Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

VDoc+: a virtual document based approach for matching large ontologies using MapReduce*

Hang ZHANG[†1,2], Wei HU[†‡1,2], Yu-zhong QU[1,2]

([1]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China*)

([2]*Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China*)

[†]E-mail: hangzhang@smail.nju.edu.cn; whu@nju.edu.cn

Received Aug. 5, 2011; Revision accepted Jan. 18, 2012; Crosschecked Feb. 27, 2012

Abstract: Many ontologies have been published on the Semantic Web, to be shared to describe resources. Among them, large ontologies of real-world areas have the scalability problem in presenting semantic technologies such as ontology matching (OM). This either suffers from too long run time or has strong hypotheses on the running environment. To deal with this issue, we propose a three-stage MapReduce-based approach V-Doc+ for matching large ontologies, based on the MapReduce framework and virtual document technique. Specifically, two MapReduce processes are performed in the first stage to extract the textual descriptions of named entities (classes, properties, and instances) and blank nodes, respectively. In the second stage, the extracted descriptions are exchanged with neighbors in Resource Description Framework (RDF) graphs to construct virtual documents. This extraction process also benefits from the MapReduce-based implementation. A word-weight-based partitioning method is proposed in the third stage to conduct parallel similarity calculation using the term frequency–inverse document frequency (TF-IDF) model. Experimental results on two large-scale real datasets and the benchmark testbed from Ontology Alignment Evaluation Initiative (OAEI) are reported, showing that the proposed approach significantly reduces the run time with minor loss in precision and recall.

Key words: Ontology matching, Virtual document, MapReduce, TF-IDF, Semantic Web
doi:10.1631/jzus.C1101007 **Document code:** A **CLC number:** TP311

1 Introduction

The Semantic Web is an ongoing effort made in the World Wide Web Consortium (W3C) Semantic Web Activity to enable data integration and sharing among different applications and parties. At present, many data producers, e.g., the Food and Agriculture Organization of the United Nations (FAO) (van Hage *et al*., 2010) and the Foundational Model of Anatomy ontology (FMA) (Rosse and Mejino, 2008), have modeled their own ontologies. The wide use of ontologies carries a practical problem; that is, in the same or overlapped domains there usually exist multiple ontologies that have heterogeneous classes, properties, and instances (in this paper, they are all referred to as entities). Ontology matching (OM) is one solution proposed here, to cope with this heterogeneity.

A considerable number of approaches have been presented to tackle the OM problem, some of which showed high precision and recall on certain synthetic or real-world datasets. However, a powerful matching algorithm often leads to long run time. Referring to the OAEI 2010 Report (Euzenat *et al*., 2010), many OM tools suffered from long run time when matching large ontologies. Some of them even took several weeks to complete the task. Modern parallel computation architectures, e.g., MapReduce (Dean and Ghemawat, 2008), show amazing computing power and have been successfully applied to a variety of fields. This inspires us to consider the combination of MapReduce-based techniques for dealing with the scalability issue on OM.

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61003018), the Natural Science Foundation of Jiangsu Province, China (No. BK2011189), and the National Social Science Foundation of China (No. 11AZD121)

1.1 Our contributions

In this paper, we propose a parallel OM approach called V-Doc+, which is based on MapReduce and virtual documents (Qu *et al*., 2006). We summarize our contributions as follows:

1. We propose a simple process for extracting the descriptions of named entities and an iterative process for blank nodes. Both of them use MapReduce.

2. We introduce two MapReduce processes to exchange the descriptions with neighbors in Resource Description Framework (RDF) graphs and construct virtual documents.

3. We design a word-weight-based partitioning approach for calculating in parallel the term frequency–inverse document frequency (TF-IDF) similarities between entities.

4. We systematically evaluate our approach on both large-scale and benchmark datasets. Experimental results demonstrate the feasibility of the proposed approach.

Note that the proposed approach supplements our previous conference paper (Zhang *et al*., 2011) in the following three aspects: (1) We complement more background on method formulation, and clearly separate the formulation from the MapReduce-based implementation; (2) We improve the word-weight-based partitioning method to make it more flexible to fit different preferences on efficiency and effectiveness; (3) We comprehensively evaluate our approach, especially for varied parameters. We also conduct new experiments on the benchmark testbed to prove that the MapReduce-based implementation causes only limited loss in accuracy.

1.2 Related work

As summarized by Shvaiko and Euzenat (2008), one of the 10 challenges for OM in the near future is how to deal with large ontologies having thousands or even hundreds of thousands of entities. Mork and Bernstein (2004) simplified some original algorithms to solve the OM problem, but this may not lead to a good recall. Hu *et al.* (2008) proposed a divide-and-conquer based partitioning approach for large ontologies, and integrated it into the OM tool Falcon-AO. Wang *et al.* (2011) used the positive and negative reduction anchors but did not partition large ontologies. In general, all these works improved efficiency on a single-node computer. Rahm (2011) surveyed current studies on large-scale schema matching and OM.

MapReduce, which uses the parallel computing architecture, is relatively new to schema or ontology matching. Vernica *et al.* (2010) proposed a new approach that computes set-similarity joins using MapReduce, showing a surprising speedup on large-scale datasets. However, this approach cannot be directly used for OM due to the incompatibility between ontology and the relational data model. Gross *et al.* (2010) compared two kinds of parallelization on OM. But this work in fact parallelized job queues in matching workflow services, which is not easy to implement.

2 Preliminaries

In this section, we first define OM, and then give a brief introduction of MapReduce.

2.1 Ontology matching

Ontology matching (also referred to as aligning or mapping) aims at finding mappings (also called matches or correspondences) between similar entities from different ontologies. The mappings may represent equivalence or other relations, e.g., disjointness or subsumption, between entities. Inspired by the definition in Euzenat and Shvaiko (2007), we define OM as follows.

Let O, O' be two ontologies. Matching O with O' finds a set of mappings $M=\{m_1, m_2, \ldots, m_n\}$. Each m_i (i=1, 2, …, n) is a 5-tuple: <id, e, e', r, sim>, where id is a unique identifier, e is an entity in O, e' is an entity in O', r is an equivalence, subsumption, or disjointness relation that holds between e and e', and sim is a confidence value (similarity) between e and e' in the range (0, 1).

