Investigating Distributed Approaches to Efficiently Extract Textual Evidences for Biomedical Ontologies

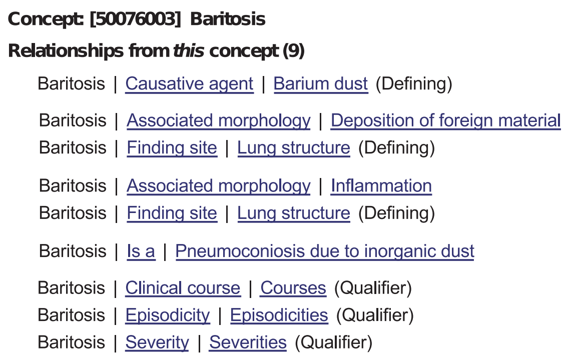
Long Cheng[1] and Yue Ma[1,2]
[1]Department of Computer Science, Technische Universität Dresden, Germany
[2] LRI – CNRS & Université Paris Sud, Orsay, France
{long.cheng, mayue}@tu-dresden.de

***Abstract*—Heterogeneous data resources in biomedicine become available both in structured and unstructured formats, such as scientific publications, healthcare guidelines, controlled vocabularies, and formal ontologies. Bridging the gaps among these heterogeneous data is useful to discover implicit knowledge. To make this happen, efficient computational approaches are a necessity for applications in such a knowledge- and data-intensive domain. In this paper, we first define a particular task, *relation alignment*, which is to identify textual evidences for biomedical ontologies. Then, we investigate two parallel approaches for this task over distributed systems and present the details of their implementations. Moreover, we characterize the performance of our methods through extensive experiments, thereby allowing researchers to make a more informed choice in the presence of large-scale biomedical data.**

***Keywords*-Biomedical ontology; relation alignment; distributed computation; high performance; big data**

I. Introduction

Biomedicine is a knowledge intensive discipline, where knowledge is often encoded in multiple resources, such as controlled vocabulary thesaurus (MeSH[1]), ontologies (Gene Ontology[2], Snomed CT[3]), and more generally in the form of textual resources such as scientific publications and healthcare guidelines. These heterogeneous data often have overlaps but in different representations (formal or textual). For example, Figure 1 gives a formal representation of the concept *Baritosis* from Snomed CT, and Wikipedia also has the sentence conveying a piece of similar knowledge: "Baritosis is a benign type of pneumoconiosis, which is caused by long-term exposure to barium dust". Moreover, different resources also form a good complement of each other. For example, the knowledge in Figure 1 does not cover that in the sentence: "Being a benign condition, baritosis neither interferes with lung function nor causes symptoms other than a mild cough". By exploiting the heterogeneous big data from different resources, it is expected that we can discovery implicit meaningful knowledge. In this paper, we are interested in a special correspondence between structured (ontology) and unstructured (text) data, namely tracing *textual evidences of biomedical ontology*, that is, for instance,



Concept: [50076003] Baritosis
Relationships from *this* concept (9)

Baritosis | Causative agent | Barium dust (Defining)

Baritosis | Associated morphology | Deposition of foreign material
Baritosis | Finding site | Lung structure (Defining)

Baritosis | Associated morphology | Inflammation
Baritosis | Finding site | Lung structure (Defining)

Baritosis | Is a | Pneumoconiosis due to inorganic dust

Baritosis | Clinical course | Courses (Qualifier)
Baritosis | Episodicity | Episodicities (Qualifier)
Baritosis | Severity | Severities (Qualifier)

Figure 1. The structured knowledge about Baritosis from Snomed CT

to find the correspondence between the formal knowledge given in Figure 1 and the sentences mentioned above.

Textual evidences are interesting to be related with formal ontologies for several reasons. First, creating and extending formal ontology is an expensive process. In contrast, narrative texts, such as medical records, health news, and scientific publications, contain rich information that is very useful to augment a knowledge base [1][2][3][4]. Second, once a natural language sentence is annotated with ontological labels, search engines can answer queries by displaying succinct structured data [5] or perform advanced search according to the semantic labels (e.g. GoPubMed[4]).

