in classification task, etc. In order to represent these rules by their items, a representative in our approach consists of two parts: a *rule table* and an *item index table*. The former is a table of rules where each line represents a rule including its identity, content, attributes and creation information. The *item index table* is a data structure that maps items to the rule table. For example, the index of a book maps a set of selected terms to page numbers. There are many different types of index described in literature. In our approach, the index table is based on *inverted list* [30] technique because it is one of the most efficient index structures [31]. This index table

consists of two parts: items and a collection of lists, one list per item, recording the identity of the rule containing that item. For example (Figure 5a), we assume that the term *"cloud"* exists in rules of which identities are *25, 171, 360*, so its list is *{25, 171, 360}*. This index table also expresses the relationship between items and their corresponding rule. By using this table, rules which are related to the given items will be retrieved by the intersection of their lists, e.g. the list of term *"pressure"* is *20, 171* so the identity (ID) of rule that contains *"cloud"* and *"pressure"* is *171*. This ID is then used to retrieve the rule and its attributes. In addition, a rule can be created by using one or more other rules, so its creation information keeps this link (Figure 5c).

```
<representative type="Clustering">
  <numcluster>3</numcluster>
  <numrep>1</numrep>
  <repmetadatafields>Id,Counts,Centers,Variances </repmetadatafields>
  <repmetadatatype>int;long;vector,3,double;matrix,3,double</repmetadatatype>
  <cluster>
    <clusterID>0</clusterID>
    ......
  </cluster>
  <cluster>
    <clusterID>1</clusterID>
      <rep>1;1500;0.005,0.006,0.007;0.0051,0.0061,0.0071,0.0052,0.0062,0.0072,0.0053,0.0063,0.0073</rep>
      <creating type="Integrating">
        <numelement>3</numelement>
        <level0>
          ......
        </level0>
        <level0>
          ......
        </level0>
        <level0>
          <site>compute-0-1</site>
            <knowledgeID>1</knowledgeID>
          <clusterID>2</clusterID>
          <numelement>2</numelement>
          <level1>
            <site>compute-0-0</site>
            <knowledgeID>1</knowledgeID>
            <clusterID>1</clusterID>
            <numelement>0</numelement>
          </level1>
          <level1>
              <site>compute-0-2</site>
              <knowledgeID>0</knowledgeID>
              <clusterID>0</clusterID>
              <numelement>0</numelement>
          </level1>
        </level0>
      </creating>
  </cluster>
  <cluster>
    <clusterID>2</clusterID>
    ...
  </cluster>
</representative>
```

Fig. 7 A representative of Clustering in XML format

3.5 Knowledge map manager

Knowledge map manager is responsible for managing and coordinating the local knowledge map and the knowledge map core. For *local knowledge map*, this component provides primitives to create, add, delete, update knowledge entries and their related components (e.g. *rule ne*} and *item index table*) in knowledge repository. It also allows to submit local meta-knowledge to its repository in *knowledge map core*. This component provides also primitives to handle the meta-knowledge in the repository as well as the concept node in the concept tree repository. A key role of this component is to keep the coherence between the *local knowledge map* and the *knowledge map core*.

4. Knowledge Map operations

4.1 Adding new knowledge

For any new mined knowledge, its corresponding meta-data and its representative are generated and mapped to a *knowledge object*. This object will be added to the *local knowledge repository* with an appropriate *concept identity*. Its meta-knowledge is then submitted to the *meta-knowledge repository* of *knowledge map core*. The adding operation is realized via the primitive ***"put"***. The Figure 8 shows a flowchart of the adding process.

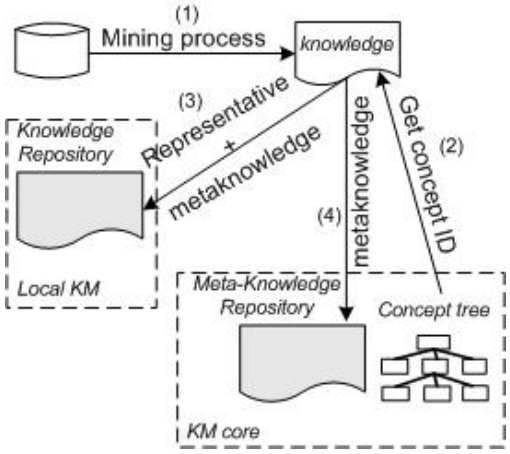


Fig. 8 Adding a new knowledge: (1) knowledge built by a mining process; (2) get an appropriate Concept Identity; (3) knowledge object is added to *local knowledge repository*; (4) *Meta-knowledge* is submitted to *meta-knowledge repository*

4.2 Update/Delete knowledge

KM allows users to update or to delete an existing knowledge meta-data via ***"update"*** and ***"delete"*** primitives. These operations are executed at local site and then the system will automatically update *knowledge map core* to ensure the coherence between core and *local knowledge map*. This operation is moreover atomic.