Specifically, 'large ontologies' in this paper means the large number of entities in the two ontologies that are matched (e.g., more than one thousand entities in each of the ontologies), rather than a large number of small ontologies.

2.2 MapReduce

Recently, MapReduce has become a very popular framework for parallel computation on a number of computing nodes (Dean and Ghemawat, 2008). The unit of the computing process in MapReduce is

job, and a job consists of two main phases: Map and Reduce. Map feeds the data from the sources and splits records into key-value pairs. These pairs are partitioned into different reducers according to the keys. Before a reducer handles the data, the partitioned pairs are sorted in terms of their keys. All values sharing the same key are clustered into the same set. MapReduce also provides programmers extensibility. Based on the interfaces, programmers are allowed to assign rules on how to partition key-value pairs and how to sort them by keys. These partitioning rules offer benefits for balancing the workload. Additionally, a combiner that performs as a 'local reducer' can be rewritten before the mapper results are distributed.

MapReduce is a powerful tool that uses parallel computation, and has been widely used in various fields. In OM, the most time-consuming process is to match every two entities. For instance, calculating the similarity for every pair of an N-size set of entities must be repeated $N(N-1)/2$ times. With the help of MapReduce, the N-size set could be partitioned into several subsets and different subsets can be calculated in different computing nodes. Therefore, the run time would be significantly reduced.

Another important use of MapReduce in OM is to solve the set-join problem. For example, for a class c, to obtain all rdfs:comment(s) of c, we need to find all RDF statements that satisfy <c, rdfs:comment, l>, where l is a literal. To apply MapReduce we can join c with its related RDF statements in a specific reducer.

3 V-Doc+

To match large ontologies under a parallel architecture, we propose a linguistic method extended from the virtual document technique (Watters, 1999). To involve the intended meaning of an entity, the virtual document (of the entity) takes both the local descriptions and the neighboring information. Meanwhile, the words in the descriptions should be weighted to reflect the proportion of information. In our approach, the construction of virtual documents and the method for matching them are described first, and then this construction is parallelized under the MapReduce framework, to improve the computational efficiency.

Specifically, the proposed approach, called V-Doc+, is constituted that includes three end-to-end MapReduce stages: extracting textual descriptions of named entities and blank nodes, exchanging information among neighbors in RDF graphs to construct virtual documents, and calculating similarities under the TF-IDF model (Salton and McGill, 1986).

3.1 Constructing virtual documents

The RDF graph model occupies an important place in the foundation of Semantic Web ontologies, in which the definitions of formulations and operations on constructing virtual documents are specified. To achieve these formulations, we design the parallel methods based on MapReduce.

3.1.1 Formulation

An RDF graph essentially consists of a number of RDF triples. An RDF triple, also called a statement, is represented in the order of <subject, predicate, object>. Each of the uniform resource identifier (URI) references (which must be entities), literals, and blank nodes, whose identification forms cannot be separated, becomes a node in an RDF graph. Note that, a literal cannot be a subject and a property (or predicate) is always an entity.

In information retrieval, a document is considered as a collection of words directly obtained or inferred from the textual content of the document. These weighted words, treated as the dimensions in the vector, represent their importance to the document. The words involved should be normalized first in a preprocessing step, e.g., parsing literals into tokens through stemming and eliminating stop words. In this study, we make a virtual document with a collection of weighted words, in which weights are rational numbers.

Briefly speaking, the description of a literal node is a collection of words extracted from its lexical form. For a named entity, the words are derived from the local name of the URI (i.e., a substring intercepted after the last slash '/' or hash '#' mark), rdfs:comment(s), rdfs:label(s), and other possible properties. We define the textual description Desc() for a named entity e as follows:

$$\begin{aligned}\mathrm{Desc}(e) = {} & \alpha_1 \mathrm{LN}(e) + \alpha_2 \mathrm{Label}(e) \\ & + \alpha_3 \mathrm{Comment}(e) + \alpha_4 \mathrm{Other}(e),\end{aligned} \tag{1}$$

where LN(e), Label(e), Comment(e), and Other(e) denote the collection of words extracted from the local name, rdfs:label(s), rdfs:comment(s), and other annotations of e, respectively, and α_1, α_2, α_3, and α_4, indicating the weights of the categories, are rational numbers in [0, 1]. The operator '+' denotes the merging of collections of words on both sides.

For a blank node b, its description (i.e., a collection of words) is obtained from the descriptions of the forward graph neighbors. Consequently, an iterative equation is given for the convergence solution. We simply define B as the set of all blank nodes, b as a blank node in B, and s as a statement involving b. We define the textual description Desc() of b as follows:

$$\mathrm{Desc}_1(b)=\sum_{\mathrm{subj}(s)=b}\mathrm{Desc}(\mathrm{pred}(s))+\sum_{\substack{\mathrm{subj}(s)=b\\ \mathrm{obj}(s)\notin B}}\mathrm{Desc}(\mathrm{obj}(s)), \tag{2}$$

$$\mathrm{Desc}_{k+1}(b)=\beta\left(\mathrm{Desc}_1(b)+\sum_{\substack{\mathrm{subj}(s)=b\\ \mathrm{obj}(s)\in B}}\mathrm{Desc}_k(\mathrm{obj}(s))\right),\ k\geq 1, \tag{3}$$

where subj(s), pred(s), and obj(s) stand for the subject, predicate, and object of s, respectively. β is an attenuation coefficient in the range [0, 1]. Two special situations should be considered: (1) The cycle definitions on blank nodes are not taken into account. (2) To avoid heterogeneity, we introduce a transformation rule that changes the expression of the rdf:List structure; i.e., the list and all the members are expressed by the rdfs:member property rather than directly using RDF collection vocabularies.