The increasing biomedical resources naturally form a data-intensive environment, which makes the construction of links among different resources difficult. On one hand, the traditional manual labeling is costly thus not practical. On the other hand, computing potential links over large datasets becomes heavily time consuming. To this end, in this paper, we study and explore efficient approaches to identifying links between texts and formal ontology. This is done by computing the relation alignment in parallel over distributed systems. The essential idea is to formalize the relation alignment as *join* operations, and then apply existing distributed join frameworks to implement the alignments.

Relation Alignment. Take Figure 1 as an example. Each line

[1]http://www.ncbi.nlm.nih.gov/mesh
[2]http://www.geneontology.org/
[3]http://www.ihtsdo.org/snomed-ct/

[4]http://www.gopubmed.org/web/gopubmed/

978-1-4799-7502-0/14 $31.00 © 2014 IEEE
DOI 10.1109/BIBE.2014.45

starting with "Baritosis" is a relationship given in Snomed CT ontology, in the form of $a|r|b$, where a, b are two concept names and r is the linking relation, called a pivot relation. For any relation r, the task of *relation alignment* for r is to identify textual fragments, called *textual evidences*, that indicate a relationship having r as its pivot relation. For example, the textual fragment "Baritosis is a benign type of pneumoconiosis" given in Table I can be aligned with the relationship "Baritosis | isa | Pneumoconiosis due to inorganic dust" in Figure 1. Thanks to big textual data available in many applications, regardless of some noisy alignments, it has been confirmed by many studies [6][7][3][4][5] that the aligned textual fragments can produce meaningful linguistic features to recognize the corresponding pivot relation r.

To realize relation alignment, we perform the following preprocessing steps on textual data. We first recognize all concept names from a given textual sentence, and then label the textual fragments with biomedical concepts[5], as shown in Table I, where the integer ids are the inner simple representation of Snomed CT for concepts. Next, possible concept id pairs for different phrases are extracted from the annotated sentences, such as (50076003,40122008) and (50076003,55727002) for the annotated sentence in Table I. For each concept pair (a, b) and its corresponding phrases $s1$ and $s2$, we finally check whether there is a relationship $a|r|b$ in a given ontology with a pivot relation r. If it is the case, the textual fragment between $s1$ and $s2$ is returned as the textual evidence for the relation r, and the sentence is called aligned with r.

In the Web scenario, textual resources are increasing dramatically. Meanwhile, the creation of structured semantic data also comes with a high speed, such as the biomedical ontology portal[6], linked data[7] and Wikidata[8]. It can be time consuming to align such large amount of data within a realistic time, even if the relation alignment seems a simple operation. In fact, suppose that there are L sentences with M annotations on average and N ontological relationships, the relation alignment needs $O(LM^2N)$ matchings. However, this can be improved via distributed computation with optimized management of ontologies, the intensively visited data in relation alignment.

Contribution. In this paper, we investigate two distributed approaches with target for efficiently extracting textual evidences over biomedical ontologies. We summarize the contributions of this paper as below:

- We formulate the *relation alignment* problem as join operations and propose two parallel solutions for fast computing the relation alignment over distributed architectures.

[5]In our implementation, this is done by invoking the tool Metamap [8].
[6]bioportal.bioontology.org/
[7]linkeddata.org/
[8]www.wikidata.org/

Table I
AN EXAMPLE SENTENCE WHOSE PHRASES ARE ANNOTATED BY SNOMED CT CONCEPTS BY METAMAP, WHERE "–" MEANS NO ANNOTATION AVAILABLE FOR THE CORRESPONDING PHRASES. MULTIPLE ANNOTATIONS HAPPEN DUE TO THE OVERLAPPING SEMANTICS OF THE CONCERNED CONCEPTS.

Sentence Phrases	Ontology Annotation
Baritosis	50076003
is	–
a benign type	30807003 | 261664005
of pneumoconiosis,	40122008
which	–
is	–
caused	23981006 | 134198009
by long-term exposure	71677004 | 24932003
to barium dust.	55727002

- We present the detailed implementations of our approaches and evaluate their performance over a commodity cluster with 192 cores and datasets up to 400 million tuples.
- We characterize the performance of our implementations through extensive experiments. Meanwhile, it is highlighted that our hash-based alignment method can compute alignments over a very large dataset (100 million tuples) in one minute.