4.2 Knowledge searching/retrieving

These operations are functions of ***find/retrieve*** primitives (Figure 9). *KM* supports different levels of search: concepts or meta-data of mined knowledge. At the concept level, *KM* allows the user to search and retrieve concepts acquired through their identity or name. The search operation can be done using different criteria such as concept (e.g. search all *meta knowledge* of a selected concept), mining task and algorithm used to build its knowledge. The retrieve operation is performed through the *knowledge identity* and the *location* of the knowledge needed. This process returns a knowledge object. This operation is executed both locally and globally, i.e. users can retrieve the knowledge needed at its local site or from a group of sites of the system.

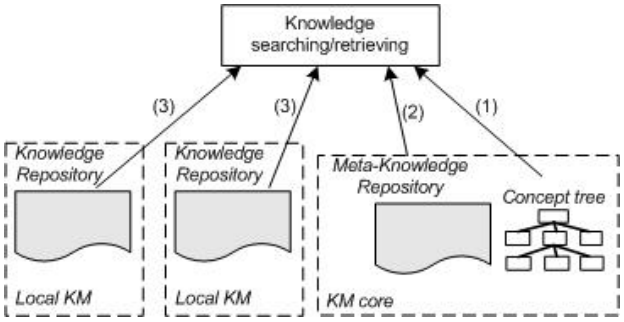


Fig. 9 Retrieving knowledge: (1) Concept (ID or name) retrieving; (2) Meta knowledge retrieving; (3) Knowledge Retrieving from different local KMs

5. Implementation and Exploitation

We have implemented a prototype of *KM* and in the current version, the topology of distribution is a flat tree where one local site is elected as the host. The *meta-knowledge repository* of *KM core* is located at this host while its *concept tree* resides in every site. In this case, only the administrator can define and update the content of this *concept tree* at one site and KM system will then update every replicas. The advantages and disadvantages of this approach will be discussed in the section 6.

In order to exploit mined knowledge, these knowledge should be managed by *KM* system. If it is not, then the first step is to create knowledge objects including meta-knowledge and representatives, and then add it in each appropriate *local KM*. In the current implementation, a knowledge object has XML format as shown in Figure 7. Their *meta-knowledge* will be automatically submitted to the *meta-knowledge repository* at the *knowledge core map* as an adding operation of a new mined knowledge. Next, users can exploit these meta-knowledge and knowledge object in their integration process or explore the knowledge. In this version, repositories of *KM core* and *Local KM* are also in XML format.

Communication Our aim is to provide an efficient *KM* for distributed environments. Our approach provides a flexible solution so that *KM* can be carried on or interact with different communication system (e.g. RMI [15]) as well as workload management systems on cluster or grid platforms (e.g. Condor [6], PBS or OpenPBS [26]). We present a scenario, as an example, where *KM* system is cooperating with Condor. In this case, each *KM* operation is an independent executable job with its appropriate parameters including input, output files and others. Users write the submit description file including resources needed and then use the Condor system to submit it. An example of a submitted file is shown in Figure 10. In this file, a user adds knowledge objects, which are stored in the file *KO1.xml*, of mined knowledge to a *local KM* at a remote site. This mined knowledge already exists on that site or has just built after a mining process. The output file *KMout.xml* contains the meta-knowledge of knowledge objects added and the user uses this information to submit to *meta-knowledge repository* of *KM core*. In this case, the user is responsible for the coherence between *KM core* and *local KM*. In addition, the parameter *nocom* in the argument line shows that user does not use the communication module of the *KM* system.

```
...
universe = java
executable = KM.class
arguments = KM nocom -c put -p KO1.xml
jar_files = KM_Lib.jar
output = KMout.xml
error = KM.error
queue
...
```

