Constructing Virtual Documents for Ontology Matching Using MapReduce

Hang Zhang, Wei Hu, and Yuzhong Qu

State Key Laboratory for Novel Software Technology, Nanjing University, China
hangzhang@smail.nju.edu.cn, {whu,yzqu}@nju.edu.cn

Abstract. Ontology matching is a crucial task for data integration and management on the Semantic Web. The ontology matching techniques today can solve many problems from heterogeneity of ontologies to some extent. However, for matching large ontologies, most ontology matchers take too long run time and have strong requirements on running environment. Based on the MapReduce framework and the virtual document technique, in this paper, we propose a 3-stage MapReduce-based approach called V-Doc+ for matching large ontologies, which significantly reduces the run time while keeping good precision and recall. Firstly, we establish four MapReduce processes to construct virtual document for each entity (class, property or instance), which consist of a simple process for the descriptions of entities, an iterative process for the descriptions of blank nodes and two processes for exchanging the descriptions with neighbors. Then, we use a word-weight-based partition method to calculate similarities between entities in the corresponding reducers. We report our results from two experiments on an OAEI dataset and a dataset from the biology domain. Its performance is assessed by comparing with existing ontology matchers. Additionally, we show how run time is reduced with increasing the size of cluster.

1 Introduction

The Semantic Web is an ongoing effort by the W3C community. To push traditional knowledge towards a common expression form, a number of data producers, such as MusicBrainz [12] and FMA [18], have published their data in the form of ontologies.

The wildly use of ontologies brings a practical problem. Due to the dispersion of Web data, there are multiple ontologies from different publishers over the world. Therefore, in the same or related domain, different ontologies may contain heterogeneous classes, properties and instances (all of them are uniformly called entities in this paper), which need ontology matching techniques to find those denoting the same thing in the real world [6,1].

To date, a number of ontology matching tools have been created to solve the problem of heterogeneity. Referring to the report of OAEI 2010 [4], some ontology matchers have good performance on real world datasets. However, a complex matching algorithm often leads to a long run time. According to our

J.Z. Pan et al. (Eds.): JIST 2011, LNCS 7185, pp. 48–63, 2012.

investigation, most ontology matching tools suffer from unsatisfiable run time in large ontology matching despite of their high precision and recall. For example, in medicine and biology domains, two large ontologies (FMA [18,23] and GALEN[1]) need to be matched. But most matchers spend hours even weeks on matching them. The main reason is their complex matching algorithms with limited CPU and memory environments [17]. Some researchers focused on the solution, such as ontology partition [8] and early pruning of dissimilar element pairs [15]. But they all fail to utilize the great power of modern parallel computing devices.

In this paper, we propose a parallel matching approach called V-Doc+ which is based on virtual document [16] and MapReduce [2]. The architecture of our approach is outlined in Fig. 1.

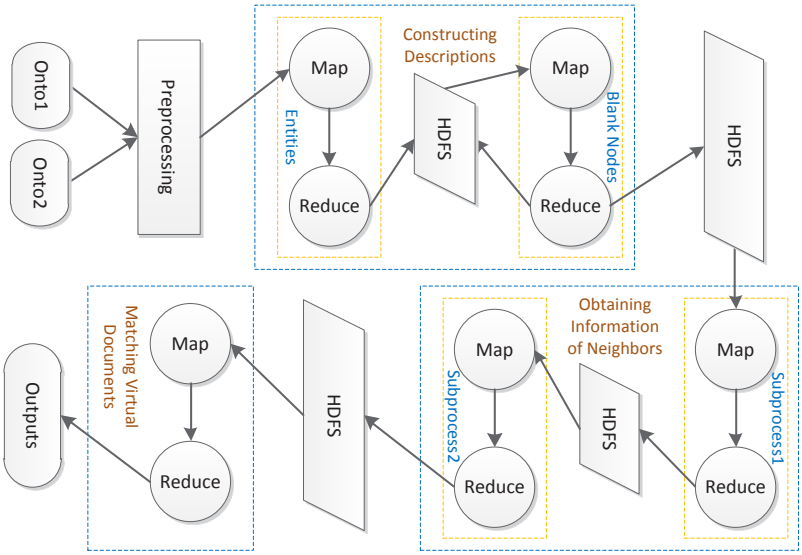

Fig. 1. Overview of the approach

Our approach contains three end-to-end MapReduce stages: constructing descriptions, obtaining information of neighbors and matching virtual documents. Before all start, in the stage of preprocessing, ontologies are splitted into the files which fit the requirement of the input format of MapReduce [2]. Based on the separated entities and RDF statements, the descriptions of entities and blank nodes will be calculated using several iterative MapReduce processes. Also, we optimize the description of each entity and blank node by annotating them with the descriptions of the neighbors. Finally, in the stage of matching virtual documents, we extract the high-weight words in the descriptions and partition the entities for reducing calculation space.

[1] `http://www.opengalen.org/`

We test V-Doc+ on the Food Ontology from OAEI 2007 and FMA vs. GALEN in medicine and biology domains. The experimental result shows a good run time of our approach with moderate precision and recall. The comparison of the efficiency in the environment with different number of nodes is also specified in the paper.