The rest of this paper is organized as follows: In Section II, we introduce our parallel alignment approaches and present their detailed implementations. In Section III we provide a quantitative evaluation of our algorithms. We report on related work in Section IV while in Section V we conclude the paper outlining plans for future work.

II. Our Approaches

In this section, we first formalize the alignment problem. Then, we introduce two efficient parallel alignment methods: the *hash-based alignments* and the *duplication-based alignments*, to efficiently improve the alignment performance in the presence of large-scale biomedical data.

A. Problem Formulation

We are given a set of *concept relationships* $\mathcal{R}$ and an *ontology annotation* $\mathcal{S}$, where $\mathcal{R}$ is in the form of $a|r|b$ for two concepts a, b and a relation r, and $\mathcal{S}$ is in the form of $a|s_i{:}s_j|b$, where a, b are the annotations and s_i as well as s_j are the extracted phrases. Based on the relation alignment problem as defined, the alignment is to find all the tuples $<R, S> \in \mathcal{R} \times \mathcal{S}$ such that $R.a = S.a$ and $R.b = S.b$. In Database lingo, $\mathcal{R}$ and $\mathcal{S}$ are two *relations*.

As a *join* can facilitate the combination of two relations based on a common key, if we consider the terms (keys) a, b in each tuple of $\mathcal{R}$ and $\mathcal{S}$ as a key entity (e.g. c), then the above alignment progress becomes to a join operation. Namely, $\mathcal{R} \bowtie \mathcal{S}$ where $R.c = S.c$. In such scenario, the process to trace the textual evidence of biomedical ontology can be computed by implementing the joins between the

Algorithm 1 Hash-based Alignments

```
    Hash-Redistribution:
 1: for each node in current system parado {
 2: Initialize $A$:array[array[tuple]]($n$),
              $B$:array[array[tuple]]($n$)
 3: for all tuples $R \in \mathcal{R}$ do
 4:     $des \leftarrow$ hash($R.a$)
 5:     $A(des).add(R)$
 6: end for
 7: for all tuples $S \in \mathcal{S}$ do
 8:     $des \leftarrow$ hash($S.a$)
 9:     $B(des).add(S)$
10: end for
11: for $i \leftarrow 0..(n-1)$ do
12:     Send $A(i)$ to $A'(i)(here.id)$,
              $B(i)$ to $B'(i)(here.id)$ at node $i$
13: end for }

    Local Alignments:
14: for each node in current system parado {
15: Initialize $T\_r$:hashmap[a,(b,r)]
16: for all received tuples $R' \in A'(here.id)$ do
17:     Add $R'$ in $T\_r$
18: end for
19: for all received tuples $S' \in B'(here.id)$ do
20:     $value \leftarrow T\_b.get(S'.a).value)$
21:     if $(value \neq null) \wedge (S'.b = value.b)$ then
22:         Output < $value.r$, $S'.s_i$, $S'.s_j$ >
23:     end if
24: end for }
```

terms in the *concept relationships* and *ontology annotation*. In the following, we mainly focus on how to devise efficient parallel algorithms to support this operation.

B. Parallel Processing

The hash-based and duplication-based join framework is widely used in various parallel join algorithms [9][10][11], therefore we can apply such two approaches to compute our alignments as well. We will explain how to compute the relation alignments in parallel from that basis. To better understand the detailed process, we consider following tuples at each relation:

$\mathcal{R}$ = { 1|r_1|2, 2|r_2|5 }
$\mathcal{S}$ = { 1|s_1:s_2|2, 2|s_2:s_4|4, 1|s_1:s_3|3, 2|s_2:s_5|5 }

As we utilize a distributed method for the input data, the data is first divided into a number of equal-size *chunks* and then assigned as input for processing on separate computation nodes. For an example two-node ($n = 2$) system, the first half tuples of each relation are assigned to the first node and others are for the second node.

Hash-based Alignments. In the hash-based framework, the parallel alignment algorithm is described in Algorithm 1. There, we divided our alignments into two phases, namely *hash-redistribution* and *local alignments*.