Fig. 10 An example of a Condor submit file

An alternative way of exploiting the *KM* system is to use its communication module with different communication middleware. The current *KM* uses Java RMI but it can easily use other communication middleware. In this version, the *KM* runtime includes a set of *KM Daemon* (Figure 11). Each local site has one *KM Daemon* that is responsible for processing local/remote requests. These *KM Daemon* are created at the start by using the primitive ***"init"***. The primitive ***"stop"*** will terminate all the *KM Daemons*. A *KM* application can send request to one or many remote sites. As shown in Figure 10, for example, the application find firstly all the meta-knowledge needed via primitive ***"find"*** (Figure 11a). This action is composed

of four steps: a request is sent to the Host (1) to look for the meta-knowledge needed. Then, this will be retrieved (2) and sent back to the source site (3), and it extracts the results as meta-knowledge objects (4). The application extracts knowledge via primitive ***"retrieve"*** (Figure 11b). This action is also composed of four steps: (1) requests are sent to the appropriate sites; (2) retrieve the knowledge found at each site; (3) sent back to the source site via KM Daemon; (4) extracts results as knowledge objects.

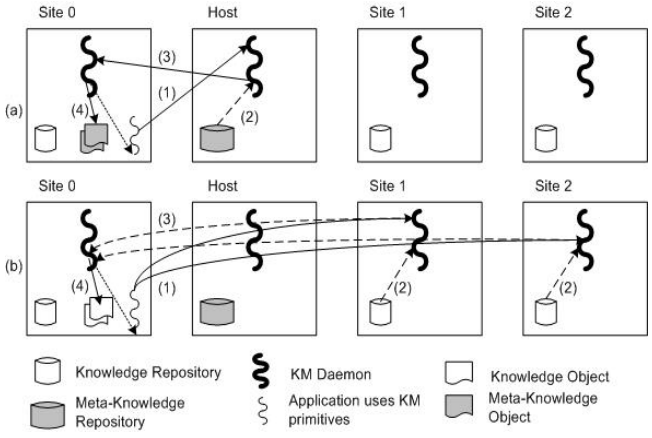


Fig. 11 An example of using Knowledge Map

Another issue of the *KM* implementation is the creation information of representatives in the integration case (c.f. 3.4). Normally there are two kinds of links: one level integration and multi-level integration. In the first kind, the creation information of a representative only contains integration information from one sub-level that is its direct children. For example, as shown in Figure 6a, cluster 111 is integrated from three sub-clusters (222, 333, 444) of which information are registered in creation information of cluster 111. Meanwhile, the cluster 333 is also integrated from two other clusters (555 and 666) but this information is not shown in the creation information of the cluster 111. The advantage of this approach is its simplicity and the saving of storage capacity used of creation information. However, a global search in each related local *KM* is needed to retrieve all sub-levels of integration in this case. Our *KM* system is implemented with a multi-level integration of creation information. In this approach, all integration levels of a representative are in its creation information (Figure 6b).

6. Evaluation and Discussion

We are using this *KM* in our framework [19][20]. It is difficult to evaluate our approach by comparing it to other systems because it is unique so far. Therefore, we evaluate different aspects of the system architecture for supporting the management, mapping, representing and retrieving the knowledge.

First, we evaluate the complexity of search/retrieve the knowledge object of the system. This operation includes two parts: searching relative concept and search/retrieve the knowledge. Let N be the number of *concept tree* entries and n be the number of *concept nodes* for each *concept tree*. The complexity of the first part is $O(\log N + \log n)$ because the concept tree entries are indexed according to the B+tree model. However, the number of concept entries as well as of concept nodes of a concept tree is smaller compared to the number of knowledge entries. So this complexity depends strongly on the cost of search/retrieve operations. Let M be the number of meta-knowledge entries in the *KM core*, so the complexity of searching a meta-knowledge entry at this level is $O(\log M)$. The complexity of retrieving a knowledge object depends on the number of knowledge entries m in *local KM*. Therefore, this complexity is $O(\log M + C\log m)$, where C is the communication cost.

Next, we discuss the knowledge map architecture. Firstly, the structure of *concept tree* is based on the concept map [23], which is one of the advantages of this model. We can avoid the problem of semantic ambiguity as well as reduce the domain search to improve the speed and accuracy of the results. In our current version, the concept tree is implemented at each site. The advantage of this approach is to reduce the communication cost of searching/retrieving task but the communication cost is high for updating task. However, the frequency of this updating task is very low compared with the frequency of the searching/retrieving tasks. Secondly, the division of knowledge map into two main components (local and core) has some advantages: (i) the core component acts as a summary map of knowledge and it is a representation of knowledge about knowledge when combined with local *KM*; (ii) avoiding the problem of having the whole knowledge on one master site (or server), which is not feasible in very large distributed system such as Grid. By representing meta-knowledge in their relationship links, the goal is to provide an integration view of this knowledge.