The rest of this paper is organized as follows. Sect. 2 presents the foundation of our approach and introduces the problem and Sect. 3 discusses related works. In Sect. 4, we give a MapReduce-based approach to construct the descriptions of entities and blank nodes. Information of neighbors is utilized in Sect. 5. Based on the virtual documents, similarities are calculated and the method is introduced in Sect. 6. Experimental results on two datasets are shown in Sect. 7. Finally, Sect. 8 concludes this paper with future work.

2 Preliminaries

In this work, all stages are established on the MapReduce framework. In the remainder of this section, we give the problem statement and introduce MapReduce briefly.

2.1 Problem Statement

There are a number of works that present different viewpoints on the ontology matching problem. In this paper, we define ontology matching as the process of finding mappings between entities from different ontologies. Each mapping consists of two entities and their confidence value. The definition is given as follows:

Definition 1 (Ontology Matching). *Let $\boldsymbol{O}$ and $\boldsymbol{O'}$ be two ontologies. The objective of ontology matching is to find a set of mappings defined as follows:*

$$\mathcal{M} = \{\boldsymbol{O}, \boldsymbol{O'}, \boldsymbol{M}\} \tag{1}$$

where $\boldsymbol{M} = \{m_1, m_2, ..., m_n\}$ denotes a set of mappings. A mapping m_i can be written as $m_i = (e, e', sim)$ where e and e' are two entities from $\boldsymbol{O}$ and $\boldsymbol{O'}$ respectively, and $sim \in (0, 1]$ denotes the similarity between e and e'.

2.2 MapReduce

MapReduce is the most popular framework for parallel computation on a number of computing nodes [2]. In MapReduce, the unit of computing process is named job, where each job consists of two main phases: map and reduce. The map inputs the data from the sources and splits each record into key/value pairs. These pairs are partitioned into different reducers according to the keys. Before a reducer handles the data, partitioned pairs are sorted by their keys and all values sharing the same key are clustered into the same set. The computation process is expressed in Fig. 2.

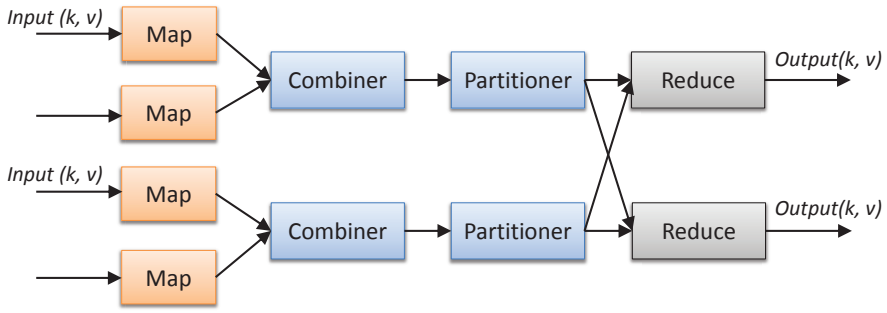

Fig. 2. Data flow in MapReduce

MapReduce also provides programmers with an extensible framework. Based on the interfaces, programmers are allowed to assign rules on how to partition key/value pairs and how to sort by keys, and these partitioning rules offers benefits for balancing the workload. Additionally, a combiner can also be rewritten to perform a local reducer to relieve the workload of the reducers.

MapReduce has the great power of parallel computing which makes used widely in a number of fields. In the ontology matching problem, one process which costs much time is to match every two entities. For some similarity-based matchers, calculating similarity on a large-scale dataset will cost too much time. As an example, calculating similarity for every pair of a n-size set of entities must be repeated $\frac{n*(n-1)}{2}$ times. Fortunately, MapReduce provides a parallel approach to partition data. According to the customized rules of partitioning, the n-size set can be partitioned into several subsets and different subset is calculated in different computing node. Thus, the run time is largely reduced.

Another usage of MapReduce for ontology matching is to solve the set-join problem. For example, for class c, finding all $rdfs{:}comment$ values of c need to find all RDF statements that satisfy $(c, rdfs{:}comment, l)$ where l is a literal. Using MapReduce process can join c with its related RDF statements in a specific reducer.

3 Related Work

Although plenty of works have been proposed for the ontology matching problem, few approaches focus on matching large ontologies. In fact, some simple matching algorithms can deal with large ontologies with a good run time, such as edit distance [19]. However, the good efficiency of them mostly relies on the lightweight algorithms, which somtimes cannot achieve high precision and recall.

Rahm [17] summarized works towards large-scale schema and ontology matching. One solution is to reduce search space. Rewriting matching processes [15] could deal with different types of matching processes and use filter operators to

prune dissimilar element pairs. The work in [8] integrated a structure-based partitioning algorithm into Falcon-AO. This divide-and-conquer algorithm could calculate anchors and partition a large ontology into small clusters. Different from Falcon-AO, the work in [22] used the positive and negative reduction anchors but did not partition ontologies. However, these works still do not improved efficiency enough for a single compute node. Simplifying original algorithms to solve the ontology matching problem is also an option [11], but may not obtain a good recall.