As shown in lines 3-10 of Algorithm 1, all tuples in $\mathcal{R}$ and $\mathcal{S}$ at each node (note that n is the number of computation nodes/cores, *here* means the id of current node and we use *parado* to mean *do in parallel*) are firstly grouped according to the hash values of one of their join keys respectively. Here, we choose the term a as the join key for both relations. Then, the grouped tuples are pushed to the corresponding remote places for local alignment. We use the synchronization at the end of the first phase to guarantee the completion of the data transfer at each node. With a hash function based on the modulo of number of computation nodes n, we have the following tuples at each node after redistribution:

node 1		*node 2*	
1|r_1|2	1|s_1:s_2|2	2|r_2|5	2|s_2:s_4|4
	1|s_1:s_3|3		2|s_2:s_5|5

Once the grouped tuples have been transferred to the appropriate remote nodes, the local alignment can commence. Line 14-24 of Algorithm 1 presents the details of this process. A local `HashMap` T_r is built based on the received tuples at first. The key is the term a and the value is the pair of b and r. Then, all received tuples of $\mathcal{S}$ are looked up over T_r to retrieve the matched values. If a value exists and also has the same term b as the looked tuples, then a responsible results will be formulated. All these processes take place in parallel at each node, and the whole alignment process terminates when all individual nodes terminate.

As we redistribute all tuples among the processors based on a hash function, the potentially matched tuples will be co-located on the same node, checking the two keys at each node using a conventional alignment method, we can easily get the final outputs. Thus, after execution, we have:

node 1 output < r_1, s_1, s_2 >
node 2 output < r_2, s_2, s_5 >

Duplication-based Alignments. The duplication-based distributed alignment method is shown in Algorithm 2. This approach can be also divided into two phases: *duplication*,

Algorithm 2 Duplication-based Alignments

```
    Duplication:
 1: for each node in current system parado {
 2: for all tuples $R \in \mathcal{R}$ do
 3:     Send $R$ to all other computation nodes
 4: end for }

    Local Alignments:
 5: for each node in current system parado {
 6: Initialize $T\_r$:hashmap[a,(b,r)]
 7: for all received tuples $R' \in \_A(here.id)$ do
 8:     Add $R'$ in $T\_r$
 9: end for
10: for all tuples $S \in \mathcal{S}$ do
11:     $value \leftarrow T\_r.get(S.a).value)$
12:     if $(value \neq null) \wedge (S.b = value.b)$ then
13:         Output < $value.r$, $S.s_i$, $\_S.s_j$ >
14:     end if
15: end for }
```

local alignments. The first phase just simply duplicates (broadcasts) the tuples of $\mathcal{R}_i$ at each node to all other nodes. This means that, after the broadcast, the composed relation $\mathcal{R}_k$ ($k \in [1, n]$) at each node will be equal to the full input $\mathcal{R}$, namely, $\mathcal{R}_k = \bigcup_{i=1}^{n} \mathcal{R}_i = \mathcal{R}$. Therefore, we have the following tuples at each node after the duplication:

node 1		*node 2*	
1|r_1|2	1|s_1:s_2|2	2|r_2|5	2|s_2:s_4|4
2|r_2|5	1|s_1:s_3|3	1|r_1|2	2|s_2:s_5|5

The following phase, as shown in lines 5-15 of Algorithm 2, is very similar to the second phase of the hash-based implementation. For example, the lookups for local tuples in $\mathcal{R}$ will commerce once the in-memory hash table of the received tuples $\mathcal{R}$ is created.

III. Experimental Evaluation

This section presents a comparative quantitative analysis of the proposed approaches. We first introduce the benchmark scenarios and then present the detailed experimental results.

A. *Benchmark Scenarios*

We use 16 IBM servers with each containing two 6-core Intel Xeon X5679 processors clocked at 2.93 GHz, 128GB of RAM and a single 1TB SATA hard-drive, connected using Gigabit Ethernet. The operating system is Linux kernel version 2.6.32-220 with gcc version 4.4.6. We implemented our approaches using the parallel language X10 [12] with version 2.3 and compiled the codes to C++.