Moreover, our *KM* system offers a knowledge map with flexible and dynamic architecture where users can easily update the *concept tree* repository as well as meta-knowledge entries. The current index technique used in a rule representative is an inverted list. However, we can improve it without affecting to whole system structure by using other index algorithms as in [22] or applying compressed technique as discussed in [32]. Moreover, flexible and dynamic features are also reflected by mapping a knowledge to a *knowledge object*. The goal

here is to provide a portable approach where knowledge object can be represented by different techniques such as an entity, an XML-based record, or a record of database, etc.

Although the implementation of the creation information of a representative might not be optimal for the storage capacity used, it takes an important advantage in the communication cost compared to one-level approach in retrieving the whole integration links. For example, we analyze these costs for two topologies (Figure 12): *flat tree* and *binary tree*. We assume that one representative is integrated by N elements and the information size of each element is 32 bytes. In the first topology, information of all elements are stored in its creation information for both cases: one level and multi-level. There is no communication cost and the storage size is $32 \times N$ bytes. In the second topology, the storage size at this representative (root of the tree) is only 32×2 bytes for one-level and $32 \times N$ bytes for multi-level. Furthermore, in one-level case, the storage size at each site (not root) in the tree is always 32×2 bytes except sites at the leaf level. In the multi-level case, this size of a site at the level h is $32 \times n_h$ bytes with $n_h = 2(n_{h-1} + 1)$ and $n_0 = 0$. However, there is no communication cost needed for multi-level case, all integration links are in this representative. In order to evaluate this cost for one-level, we assume that the communication is executed in parallel at each level of binary tree with the same latency between two sites and the searching time at each site is negligible compared with communication time. This means that all sites at the same level, each one sends two requests to its two children (one request/child), will receive their replies at the same time. So the communication cost is evaluated by: $2 \times l\log_2 N$ where l is the communication latency between two sites or more general is $O(l\log_p N)$ with p depends on the chosen topology. In the Grid environment, the communication latency and the number of participating sites are important factors affecting the overall performance of distributed tasks.

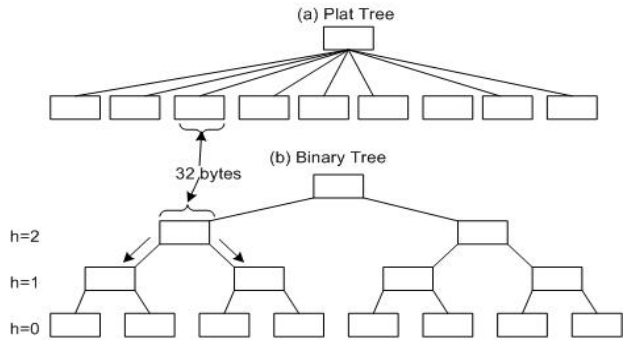


Fig. 12 Different topology of integration

7. Conclusion

In this paper, we presented an architecture of the knowledge map layer. This new approach aims at managing effectively the mined knowledge on large scale distributed platforms. The purpose of this research is to provide a knowledge map to facilitate the management of the results as well as to provide a viable environment for the *DDM* applications.

Throughout evaluations of each component and it function, we can conclude that knowledge map is an efficient and flexible system in a large and distributed environment. It satisfies the needs for managing, exploring, and retrieving the mined knowledge of *DDM* in large distributed environment. This knowledge map is integrated in the ADMIRE framework. Experimental results on real-world applications are also being produced [20] and this will allow us to test and evaluate deeply the system robustness and the distributed data mining approaches at very large scale.

References

[1] S. S. R. Abidi and Cheah Yu-N, *A Convergence of Knowledge Management and Data Mining: Towards "Knowledge-Driven" Strategic Services*, 3rd International Conference on the Practical Applications of Knowledge Management, Manchester, 2000

[2] B. G. Buchanan and E.H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of The Standford Heuristic Programming Projects*, Reading, MA: Addison-Wesley, 1984.

[3] T. Buzan and B. Buzan, *The Mind Map Book*, Plume, 1996.

[4] M. Cannataro, D. Talia and P. Trunfio, *Distributed Data Mining on the Grid*, Future Generation Computer Systems, North-Holland vol. 18, no. 8, pp. 1101-1112.

[5] S. M Chen, J-S. Ke and J-F. Chang, *Knowledge Representation Using Fuzzy Petri Nets*, IEEE transaction on Knowledge and Data Engineering vol.2, no.3, 1990, 311-319

[6] http://www.cs.wisc.edu/condor/

[7] T. H Davenport and L. Prusak, *Working Knowledge*, Havard Business School Press, 1997.