There exists an approach that computes set-similarity joins with MapReduce [20]. This approach proposed both self-join and R-S join cases, and partitioned data in order to reduce matching space and balance the workload. The experiments showed a surprising run time and a good speedup on large-scale datasets. But the approach cannot be directly applied to matching ontologies as there are blank nodes existing.

For matching large ontologies, some researchers investigated and compared two kinds of parallelization on matching ontologies [7], and intra-matcher parallelization has been proved more versatile. The experiment also showed a feasibility of parallel matching. However, this kind of approaches is in essence a parallelizing matching workflow service consisting of a job queue, which is not easy to implement.

4 Constructing Descriptions

The construction of descriptions which are described by a collection of words with weights consists of two MapReduce sub-stages. We first preprocess to transform ontologies into records which can be sent to mappers directly. To minimize the network traffic, in all MapReduce processes, the real URIs are not transferred. So each URI in the RDF statements is replaced with an identifier based on unique name assumption. For example, we identify a class using a token c_i, a property using a token p_i and an RDF statement using s_i while w_i denotes a external URI involved.

In this section, we focus on how to construct descriptions in the MapReduce framework.

4.1 Descriptions of Entities

The first sub-stage is for named entities whose descriptions can be obtained by the local information. For an entity e, the description is defined as follows:

$$\begin{aligned} Desc(e) = {} & \alpha_1 * \text{collection of words in the local name of } e \\ & + \alpha_2 * \text{collection of words in the rdfs:label of } e \\ & + \alpha_3 * \text{collection of words in the rdfs:comment of } e \\ & + \alpha_4 * \text{collection of words in other annotations of } e \end{aligned} \tag{2}$$

where α_1,α_2,α_3 and α_4 are fixed rational numbers in [0,1].

Fig. 3 presents an example data flow of constructing descriptions for entities. In the process, we ignore blank nodes. All the information of $rdfs{:}comment$ and other annotations come from RDF statements. So the records from input involve classes, properties and RDF statements.

The map function extracts the identifiers to check if they are RDF statements. For each record of statement, the map function replaces its identifier with its subject. But for classes and properties, the keys are directly emitted. For example, in the figure, a record of RDF statement $(s1, (c2, p1, l1))$ is emitted by changing the key $s1$ to $c2$. However, a record of class $(c1, class1)$ 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. The reduce function then calculates $Desc(c)$ or $Desc(p)$ according to Equation (2).

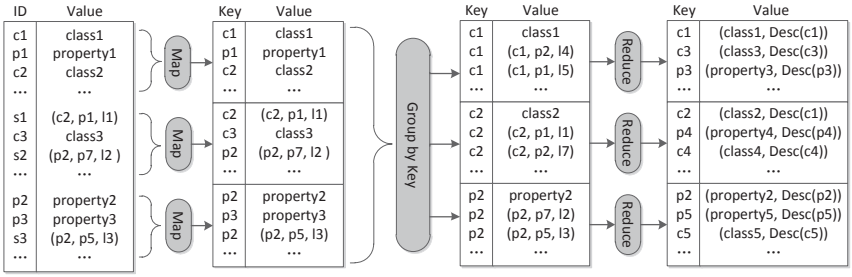

Fig. 3. Example data flow of constructing descriptions for entities

4.2 Descriptions of Blank Nodes

The second sub-stage is constructing descriptions for blank nodes. Having no local description, blank nodes get their information from neighbors, which may involve an iterative process. The following iteration equations give a convergence solution:

$$Desc_1(b) = \sum_{subj(s)=b} Desc(pred(s)) + \sum_{\substack{subj(s)=b \\ obj(s)\notin B}} Desc(obj(s)) \tag{3}$$

$$Desc_{k+1}(b) = \beta * (Desc_1(b) + \sum_{\substack{subj(s)=b \\ obj(s)\in B}} Desc_k(obj(s))) \tag{4}$$

where $subj(s)$, $pred(s)$ and $obj(s)$ denote subject, predicate and object of an RDF statement respectively. β is an attenuation coefficient in the [0,1) range.

However, MapReduce is not designed to handle the recursive problem. So we transform each step of above equations into a MapReduce process. Firstly, we establish a blank node structure to implement WritableComparable interface, which extends the simple transmission unit for carrying the information of remaining nodes. Thus, the record of blank node b is extended from $Desc(b)$ to $(Desc(b), \{neb_1, neb_2, ..., neb_n\})$, where neb_i denotes the remaining node waiting to be calculated.

We build a k-times-repeated MapReduce process to calculate descriptions for blank nodes. Before map function starts, every remaining nodes set is initialized with the current blank node. The input of map function derives from the output of the first sub-stage. Fig. 4 shows an example data flow of constructing descriptions for blank nodes. For each neb_i in $(Desc(b), \{neb_1, neb_2, ..., neb_n\})$, a map function generates a new key-value pair where the key is neb_i and the value is $(Desc(b), \{\})$. Thus, a record of blank node may be replicated as many times as the number of remaining nodes. The treatment of RDF statements is similar with that in the first sub-stage.