As our extracted real datasets are relative small (because of the system limitation on the pre-processing of *ontology annotation*), to evaluate our parallel alignment approaches over large datasets, we create some synthetic datasets, while maintain some of their characteristics closed to the real data (e.g. the format). The details of our test data is given in Table II, with **bold** font indicating default values. We are interested in how efficient the presented two approaches are, and how the utilized system resources (namely computation nodes) and the alignment workloads impact their performance, therefore, we focus on the following four potential factors in our evaluations:

(1) *Number of nodes*: To evaluate the scalability of our proposed methods and check their efficiency on a distributed system, we vary the number of processing cores for each test, from 12 cores (1 node) up to 192.

(2) *Cardinality*: To see how the performance changes with increasing dataset size, similar as the evaluation works on joins [13][14], we just fix the cardinality of relation $\mathcal{R}$, to 25 million, and varying the $|\mathcal{S}|$ from 25 million to 400 million, a number which is extreme big for the available *annotation pairs*. As the skew handling is beyond the scope of this work, we just keep the data uniform distributed based on their first join key a as stated previously.

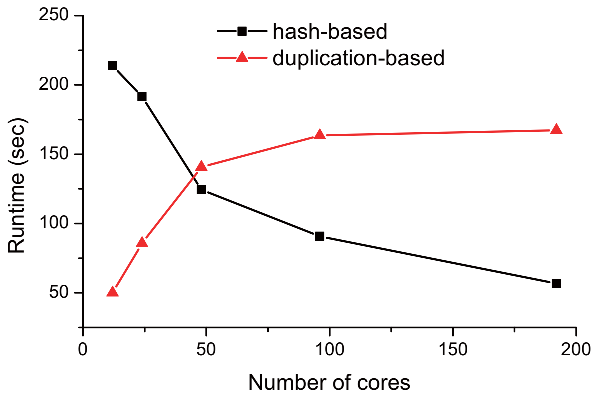


Figure 2. Runtime by varying the number of computation cores ($|\mathcal{S}|$=100M).

(3) *String pair length*: String pairs <s_i, s_j> in each tuple S of $\mathcal{S}$ are always longer than the join keys (long integers), which could result heavy network communication during the redistribution. In the meantime, transferring such strings normally needs extra serialized/deserialized processing in a distributed environment, which could bring heavy local computation. This means that the longer the string pairs are, the worse the performance could be. To check the detailed effects of this factor, we varying the responsible length from 9 bytes to 128 bytes. We set 36 bytes as the default value from the basis of our statistics over the real datasets.

(4) *Alignment rate*: We define the alignment rate as the ratio of alignment tuples in the relation $\mathcal{R}$. As the cost of local lookups over built *hashmap* for each tuple $S \in \mathcal{S}$ could be different in the cases of alignments and non-alignments (because we have to perform double key checking in our alignments), we exam this difference by varying the value of the rate in our tests, from 0 to 100%, and choose 50% as the default value for a general case.

For all the experiments, to focus on the core performance of our implementation, we only count the number of final alignments, but do not actually output results. Moreover, we record the mean value based on three measurements and we empty the file system cache between tests to minimize the effects of caching by the operating system.

B. *Experimental Results*

Scalability. We evaluate the scalability of two proposed implementations by varying the number of processing cores over the default dataset, from 12 cores (1 node) to 192, and present the results in Figure 2. It can be seen that the runtime of the hash-based method generally scales well with increasing the number of cores, showing the efficiency of the parallel implementation. Here, we highlighted that, with the hash-based method, we can compute the alignments over 100 million tuples in one minute (using 192 cores). In comparison, the time cost of the duplication approach

Table II
DETAILS OF THE TEST DATASETS

Factors	String Pair Length (Byte)	Alignment Rate	Cardinality Concept Relationships $\mathcal{R}$	Cardinality Annotation Pairs $\mathcal{S}$
Datasets	9, 18, **36**, 72, 128	0, 20%, **50%**, 80% 100%	25M	25M, 50M, **100M**, 200M, 400M