To better express the descriptions of the virtual documents, we group the descriptions of neighbors. Three neighboring operations are defined to present different types of neighbors. For an entity e, we denote SN(e) (subject neighboring operation) as the nodes placed at the subject position of RDF triples. Similarly, PN(e) (predicate neighboring operation) and ON(e) (object neighboring operation) are defined as the nodes in the predicate position and object position of triples, respectively. Then we formalize SN(e), PN(e), and ON(e) as follows:

$$\mathrm{SN}(e)=\bigcup_{\mathrm{subj}(s)=e}\{\mathrm{pred}(s),\mathrm{obj}(s)\}, \tag{4}$$

$$\mathrm{PN}(e)=\bigcup_{\substack{\mathrm{pred}(s)=e\\ \mathrm{subj}(s)\notin B}}\{\mathrm{subj}(s),\mathrm{obj}(s)\}, \tag{5}$$

$$\mathrm{ON}(e)=\bigcup_{\substack{\mathrm{obj}(s)=e\\ \mathrm{subj}(s)\notin B}}\{\mathrm{subj}(s),\mathrm{pred}(s)\}. \tag{6}$$

Based upon the formulations described above and the neighboring operations, the virtual document VD(e) for an entity e is specifically defined as

$$\mathrm{VD}(e)=\mathrm{Desc}(e)+\gamma\,\mathrm{Neigh}(e), \tag{7}$$

$$\mathrm{Neigh}(e)=\sum_{e'\in\mathrm{SN}(e)}\mathrm{Desc}(e')+\sum_{e'\in\mathrm{PN}(e)}\mathrm{Desc}(e')+\sum_{e'\in\mathrm{ON}(e)}\mathrm{Desc}(e'), \tag{8}$$

where γ is an attenuation coefficient in the range [0, 1] and Neigh(e) denotes the summarization of neighbors' descriptions of e. Consequently, the content of a virtual document becomes its local descriptions combined with the descriptions of its neighbors.

3.1.2 MapReduce-based implementation

As described in Section 3.1.1, the construction of virtual documents consists of two MapReduce sub-stages: (1) extracting textual descriptions for named entities and blank nodes; (2) exchanging the descriptions with neighbors to construct virtual documents.

Fig. 1 presents an example data flow for constructing descriptions of entities. In this process, we ignore blank nodes. The records from input involve classes, properties, instances, and RDF statements. In Fig. 1, the Map function extracts the identifiers to check if they are RDF statements. For each record of the statement, the Map function replaces the identifier with its subject, while classes and properties are directly emitted. For example, a record of RDF statement (s_1, (c_2, p_1, l_1)) is emitted by changing the key s_1 to c_2. However, a record of class (c_1, class$_1$) is emitted without any changes. After aggregating by keys for each class c or property p, all related RDF statements are partitioned to the same reducer that can obtain the description Desc(c) or Desc(p). This is shown in Eq. (1).

In the case of having no local description, blank nodes obtain their descriptions from neighbors, according to Eqs. (2) and (3). However, MapReduce is not designed explicitly for handling this recursive

problem. Thus, we transform each step of Eqs. (2) and (3) to a MapReduce process.

First, we establish a blank node structure to extend the simple transmission unit for carrying the information of remaining nodes. Thus, the record of the blank node b is extended from the simple structure Desc(b) to (Desc(b), {neb$_1$, neb$_2$, …, neb$_n$}), where neb$_i$ denotes a neighboring node to be included.

Then, we build a k-times-repeated MapReduce process to calculate the descriptions for blank nodes, in which the input of the Map function is derived from the output of the first sub-stage. For each neb$_i$ in (Desc(b), {neb$_1$, neb$_2$, …, neb$_n$}), the Map function generates a new key-value pair, where the key and value are neb$_i$ and (Desc(b), {}), respectively. Thus, a record of the blank node may be replicated as many times as the number of remaining nodes. The Reduce function aggregates a blank node b with the related entities or RDF statements. For each related entity e, the description of the blank node is updated by β^kDesc(e), where k denotes the number of times by which the process has been repeated. Meanwhile, in the statement s whose subject is b, we add its object to the remaining nodes of b.

A brief example is illustrated below. In Fig. 2, the record of blank node b_1 has two remaining nodes $\{b_1, c_5\}$ and is thus partitioned with the statement (b_1, p_6, c_1) and Desc(c_5), respectively. In the reducer loading (b_1, p_6, c_1), p_6 and c_1 are added to the remaining nodes of b_1. Meanwhile, Desc(b_1) is updated by β^kDesc(c_5) in another reducer. Notice that due to the existence of duplicated records in the output, we need a combiner to combine the results.

Note again that the cycle descriptions are not taken into account. We store the route of nodes that have been calculated and ignore all new nodes appearing in the route. The process of constructing descriptions for blank nodes should be repeated k times. According to our experience, five iterations are usually enough for convergence.

However, the complete construction of a virtual document needs both local descriptions and neighboring information. The related process contains two MapReduce stages. The first stage is to inform each entity the locations of its neighbors. In the second stage, the content of its description is enriched through exchanging the descriptions with the neighbor nodes.