The reduce function aggregates a blank node b with the related entities or RDF statements. For each related entity e, we update the description of blank node with $\beta^k * Desc(e)$ where k denotes the number of times the process has be repeated. For statement s whose subject is b, we add the object of s to the remaining nodes of b. For example, in Fig. 4, the record of blank node $b1$ has two remaining nodes $\{b1, c5\}$ and is thus partitioned to the reducers with $(b1, p6, c1)$ and $Desc(c5)$ respectively. In the reducer which loads $(b1, p6, c1)$, $p6$ and $c1$ is added to the remaining nodes of $b1$. Meanwhile, $Desc(b1)$ is updated by $\beta^k * Desc(c5)$ in another reducer. Notice that there exist duplicated records in the output. So, a combinator is needed to integrate results.

It also should be noticed that we do not consider the cycle descriptions in this process. We store the route of nodes which 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 experiments, five times of iteration is usually enough to converge.

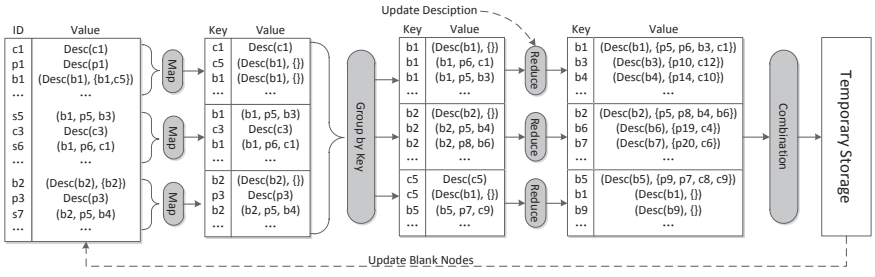

Fig. 4. Example data flow of construction of descriptions for blank nodes

5 Exchanging Information with Neighbors

The construction of virtual document needs both local descriptions and neighbor information. This section considers to use information of neighbors to update the description of each entity for constructing virtual document. The following two equations give the definition of virtual document:

$$VD(e) = Desc(e) + \gamma * Neigh(e) \tag{5}$$

$$Neigh(e) = \sum_{e' \in SN(e)} Desc(e') + \sum_{e' \in PN(e)} Desc(e') + \sum_{e' \in ON(e)} Desc(e') \tag{6}$$

where $SN(e)$ denotes the set of predicates and objects in the RDF statements whose subject is e, $PN(e)$ denotes the set of subjects and objects in the RDF statements whose predicate is e and $ON(e)$ stands for the set of subjects and predicates in the RDF statements whose object is e. We define γ as the repeat times of the MapReduce process for blank nodes and let $\gamma = 0.1$. But for some cases that most local information consist of trivial serial numbers or other random tokens, γ should be increased.

The calculation process contains two stages. The first stage is to notice each node with its neighbors and the second stage exchanges the descriptions between the neighbors.

Fig. 5 shows an example data flow. For each RDF statement (s, p, o), the map function of the first stage generates three new key-value pairs: $(s, \{p, o\})$, $(p, \{s, o\})$ and $(o, \{s, p\})$. After that, for every entity e and blank node b, all neighbors can be obtained in the reduce function. For example, an RDF statement $(b1, p5, b3)$ is a record from input, after mapping, $(b1, \{p5, b3\})$, $(p5, \{b1, b3\})$ and $(b3, \{b1, p5\})$ are partitioned into three different reducers. In the reducer loading $(b1, \{p5, b3\})$, $b1$ gets to know that there exist neighbors $p5$ and $b3$ and adds them in a temporary structure. The outputs of reducers are stored in a temporary storage waiting the map function of the second stage to read.

With the locations of neighbors, every node sends its description in the second stage. For the output value $(Desc(c1), \{b2, p4, p1\})$ in the temporary storage, $c1$ sends its description $Desc(c1)$ to $b2$, $p4$ and $p1$. Thus, in the reduce function, every entity gets all the descriptions of the neighbors and updates its own description.

Because of frequency skew, some reducers may meet an unbalanced workload so that the whole calculation process is delayed heavily. Consequently, we count the appearing time of each entity in all RDF statements and find those with the highest frequency. Then we arrange some specific reducers to calculate the entities with the top-frequency.