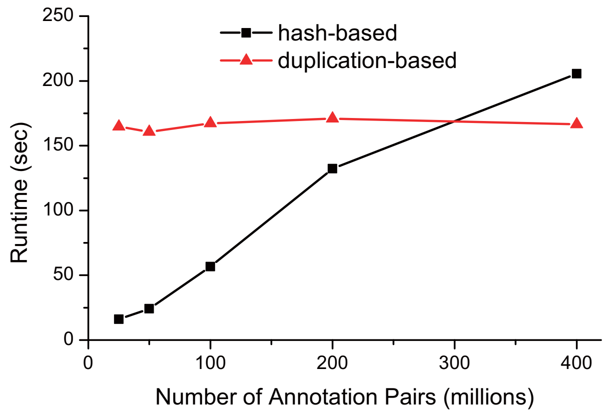


Figure 3. Runtime by varying the number of *annotation pairs* (192 cores).

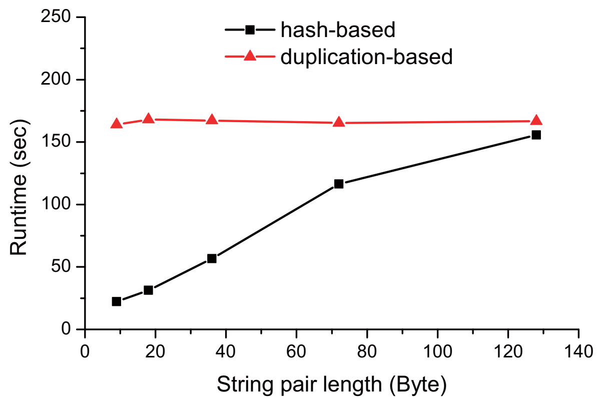


Figure 4. Runtime by varying the length of string pairs in $\mathcal{S}$ (192 cores).

increases with the number of nodes, which is out of our expectation. The possible reason is that the size of the *concept relationships* $\mathcal{R}$ is large, broadcasting such data around the network results very heavy network communication, and the responsible overhead becomes larger than the benefits from the distributed implementation.

Cardinality. The results of the *cardinality* experiments are demonstrated in Figure 3. We can see that the runtime of hash-based method increases sharply with increasing the number of annotation pairs. In comparison, the duplication-based approach is nearly constant. The possible reason is that large number of annotation pairs of $\mathcal{S}$ needs to be redistributed in the former method while the latter method only needs to broadcast the concept ontologies, the number

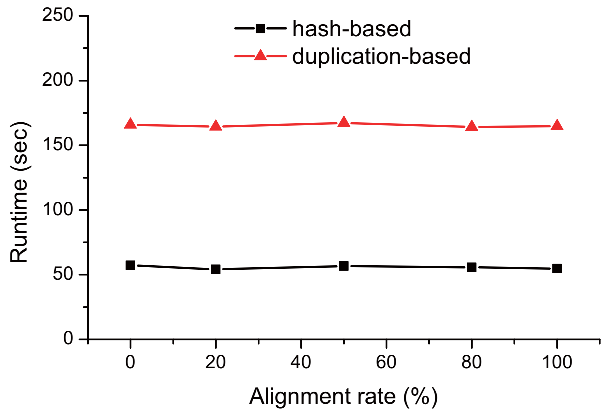


Figure 5. Runtime by varying the alignment rate as defined (192 cores).

of which is constant in our tests (namely 25M as shown in Table II). Moreover, it can be also observed that the hash-based method performs better than the duplication-based method at first, and then become worse when the number of annotation pairs reaches a vary big number. In such scenarios, the redistributed cost of $\mathcal{R}$ and $\mathcal{S}$ becomes larger then the time on broadcasting $\mathcal{R}$. This indicates that duplication-based method should be chosen for processing a very large data $\mathcal{S}$ while hash-based method should be used when processing relative small datasets.

String Length. We record the runtime of the two methods by varying the length of the string pairs. The results are presented in Figure 4. We can see that the runtime of the hash-based method is always increasing with increasing the length. This is consistent with our assumption, because the serialization/deserialization operation as well as long string transferring impacting the performance. In comparison, the duplication-based method is nearly constant. The reason is the same as the reason described above, that the broadcast tuples are fixed. Furthermore, it can be also seen that the runtime of the hash method is always smaller than the duplicated one in this test. As the extracted phrases in real datasets are normally not so long, in such scenarios, hash-based approach will be a better choice for computing the alignments.