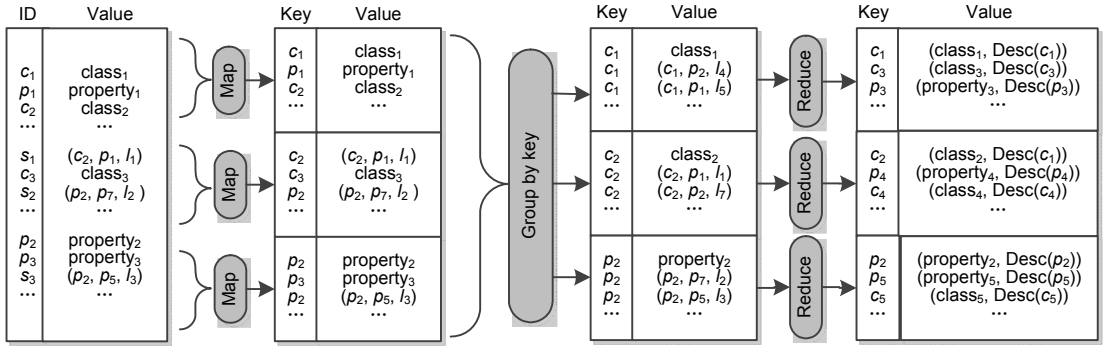

Fig. 1 Data flow of extracting textual descriptions for named entities

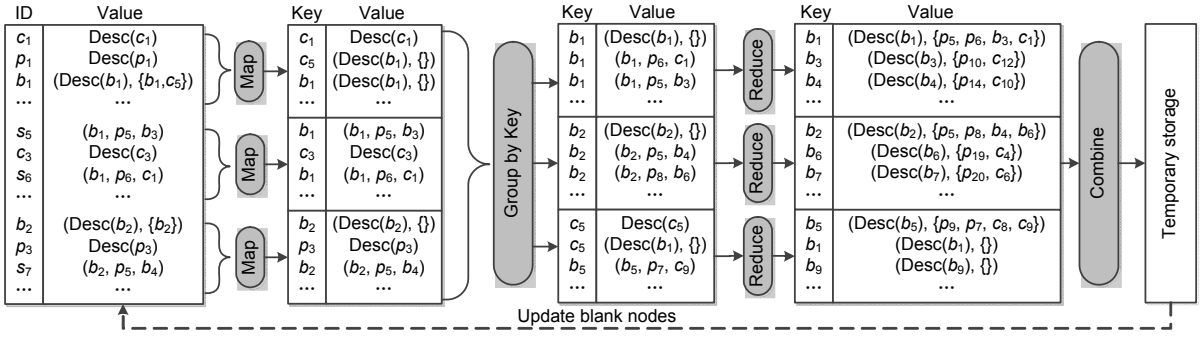

Fig. 2 Data flow of extracting textual descriptions for blank nodes

Fig. 3 exemplifies a data flow for exchanging neighboring descriptions. The Map function at the first stage generates three new key-value pairs (s, {p, o}), (p, {s, o}), and (o, {s, p}) for the RDF statement <s, p, o>. After that, all neighbors of each entity e and blank node b can be obtained in the Reduce function. For example, an RDF statement <b_1, p_5, b_3> is a record from input; after mapping, (b_1, {p_5, b_3}), (p_5, {b_1, b_3}), and (b_3, {b_1, p_5}) are partitioned into three different reducers. In the reducer loading <b_1, p_5, b_3>, b_1 gets to know that there exist two neighbors p_5 and b_3 and adds them into a temporary structure. The outputs of reducers are stored in a temporary storage to be used as an input of the second stage. Having known the locations of neighbors, every node sends its description in the second stage. For the record (Desc(c_1), {b_2, p_4, p_1}) in the temporary storage, c_1 sends its description Desc(c_1) to b_2, p_4, and p_1, respectively. Thus, every entity obtains all the descriptions of the neighbors and updates its own description.

Because of frequency skew, some reducers may encounter an unbalanced workload, which makes the whole calculation process significantly delayed. As a solution, we count how many times entities appear in RDF statements and partition those with the highest frequency averagely.

3.2 Matching virtual documents

The vector space model (VSM) and the prevalent TF-IDF technique are used to calculate the similarities between virtual documents. We also develop the MapReduce processes to enable parallel computation.

3.2.1 Formulation

In the vector space, the coordinate basis vector consists of the words occurring in ontologies. The number of the unique words determines the dimensionality of the vector. Further, since a collection of words with a weighting scheme constitutes the virtual document, each virtual document can be seen as a vector. The words reflect the importance of information for a specific virtual document, and the components of the vectors become the scores corresponding to the words' weights. The scores give the degree of importance between the words and the virtual documents. Consequently, the characteristics of virtual documents include the words and their weights.

To select words that represent each virtual document, we adopt the TF-IDF technique to optimize the representation of vectors. The word weight is calculated using the following equations:

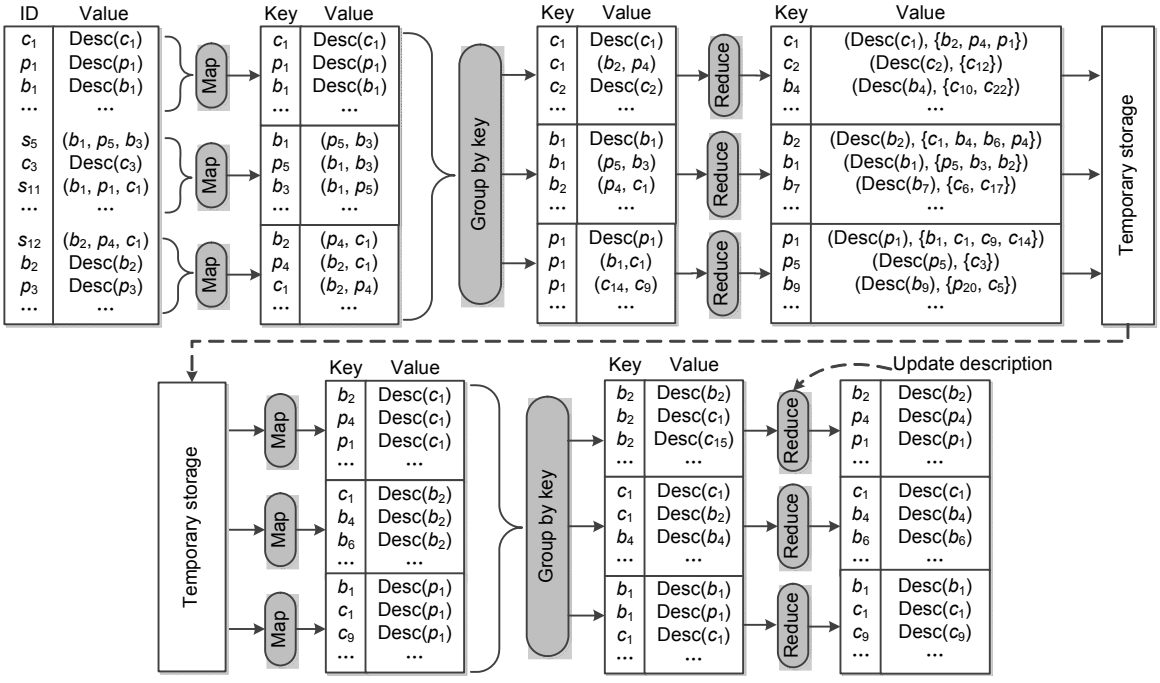

Fig. 3 Data flow of exchanging descriptions of neighbors

$$\text{WordScore} = \text{TF} \cdot \text{IDF}, \tag{9}$$

$$\text{TF} = w / W, \tag{10}$$

$$\text{IDF} = \frac{1}{2}[1 + \log_2(N / n)]. \tag{11}$$

In Eq. (10), the refined word occurrence in the description of a virtual document is defined by w, whereas W denotes the number of all words' occurrences in a specific virtual document. For each of the words, in Eq. (11), n specifics the number of virtual documents containing the word and N means the number of all the virtual documents. Therefore, the correlation between a word and a virtual document can be reflected.

Finally, the cosine similarity is utilized in the measurement of the similarity between two entities. The measure is as follows:

$$\text{sim}(e_1, e_2) = \frac{\text{VD}(e_1) \times \text{VD}(e_2)}{|\text{VD}(e_1)| \cdot |\text{VD}(e_2)|}. \tag{12}$$

Thus, if the virtual documents of two entities share no words, the similarity would be 0; if they are completely equal, it should be 1.0.

3.2.2 MapReduce-based implementation

To filter unnecessary matching and reduce the computation space, the implementation of similarity computation on the MapReduce framework is important. It involves a word-weight-based partitioning. For a description $\{(\text{word}_1, \text{score}_1), (\text{word}_2, \text{score}_2), \ldots, (\text{word}_n, \text{score}_n)\}$, we normalize the scores to [0, 1] and rank (word, score) pairs to satisfy $\text{score}_1 \geq \text{score}_2 \geq \ldots \geq \text{score}_n$. Then we define the important words as the set of textual words, $\{\text{word}_1, \text{word}_2, \ldots, \text{word}_i\}$, where i is a minimal integer satisfying $\text{score}_1 + \text{score}_2 + \ldots + \text{score}_i \geq \delta$. Here, δ is fixed in [0, 1].

In fact, to choose the value of δ we should consider the dataset size, accuracy, and run time. Basically speaking, a higher accuracy requires a larger value of δ and a longer run time. A value of 0.8 is set when the size of the dataset exceeds a predefined value given by the user and a value of 0.99 is set when dealing with a small-scale dataset, because an acceptable run time can be obtained for small datasets even if a very high value of δ is assigned. This does not hold true for large datasets. Section 4 gives an experiment on the choice of δ.

Fig. 4 gives a brief explanation of the process. Each mapper in the figure ranks the words and extracts the important words into keys. For instance, suppose that we extract three important words word_1, word_2, and word_3 for entity c_1 (Fig. 4). Three key-value pairs (word_1, $\text{Desc}(c_1)$), (word_2, $\text{Desc}(c_1)$), and (word_3, $\text{Desc}(c_1)$) are generated and partitioned. Then, the descriptions of entities c_1, c_8, and c_5 are grouped into the same reducer as sharing the same key word_1. Finally, we use Eq. (12) to calculate the similarity between each two of them.

To balance the workload, we load the frequency of words into the memory and construct a customized partitioner to choose the corresponding reducer. The words should be distributed evenly in each node according to the frequency.

4 Experimental results

We implemented V-Doc+ based on Hadoop 0.21.0. All the experiments were performed over a 10-node cluster and a Gigabit Ethernet interconnect. The NameNode was equipped with an Intel processor having six 2.8 GHz cores and a 12 MB cache, 32 GB memory, and a 2 TB hard disk. For all JobTrackers

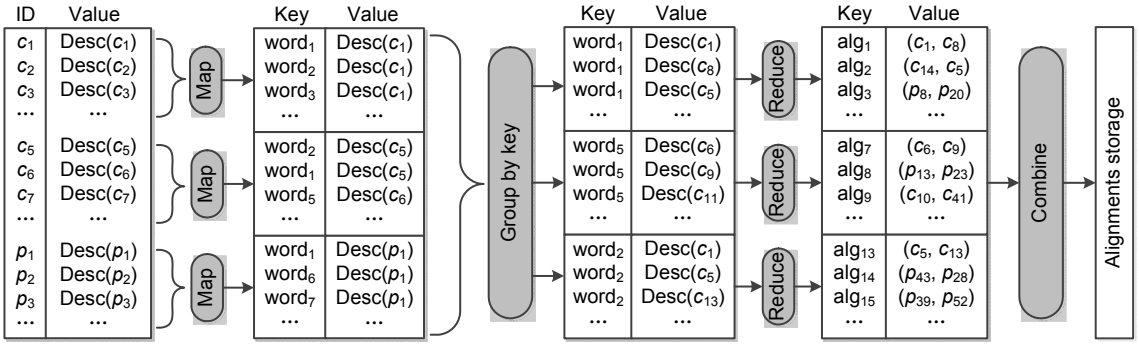

Fig. 4 Data flow of similarity computation

and slaves, the CPUs were Intel Quad cores 2.4 GHz/12 MB cache, and the storage was 24 GB memory and a 2 TB hard disk. For comparison, we also created a non-parallel version called V-Doc (Qu *et al*., 2006).

According to the optimal result generated by experiments having varied parameters, we configured some key parameters as follows: for the stage of description extraction, α_1=1, α_2=0.5, α_3=0.25, α_4=0, and β=0.5; for description exchange, γ=0.1. All the above parameters were set exactly the same as in Qu *et al*. (2006). But for virtual document matching, the newly involved threshold δ varied, and was assigned in terms of specific experiments.