[8] Y. Deng, S-K. Chang, *A G-Net model for Knowledge Representation and Reasoning*, IEEE transaction on Knowledge and Data Engineering vol.2, no.3, 1990, 295-310

[9] M. H. Dunham, *Data Mining Introductory and Advanced Topics*, Prentice Hall, 2002.

[10] M. J. Eppler *Making Knowledge Visible through Intranet Knowledge Maps: Concepts, Elements, Cases* Proceedings of the 34th Hawaii International Conference on System Sciences, 2001.

[11] C. Faloutsos and K. Lin, *FastMap: A Fast Algorithm for Indexing, Data-Ming and Visualization of Traditional and*

Multimedia Datasets, Proceedings of SIGMOD'95 Conference, 1995, 163-174.
[12] D. Gasevic, D. Djuric and V. Devedzic, *Model Driven Architecture and Ontology Development* Springer-Verlag, 2006, 46-57
[13] S. K. Gupta, V. Bhatnagar and S.K. Wasan, *A propasal for Data Mining Management System*, Integrating Data Mining and Knowledge Management Workshop, IEEE ICDM, 2001.
[14] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd ed.Morgan Kaufmann Publishers, 2006.
[15] http://java.sun.com/javase/technologies/
[16] H. Kargupta and P. Chan, *Advances in distributed and Parallel Knowledge Discovery*, 1st ed. AAAI Press/The MIT Press, London, 2000
[17] J-C. Silva, C. Giannella, R. Bhargava, H. Kargupta, and M. Klusch, *Distributed Data Mining and Agents* International Journal of Engineering Applications of Artificial Intelligence, 18 (7), Elsevier Science, 2005.
[18] N-A. Le-Khac, M.T. Kechadi, J. Carthy *ADMIRE framework: Distributed data mining on data grid platforms*, Proceedings of 1st International Conference on Software and Data Technologies ICSOFT'06,2006, 67-72
[19] N-A. Le-Khac, Lamine M. Aouad and M.T. Kechadi *Knowledge Map: Toward a new approach supporting the knowledge management in Distributed Data Mining*, KUI Workshop, IEEE International Conference on Autonomic and Autonomous Systems ICAS'07, Athens, Greece,2007.
[20] Lamine M. Aouad, N-A. Le-Khac and M.T. Kechadi *Variance-based Clustering Technique for Distributed Data Mining Applications*, International Conference on Data Mining (DMIN'07), USA, 2007
[21] F. Lin and C.M. Hsueh, *Knowledge map creation and maintenance for virtual communities of practice*, Journal of Information Processing and Management, vol. 42, 2006, 551-568.
[22] M. Martynov and B. Novikov, *An Indexing Algorithm for Text Retrieval*, Proceedings of the International Workshop on Advances in Databases and Information system (ADBIS'96), Moscow, 1996, 171-175.
[23] J.D. Novak and D.B. Gowin, *Learning how to learn*, Cambridge University Press, 1984.
[24] M.C.F. Oliveira and H. Levkowitz, *From Visual Data Exploration to Visual Data Mining: A survey*, IEEE transaction on visualization and computer graphics vol.9, no.3, 2003, 378-394
[25] J-L. Peterson, *"Petri Nets"*, Journal of ACM Computing Surveys vol 9 no.3, 1977, 223252.
[26] http://www.openpbs.org/
[27] P-N. Tan, M. Steinbach and V. Kumar, *Introduction to Data Mining*,1st ed. Pearson Education, 2006.
[28] A. Veloso, B. Possas, W. Meira and M. B. Carvalho, *Knowledge Management in Association Rule Mining*, Integrating Data Mining and Knowledge Management Workshop, IEEE ICDM, 2001
[29] M.N. Wexler, *The who, what and why of knowledge mapping*, Journal of Knowledge Management, vol. 5, 2001, 249-263.
[30] J. Zobel, A. Moffat, R. Sacks-Davis *An efficient indexing technique for full-text database systems*, Proceeding of the 18th VLDB Conference Vancouver, British Columbia, Canada, 1992, 352-362.
[31] J. Zobel, A. Moffat, R. Sacks-Davis *Searching Large Lexicons for Partially Specified Terms using Compressed Inverted Files*, Proceeding of the 19th VLDB Conference Dublin, Ireland, 1993, 290-301.
[32] J. Zobel and A. Moffat *Inverted Files for Text Search Engines*, Journal of ACM Computing Surveys, Vol. 38, No.2, Article 6, 2006