6 Matching Virtual Documents

In the final stage, we calculate the similarities between virtual documents with the TF/IDF technique [14]. The value of TF can be easily calculated for a specific

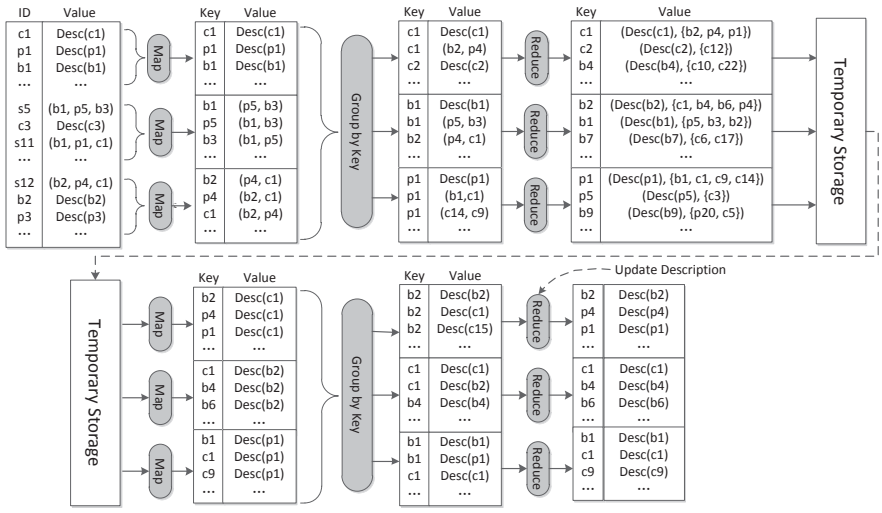

Fig. 5. Example data flow of exchanging information with neighbors

virtual document. To obtain the value of IDF, we build an additional MapReduce process for calculating the frequency of each word, and the number of virtual documents can be obtained in the preprocessing by counting the number of entities. Cosine similarity is used to measure the similarity. Equation (7) gives the function:

$$sim(e_1, e_2) = \frac{VD(e_1) \times VD(e_2)}{|VD(e_1)| * |VD(e_2)|} \tag{7}$$

Given a threshold θ, we define (e_1, e_2) be an ontology matching alignment where $sim(e_1, e_2) > \theta$.

Consider the objective of ontology matching, in the stage of matching virtual documents, we ignore the instances. However, if we calculate similarity for every two virtual documents respectively, lots of time would be wasted. So reducing the calculation space is necessary. In this stage, we propose a word-partition-based method to filter unnecessary matchings. For each description $\{(word_1, score_1),(word_2, score_2),...,(word_n, score_n)\}$, we normalize *scores* in $[0, 1]$ and rank $(word, score)$ pairs according to the value of *scores*. Thus, $score_1 \geq score_2 \geq ... \geq score_n$. We define the important words as the set of words $\{word_1, word_2, ..., word_i\}$, where i is the minimal integer which satisfy $score_1 + score_2 + ... + score_i \geq \delta$. δ is a fixed rational number in $[0, 1]$.

Fig. 6 explains the process. Each mapper ranks the words and put the top words into the keys. For example, we select three words $word1, word2, word3$ for entity $c1$ in the map phase and generate new key-value pairs: $(word_1, Desc(c1))$, $(word_2, Desc(c1))$ and $(word_3, Desc(c1))$. After partitioning, descriptions of entities $c1$, $c8$ and $c5$ group into the same reduer with key $word1$. Then we use Equation (7) to calculate the similarity between each two of them.

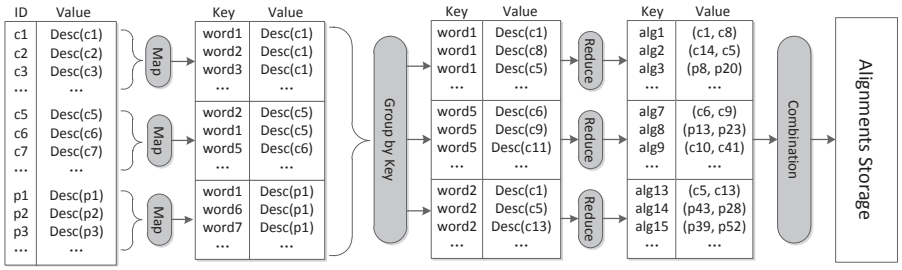

Fig. 6. Example data flow of matching virtual documents

Workload Balance. We load the frequency of words in the memory and construct a customized partitioner to choose the corresponding reducer. The words should be distributed in each node according to the frequency as average as possible. But for those with too high frequency, it is very hard to arrange or even split them. In this case, we assign one or more reducers to compute these high-frequency words while ignoring other keys. But we also allow them to choose reducers randomly if the number of computing nodes is too small.

7 Evaluation

We developed a parallel computing system, called V-Doc+, for our approach. V-Doc+ is based on the Hadoop framework[2], which provides an open-source software for scalable and distributed computing. Every mechanism of MapReduce corresponds to a process in Hadoop implementation. Given rich libraries, programmers are allowed to implement customized data structure, input/output record format, map/reduce function, and the way to partition.

In our program, each stage discussed above was implemented in one or more map/reduce functions. Particularly, some supporting functions, such as word statistics and combination, were added to the proper places in the whole implementation.

We ran our experiments on 10-node cluser and a Gigabit Ethernet interconnect. The NameNode equipped with an Intel processor with six 2.80GHz cores and 12M cache while the storage has 32GB memory and 2TB hard disk. For JobTracker and slave, the CPU is all Intel Quad Core and 2.4GHz/12M cache. The storage of JobTracker and slave is a little smaller than NameNode, which has 24GB memory and 2T hard disk. For compatibility consideration, we installed 0.21.0-version of Hadoop which is based on JDK v1.6.0 on Redhat Enterpreise Linux Server 6.0 system. All stage of MapReduce process can be monitored in a Web browser.