We used the well-known F1-measure to measure the mappings. Let M_c be the set of generated mappings and M_r be the set of reference mappings. The values of precision and recall of M_c with respect to M_r are $|M_c \cap M_r|/|M_c|$ and $|M_c \cap M_r|/|M_r|$, respectively. The value of F1-measure can be computed as

$$\text{F1-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \quad (13)$$

Preprocessing was conducted before starting the experiments. We cached all stop words and entities/words frequency statistics in memory. To distinguish the source of the record easily, the identifiers of entities, blank nodes, and RDF statements were attached with their ontology names.

4.1 Large-scale test

4.1.1 Datasets

V-Doc+ was designed for large-scale real-world OM. We selected two datasets of large size: two Food ontologies from OAEI 2007, and FMA versus GALEN. Food involves two large ontologies, NALT and AGROVOC, which in total contain 70 765 classes/properties. We downloaded the gold standard and assessed with the F1-measure. FMA versus GALEN contains 82 255 classes/properties.

Several facts should be considered: (1) Due to high network cost and repeated job initialization, MapReduce does not have superiority in small-scale cases; (2) The Food ontologies and the results of the participants were given only in OAEI 2007; thus, we cannot obtain new results later than this version in 2007; (3) For FMA versus GALEN, we found neither tools publishing their experimental results nor reference mappings. We tested only run time and speedup of V-Doc+ on this dataset.

4.1.2 Performance on a large-scale test

The task of our evaluation is fourfold. First, it is necessary to test precision, recall, and F1-measure. To test these parameters, we need to investigate several OM tools for comparison. Second, we want to show the reduction of run time as compared to the non-parallel tools. Third, we can specify the speedup on different numbers of computing nodes and the run time percentage for each processing stage. Finally, the variance of overall performance would be exhibited, based on different values of threshold δ chosen in the word-weight-partition process, as this value influences the balance between run time and accuracy.

We compared V-Doc+ with Falcon-AO, DSSim, RiMOM, Prior+, and COMA. Falcon-AO (Hu *et al*., 2008) integrates three matchers: I-Sub, V-Doc, and GMO. For matching large ontologies having limited memory, Falcon-AO gives a divide-and-conquer approach that can partition entities into small blocks and matches these blocks. DSSim (Nagy and Vargas-Vera, 2011) builds a multi-agent system to solve the OM problem by considering uncertainty. The main approach of DSSim is to use different domains for finding ontology mappings. RiMOM (Li *et al*., 2009) integrates multiple matchers to improve effectiveness by combining both linguistic and structural features. Prior+ (Mao *et al*., 2010) is an adaptive OM tool based upon several different techniques, such as IR-based similarity and neural network. In Falcon-AO and COMA (Do and Rahm, 2007), block matching is considered, and a fragment-based matcher is developed to deal with the large OM problem. COMA partitions ontologies in the form of trees. Additionally, we ran V-Doc on Food and FMA versus GALEN. We failed to complete the test for V-Doc because the memory exceeded the maximal configuration.

Fig. 5 shows the results of V-Doc+ and other five matchers on Food with δ=0.8. The precision, recall, and F1-measure of V-Doc+ were worse than those of some other schemes. It is mostly because other tools combine several algorithms but in V-Doc+ only one algorithm is used. In fact, V-Doc+ performed at par or better than the other four tools. This, however, excluded Falcon-AO.

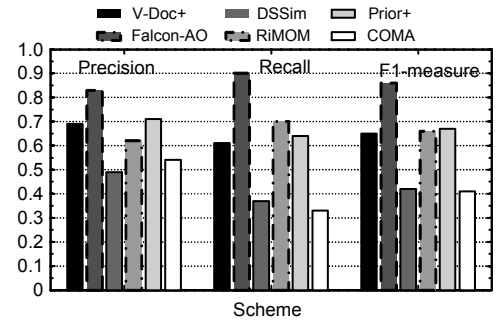

Fig. 5 Precision, recall, and F1-measure comparison on Food with δ=0.8

Table 1 illustrates the run time over a 10-node cluster on Food with δ=0.8. V-Doc+ used only 10 min to complete the task. Among other competitors, Prior+ costed the least, 1.5 h, while DSSim, which was the slowest, took one week. Therefore, though V-Doc+ did not achieve the best F1-measure, the parallelization made it much faster.

Table 1 Run time comparison on Food on a 10-node cluster with δ=0.8

V-Doc+	Falcon-AO	DSSim	RiMOM	Prior+
10 min	6 h	1 week	4 h	1.5 h

We also analyzed the run time at each stage of the whole approach, with the same 10-node cluster and δ=0.8 (Fig. 6). For both Food and FMA versus GALEN, description extraction and similarity calculation took most of the time (both a few minutes).

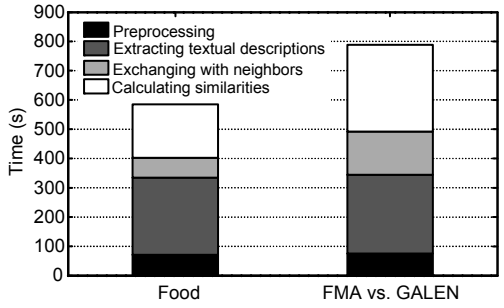

Fig. 6 Run time at each stage on a 10-node cluster with δ=0.8

To evaluate the speedup, we calculated the run time on variably sized clusters. For a test case, we executed it on 2-, 4-, 6-, 8-, and 10-node environments (Fig. 7). The run time kept decreasing as we increased the number of nodes. However, the decrease trend was slowing down. The speedup presented in Fig. 8 reflected a growth trend that kept slowing down. From 8 to 10 nodes, the speedup was almost unchanged. This tells that V-Doc+ may achieve the best efficiency on a 10-node cluster.