According to experiments with varied parameters and the optimal result generated, we configured the parameters as follows: for calculating description stage, we set $\alpha_1 = 1$, $\alpha_2 = 0.5$, $\alpha_3 = 0.25$, $\alpha_4 = 0$ and $\beta = 0.5$. For exchanging information of neighbors stage and matching virtual documents stage, $\gamma = 0.1$ and

[2] `http://hadoop.apache.org/`

$\delta = 0.75$ respectively. In practice, α should be configured differently due to the fact of datasets. Particularly, if there is no differentiation between local names, α_1 should be lower or even be 0.

Preparation. Before starting experiments, some preprocessing were built. Firstly, we cached stopwords and entities/words frequency statistics in the memory. Also, we formatted the file system before each experiment starting and made sure that there was no other programs running on every computing node. For distinguishing the source of the record easily, the identifers of entities, blank nodes and RDF statements were attached with the ontology name.

7.1 Datasets

According to our investigation, we chose two datasets according to their sizes: the Food Ontology in OAEI 2007 and FMA vs. GALEN. The reasons are as follows:

1. Our approach is designed to match large ontologies. Due to the network cost and repeated MapReduce job initialization, for small ontologies, it may perform worse than other tools. So, large ontologies are more suitable.
2. The last Food Ontology version and the results of the participants were published in OAEI 2007. So we cannot obtain this dataset after OAEI 2007.
3. Although OAEI publishes the campaign results of all tracks, such as Anatomy whose size is also suitable, we find no reference mappings for other size-suitable datasets so that we cannot evaluate their precisions and recalls.

Table 1 shows the statistical data of the Food Ontology which has lots of multi-lingual texts. It contains two ontologies: NALT and AGROVOC. NALT is developed by United Nations Food Organization and AGROVOC comes from Agriculture Organization. After the end of OAEI 2007, the results of the participants and the gold standard used to evaluate precision and recall are published on the OAEI website[3]. We downloaded them and used the gold standard to calculate precision, recall and value of F1 method in our experiment.

Table 1. Statistical data of the Food Ontology

Ontology	Classes and Properties	Statements
NALT	42,326	174,278
AGROVOC	28,439	319,662

Table 2. Statistical data of FMA vs. GALEN

Ontology	Classes and Properties	Statements
FMA	72,659	576,389
GALEN	9,596	59,753

For another dataset, we matched FMA ontology and GALEN ontology. FMA is more larger than GALEN. Unfortunately, we cannot find other ontology matching tools publishing their results of FMA vs. GALEN matching. Also,

[3] `http://oaei.ontologymatching.org/2007/`

no gold standard was found. Consequently, for FMA vs. GALEN, we only ran our experiment program on it and showed the run time and the speedup for different numbers of compute nodes. The statistical data of FMA vs. GALEN is showed in Table 2.

7.2 Experimental Results

The goal of our evaluation is threefold. Firstly, it is necessary to test precision, recall and F-Measure although the strength of V-Doc+ is its good efficiency. To test it, we investigated several ontology matchers and compared V-Doc+ with their performance. Secondly, we want to show how much the run time was reduced comparing with the non-parallel matchers. Thirdly, we showed the speedup on different computing nodes environment and presented the run time for each stage in our approach.

We evaluated V-Doc+ on precision, recall and F-Measure on the Food Ontology from OAEI 2007 [5] with some matchers: Falcon-AO, DSSim, RiMOM, Prior+ and COMA. One is Falcon-AO [8], which integrated three matchers: I-Sub, V-Doc and GMO, where V-Doc is a virtual-document-based technique which runs in a non-parallel way and GMO is a graph matching technique based on structural similarity. To match large ontologies with limited memory, Falcon-AO also constructed a divide-and-conquer approach which can partition entities into small clusters. DSSim [21] gave a multi-agent system to solve the ontology matching problem while considering uncertainty. The main approach of DSSim was to use different domains for finding ontology mappings. RiMOM [10] integrated multiple matchers to improve effectiveness by combining both literal and structure features. Another matcher Prior+ [13] is an adaptive ontology matching tool which was based on several different techniques, such as IR-based similarity and neural network. Like Falcon-AO, COMA [3] also considered the block matching and provided a fragment-based matcher to solve the large ontology matching problem. Differently, it partitioned data represented as trees.

F-Measure. We gave the performance of our experiment on the Food Ontology using F-Measure to assess the results. Firstly, we calculated precision and recall according to the gold standard that OAEI provides. Then, the value of F-Measure was calculated using the following equation:

$$F\text{-}Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

Fig. 7 shows the result of comparison between V-Doc+ and other five matchers on the Food Ontology. From the figure, we observe that the precision, recall and f1-measure of V-Doc+ is no better than some of matchers. It mostly dues to the combination of varied algorithms for other tools while one in V-Doc+. However, they are all lower than Falcon-AO's. The main reason is that Falcon-AO combined several matchers, including I-Sub, V-Doc and GMO, where V-Doc is a non-parallel implementation of the virtual document technique which has the similar precision and recall with V-Doc+.