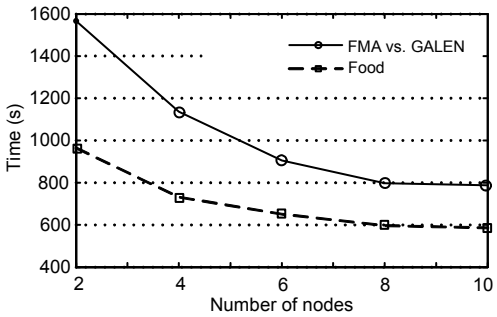

Fig. 7 Run time on variably sized clusters

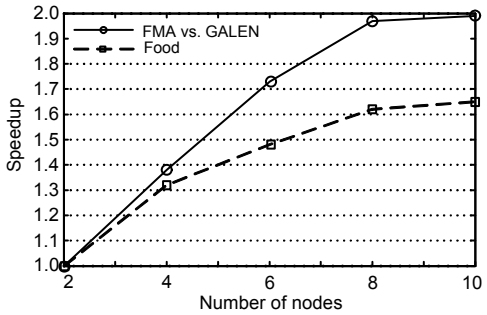

Fig. 8 Speedup on variably sized clusters

According to our statistical data, δ=0.8 was used in most experiments to achieve a balanced performance. In general, increasing δ leads to more key-value records to partition and more mappings to calculate, and thus more abundant output with longer time. We designed an experiment with varied values of δ to demonstrate how this parameter influences the balance between effectiveness and efficiency.

As illustrated in Fig. 9, when δ was smaller than 0.8, the increase of δ resulted in an approximately linear increase of recall and F1-measure, and a slight decrease of precision. When δ was greater than 0.8, the three measures kept almost unchanged.

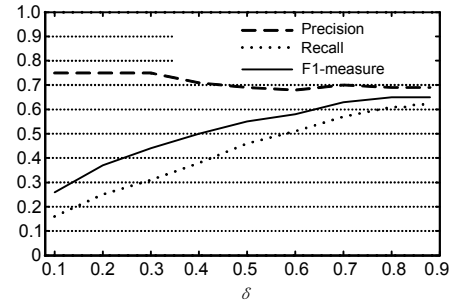

Fig. 9 Precision, recall, and F1-measure on Food with varied δ

Fig. 10 illustrates how the run time changes when varying the value of δ, on a 10-node cluster. The run time increased quite slowly until δ=0.8. The main reason is that the word collection of a virtual document description consists of a small number of words with high weights, but most words have small weights. Also, note that the run time under δ=0.99 for small datasets was not significantly larger than that under a small value of δ, which will be confirmed in Section 4.2. Furthermore, Figs. 9 and 10 together indicate that 0.8 is probably a good value for δ to achieve a tradeoff between effectiveness and efficiency.

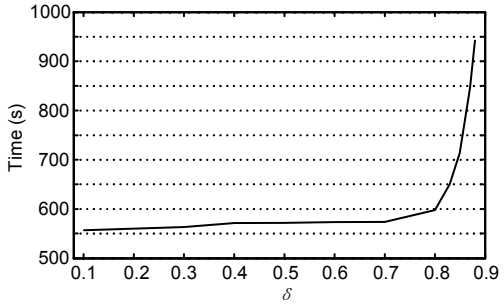

Fig. 10 Run time on Food on a 10-node cluster with varied δ

4.2 Benchmark test

4.2.1 Datasets

We verified the performance of V-Doc+ with experiments on the OAEI 2006 Benchmark testbed. Benchmark provides small-size ontologies to identify the strengths and weaknesses of an OM tool and a way of making comparison between different tools.

In general, the test cases in Benchmark can be divided into five groups: tests 101–104 (the easiest task where the same classes and properties own the literal-similar or same names), tests 201–210 (in which some linguistic features have been discarded, but structural features can be utilized), tests 221–247 (in which the ontologies maintain the linguistic features while changing their structural features), tests 248–266 (in which both structural and linguistic features are suppressed), and tests 301–304 (in which real-life ontologies are included).

4.2.2 Performance on a benchmark test

In this experiment, we compared V-Doc+ with the non-parallel version (i.e., V-Doc) and other tools attending the OAEI 2006 campaign. Here, we still chose δ=0.8 which was used in the large-scale test and δ=0.99 to see the best effectiveness that V-Doc+ can achieve.

Other OM tools in the test were Falcon-AO (Hu *et al*., 2008), AUTOMS (Kotis *et al*., 2006), COMA (Do and Rahm, 2007), DSSim (Nagy and Vargas-Vera, 2011), HMatch (Castano *et al*., 2006), JHU/APL (Bethea *et al*., 2006), Prior+ (Mao *et al*., 2010), and RiMOM (Li *et al*., 2009). The full performance indicators were shown in detail in the final report of OAEI 2006.

Table 2 lists the complete F1-measure. Both δ=0.8 and δ=0.99 are given for V-Doc+. For δ=0.99, the value of F1-measure on each test group was similar to that of V-Doc, especially for the average. This proves that parallelization does not cause much loss in precision or recall. For δ=0.8, the average performance decreased. The main reason is twofold. First, the linguistic and structural features of some test groups, especially tests 248–266, were suppressed,

Table 2 F1-measure comparison for the five groups of test cases in Benchmark

Ontology matching tool	F1-measure					
	#101–104	#201–210	#221–247	#248–266	#301–304	Average
V-Doc+						
δ=0.80	**1.00**	0.80	0.99	0.27	0.69	0.71
δ=0.99	**1.00**	0.83	0.99	0.42	0.68	0.76
V-Doc	**1.00**	0.84	**1.00**	0.41	0.74	0.77
Falcon-AO	**1.00**	0.92	0.99	0.56	0.81	0.84
AUTOMS	0.97	0.74	0.97	0.32	0.77	0.71
COMA	**1.00**	0.95	0.96	0.60	0.73	0.83
DSSim	0.99	0.44	0.99	0.00	**0.82**	0.57
HMatch	0.95	0.44	0.95	0.01	0.54	0.53
JHU/APL	**1.00**	0.64	0.99	0.09	0.21	0.59
Prior+	**1.00**	0.67	0.99	0.05	0.81	0.63
RiMOM	**1.00**	**0.97**	0.99	**0.65**	0.81	**0.87**

The bold numbers specify the best performances

which made the neighboring descriptions important for a high accuracy for this kind of case. Furthermore, decreasing the value of δ implies cutting down the possible words in the neighboring descriptions, thus reducing the recall, or cutting down the possible words that can be matched. In fact, even if δ=0.8, V-Doc+ still exceeded half of the other competitors in terms of average performance. However, some tools that integrate more than one matcher, like RiMOM and COMA, performed better than V-Doc+. This could be a research direction for our future work.