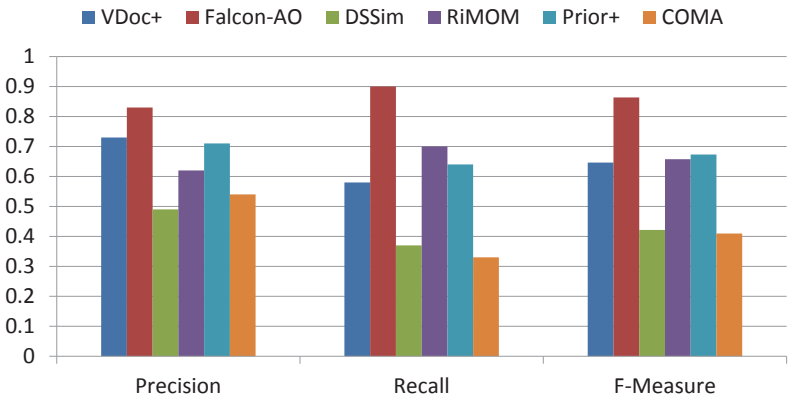

Fig. 7. Precision, recall and F1-Measure on the Food Ontology

Run Time. The strength of our approach is its efficiency. Table 3 shows the run times for 10-node cluster on the Food Ontology. V-Doc+ only spent ten minutes on running. Among other matchers, the one cost the least is Prior+ which spent 1.5 hours while DSSim cost one week which is the slowest. Consequently, although V-Doc+ does not achieve the best F-Measure, the parallelization makes it much faster than others.

Table 3. Run times comparison among V-Doc+, Falcon-AO, DSSim, RiMOM and Prior+ on the Food Ontology

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

To analyze each stage of the whole approach, we calculated the run time for details presented in Fig. 8, where 10-node cluster is used. For both the Food Ontology and FMA vs. GALEN, constructing descriptions and calculating similarities spent the most of time.

In order to evaluate the speedup, we calculated the run time for varied cluster sizes. For each dataset, we ran our program on 2, 4, 8, 10 nodes environment. Fig. 9 shows the result. In the figure, we see that the run time keeps reducing while increasing the number of compute nodes. But we also notice that the growth trend of efficiency is reducing. Fig. 10 shows the speedup which also reflects the reducing growth of efficiency with increased cluster size. From 8-node to 10-node, the speedup tends to be unchanged. An interesting thing is that the speedup on the Food Ontology is smaller than that on FMA vs. GALEN. The main reason is that, for any dataset, our approach must repeat the MapReduce process several times on constructing the descriptions for blank nodes. Although other MapReduce process stages on the Food Ontology cost less time than that on FMA vs. GALEN, they share the similar run time on the blank node stage, which leads to a low speedup.

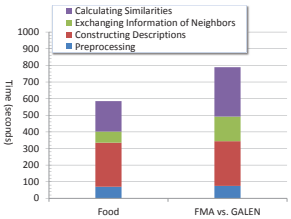

Fig. 8. Run time of each component on a 10-node cluster

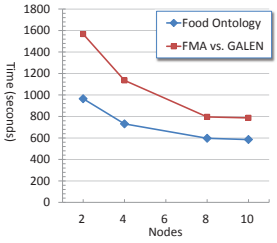

Fig. 9. Run time on different cluster sizes

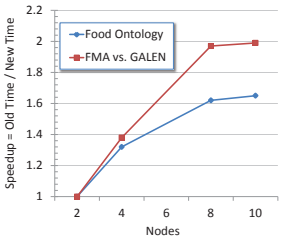

Fig. 10. Speedup on different cluster sizes

Some small scale datasets (less than 1,000 classes and properties and less than 3,000 statements) have also been tested for the run time and the speedup, but the result is not so good. Due to the network cost and the repeated reading of files, the whole process cost as much time as other tools.

8 Conclusion

Matching large ontologies is an inevitable obstacle for data fusion and only a few ontology matchers can finish matching task in a satisfiable run time. MapReduce is a wildly used computing framework for parallel computation, and it has been used in a number of fields. However, there is few studies on matching ontologies using MapReduce. In this paper, we proposed a 3-stage parallelized ontology matching method, called V-Doc+, using virtual document technique based on the MapReduce framework, which largely reduces the run time from hours to minutes.

Each stage of our approach establishes several MapReduce processes. For blank nodes, the descriptions are updated iteratively by emitting neighbors to the reducers. For similarities, we calculate the high-weight words in the descriptions of entities by ranking the frequency and emit the entities to reducers. Also, we considered the workload balance. The frequency of entities and words in RDF statements is calculated in the preprocessing. Then the statistical data of frequency is loaded in the memory to assist automatically partition.

For performance test, we conducted experiment on two large real datasets and the results showed a good efficiency and moderate precision and recall. For the Food Ontology from OAEI 2007, V-Doc+ used ten minutes to finish the task while other tools spent hours even weeks. The speedup with increased computing nodes is also illustrated.