We also counted the run time for V-Doc+ and V-Doc on Benchmark. V-Doc used 8 s on average for a test, while V-Doc+ took 7 s. Thus, MapReduce did not have a significant impact on the benchmark test.

5 Concluding remarks

In this paper, we propose a three-stage parallel OM approach called V-Doc+, which combines the virtual document technique and MapReduce. We compared our approach with five ontology matchers on two large datasets. Experimental results showed that our approach achieved good run time with a minor loss of precision and recall. Further experiments on the Benchmark testbed showed that our MapReduce-based method gained a comparable performance on small datasets. In the future, we will develop new MapReduce-based algorithms, e.g., utilizing ontology structures, to further improve the matching robustness of our algorithms and the accuracy of the mappings.

References

Bethea, W.L., Fink, C.R., Beecher-Deighan, J.S., 2006. JHU/APL Onto-Mapology Results for OAEI 2006. Proc. ISWC Workshop on Ontology Matching, p.144-152.

Castano, S., Ferrara, A., Messa, G., 2006. Results of the HMatch Ontology Matchmaker in OAEI 2006. Proc. ISWC Workshop on Ontology Matching, p.134-143.

Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM*, **51**(1):107-113. [doi:10.1145/1327452.1327492]

Do, H.H., Rahm, E., 2007. Matching large schemas: approaches and evaluation. *Inform. Syst.*, **32**(6):857-885. [doi:10.1016/j.is.2006.09.002]

Euzenat, J., Shvaiko, P., 2007. Ontology Matching. Springer, Heidelberg, Germany. [doi:10.1007/978-3-540-49612-0]

Euzenat, J., Ferrara, A., Meilicke, C., Nikolov, A., Pane, J., Scharffe, F., Shvaiko, P., Stuckenschmidt, H., Šváb-Zamazal, O., Svátek, V., *et al.*, 2010. Results of the Ontology Alignment Evaluation Initiative 2010. Proc. ISWC Workshop on Ontology Matching, p.85-117.

Gross, A., Hartung, M., Kirsten, T., Rahm, E., 2010. On matching large life science ontologies in parallel. *LNCS*, **6254**:35-49. [doi:10.1007/978-3-642-15120-0_4]

Hu, W., Qu, Y.Z., Cheng, G., 2008. Matching large ontologies: a divide-and-conquer approach. *Data Knowl. Eng.*, **67**(1):140-160. [doi:10.1016/j.datak.2008.06.003]

Kotis, K., Valarakos, A.G., Vouros, G.A., 2006. AUTOMS: Automated Ontology Mapping Through Synthesis of Methods. Proc. ISWC Workshop on Ontology Matching, p.96-106.

Li, J.Z., Tang, J., Li, Y., Luo, Q., 2009. RiMOM: a dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng.*, **21**(8):1218-1232. [doi:10.1109/TKDE.2008.202]

Mao, M., Peng, Y.F., Spring, M., 2010. An adaptive ontology mapping approach with neural network based constraint satisfaction. *Web Semant.*, **8**(1):14-25. [doi:10.1016/j.websem.2009.11.002]

Mork, P., Bernstein, P., 2004. Adapting a Generic Match Algorithm to Align Ontologies of Human Anatomy. Proc. 20th Int. Conf. on Data Engineering, p.787-790. [doi:10.1109/ICDE.2004.1320047]

Nagy, M., Vargas-Vera, M., 2011. Multi-agent ontology mapping framework for the semantic Web. *IEEE Trans. Syst. Man Cybern. A*, **41**(4):693-704. [doi:10.1109/TSMCA.2011.2132704]

Qu, Y.Z., Hu, W., Cheng, G., 2006. Constructing Virtual Documents for Ontology Matching. Proc. 15th Int. Conf. on World Wide Web, p.23-31. [doi:10.1145/1135777.1135786]

Rahm, E., 2011. Towards Large-Scale Schema and Ontology Matching. *In*: Bellahsene, Z., Bonifati, A., Rahm, E. (Eds.), Schema Matching and Mapping. Springer, Heidelberg, Germany, p.3-27. [doi:10.1007/978-3-642-16518-4_1]

Rosse, C., Mejino, J.L.V., 2008. The foundational model of anatomy ontology. *Comput. Biol.*, **6**(1):59-117. [doi:10.1007/978-1-84628-885-2_4]

Salton, G., McGill, M.J., 1986. Introduction to Modern Information Retrieval. McGraw-Hill, NY, USA.

Shvaiko, P., Euzenat, J., 2008. Ten challenges for ontology matching. *LNCS*, **5332**:1164-1182. [doi:10.1007/978-3-540-88873-4_18]

van Hage, W.R., Sini, M., Finch, L., Kolb, H., Schreiber, G., 2010. The OAEI food task: an analysis of a thesaurus alignment task. *Appl. Ontol.*, **5**(1):1-28. [doi:10.3233/AO-2010-0072]

Vernica, R., Carey, M., Li, C., 2010. Efficient Parallel Set-Similarity Joins Using MapReduce. Proc. Int. Conf. on Management of Data, p.495-506. [doi:10.1145/1807167.1807222]

Wang, P., Zhou, Y.M., Xu, B.W., 2011. Matching Large Ontologies Based on Reduction Anchors. Proc. 22nd Int. Joint Conf. on Artificial Intelligence, p.2343-2348. [doi:10.5591/978-1-57735-516-8/IJCAI11-390]

Watters, C., 1999. Information retrieval and the virtual document. *J. Am. Soc. Inform. Sci.*, **50**(11):1028-1029. [doi:10.1002/(SICI)1097-4571(1999)50:11<1028::AID-ASI8>3.0.CO;2-0]

Zhang, H., Hu, W., Qu, Y.Z., 2011. Constructing virtual documents for ontology matching using MapReduce. *LNCS*, **7185**:48-63.