Currently, a number of matchers obtain good precision and recall by combining multiple matchers. However, our approach is restricted to a certain linguistic matching algorithm. In the future work, we look forward to integrating a new parallelized algorithm based on ontology structures to improve precision and recall.

Acknowledgements. This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 61003018 and 61021062, in part by the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 20100091120041, and also in part by the Natural Science Foundation of Jiangsu Province under Grant No. BK2011189. We appreciate Jianfeng Chen for conduction on MapReduce programming.

References

1. Bleiholder, J., Naumann, F.: Data Fusion. ACM Computing Surveys 41(1), 1–41 (2008)
2. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM 51(1), 107–113 (2008)
3. Do, H., Rahm, E.: Matching Large Schemas: Approaches and Evaluation. Information Systems 32(6), 857–885 (2007)
4. Euzenat, J., Ferrara, A., Meilicke, C., Nikolov, A., Pane, J., Scharffe, F., Shvaiko, P., Stuckenschmidt, H., Šváb-Zamazal, O., Svátek, V., Trojahn, C.: Results of the Ontology Alignment Evaluation Initiative 2010. In: ISWC Workshop on Ontology Matching (2010)
5. Euzenat, J., Isaac, A., Meilicke, C., Shvaiko, P., Stuckenschmidt, H., Šváb, O., Svátek, V., Hage, W., Yatskevich, M.: First Results of the Ontology Alignment Evaluation Initiative 2007. In: ISWC Workshop on Ontology Matching (2007)
6. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
7. Gross, A., Hartung, M., Kirsten, T., Rahm, E.: On Matching Large Life Science Ontologies in Parallel. In: Lambrix, P., Kemp, G. (eds.) DILS 2010. LNCS, vol. 6254, pp. 35–49. Springer, Heidelberg (2010)
8. Hu, W., Qu, Y., Cheng, G.: Matching Large Ontologies: A Divide-and-Conquer Approach. Data & Knowledge Engineering, 140–160 (2008)
9. Jean-Mary, Y., Shironoshita, E., Kabuka, M.: Ontology Matching with Semantic Verification. Journal of Web Semantics 7(3), 235–251 (2009)
10. Li, J., Tang, J., Li, Y., Luo, Q.: RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. IEEE Transactions on Knowledge and Data Engineering 21(8), 1218–1232 (2009)
11. Mork, P., Bernstein, P.: Adapting a Generic Match Algorithm to Align Ontologies of Human Anatomy. In: Proceedings of the 20th International Conference on Data Engineering, pp. 787–790 (2004)
12. Moutselakis, E., Karakos, A.: Semantic Web Multimedia Metadata Retrieval: A Music Approach. In: 13th Panhellenic Conference on Informatics, pp. 43–47 (2009)
13. Mao, M., Peng, Y., Spring, M.: An Adaptive Ontology Mapping Approach with Neural Network Based Constraint Satisfaction. Web Semantics: Science. Services and Agents on the World Wide Web 8(1), 14–25 (2010)
14. McGill, M., Salton, G.: Introduction to Modern Information Retrieval. McGraw-Hill (1983)
15. Peukert, E., Berthold, H., Rahm, E.: Rewrite Techniques for Performance Optimization of Schema Matching Processes. In: Proceedings of 13th International Conference on Extending Database Technology, pp. 453–464. ACM Press, New York (2010)
16. Qu, Y., Hu, W., Cheng, G.: Constructing Virtual Documents for Ontology Matching. In: 15th International World Wide Web Conference, pp. 23–31. ACM Press, New York (2006)

17. Rahm, E.: Towards Large-Scale Schema and Ontology Matching. Data-Centric Systems and Applications, Part I, 3–27 (2011)
18. Rosse, C., Mejino, L.: The Foundational Model of Anatomy Ontology. In: Burger, A., Davidson, D., Baldock, R. (eds.) Anatomy Ontologies for Bioinformatics: Principles and Practice, vol. 6, Part I, pp. 59–117. Springer, Heidelberg (2008)
19. Stoilos, G., Stamou, G., Kollias, S.: A String Metric for Ontology Alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 624–637. Springer, Heidelberg (2005)
20. Vernica, R., Carey, M., Li, D.: Efficient Parallel Set-Similarity Joins Using MapReduce. In: SIGMOD 2010 Proceedings of the 2010 International Conference on Management of Data, pp. 495–506. ACM Press, New York (2010)
21. Vargas-Vera, M., Nagy, M.: Towards Intelligent Ontology Alignment Systems for Question Answering: Challenges and Roadblocks. Journal of Emerging Technologies in Web Intelligence 2(3), 244–257 (2010)
22. Wang, P., Zhou, Y., Xu, B.: Matching Large Ontologies Based on Reduction Anchors. In: Proceedings of International Joint Conferences on Artificial Intelligence, pp. 2343–2348 (2011)
23. Zhang, S., Bodenreider, O.: Hybrid Alignment Strategy for Anatomical Ontologies: Results of the 2007 Ontology Alignment Contest. In: ISWC Workshop on Ontology Matching (2007)