Alignment Rate. Finally, we examine how *alignment rate* impacts the performance for each algorithm. The responsible results are demonstrated in Figure 5. There, we can see that the runtime of both algorithms nearly keep constant, and does not increase with the increment of the alignment rates, which is conflict with our previous assumption. This could be that the computation cost of alignment tasks is very light-weighted, compared with cost on network communication, because the whole time cost is composed by the time on data movements and local lookup in our implementations. The results also imply that reducing the cost on network communication will more meaningful than optimization local alignments when processing large biomedical datasets.

IV. Related Work

Relation alignment has been proposed in the framework of distant supervision. Distant supervision has proven a successful approach for information extraction in a big data environment because no supervised corpus is needed when a formal knowledge base is available with the sample relationships we want to extract. The existing efforts have been put on applying relation alignment to different tasks, including populating a knowledge base with relationships [6][7], formalizing biomedical ontologies [3][4], and discovering new attributes of given objects [5]. To the best of our knowledge, unlike this paper, there is little work on high performance computing of relation alignment. We consider it as a necessary topic as the dramatic increase of un- and structured biomedical information available on the Web. The experimental results show that our approach can significantly accelerate automatic generation of labeled data by a distributed management of biomedical ontologies.

In terms of joins, the efficient parallelisation of such operations over distributed machines becomes increasingly desirable as applications grow in scale. Up till now, the hash-based method is widely used and various distributed join algorithms have been proposed [15]. In contrast, the duplication-based approach is not so popular, but also adopted in some work [16], the performance of which is heavily rely on underlying high-speed networks. Moreover, different techniques, such as dynamic scheduling [17] and skew handling [14][18], have been applied in the implementation of distributed joins to improve the performance. We will investigate and apply such methods so as to achieve even higher performance under different alignment workloads.

V. Conclusions

In this work, we have formulated the problem of *relation alignment*, which is to identify textual evidences for biomedical ontologies. We have proposed two parallel approaches to fast computing the alignment over distributed systems. We ran experiments over different data sizes and characterized the performance of our implementations. The results let us have a more informed choice when processing massive biomedical data.

Current work lies in extending this approach to rapid ontology annotation extraction methods to achieve very high throughputs of relation alignment in our real system. Our long term goal is to develop a highly scalable distributed analysis framework for bridging large-scale bio-information textual and formal data.

Acknowledgments. Long Cheng was supported by the DFG in grant KR 4381/1-1. We acknowledge financial support by the DFG Research Unit FOR 1513, project B1. The computations were performed on the High-performance Systems Research Cluster at IBM Research Ireland.

References

[1] P. Cimiano, *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[2] J. Völker, "Learning expressive ontologies," Ph.D. dissertation, Universität Karlsruhe, 2009.

[3] Y. Ma and F. Distel, "Learning formal definitions for Snomed CT from text," in *AIME*, 2013, pp. 73–77.

[4] Y. Ma and F. Distel, "Concept adjustment for description logics," in *K-Cap*, 2013, pp. 65–72.

[5] R. Gupta, A. Halevy, X. Wang, S. Whang, and F. Wu, "Biperpedia: An ontology for search applications," in *PVLDB*, 2014.

[6] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *ACL/AFNLP*, 2009, pp. 1003–1011.

[7] L. Yao, S. Riedel, and A. McCallum, "Collective cross-document relation extraction without labelled data," in *EMNLP*, 2010, pp. 1013–1023.

[8] A. R. Aronson and F.-M. Lang, "An overview of metamap: historical perspective and recent advances." *JAMIA*, vol. 17, no. 3, pp. 229–236, 2010.

[9] D. A. Schneider and D. J. DeWitt, "A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment," in *SIGMOD*, 1989, pp. 110–121.

[10] D. DeWitt and J. Gray, "Parallel database systems: the future of high performance database systems," *Commun. ACM*, vol. 35, no. 6, pp. 85–98, Jun. 1992.

[11] L. Cheng, S. Kotoulas, T. E. Ward, and G. Theodoropoulos, "QbDJ: A novel framework for handling skew in parallel join processing on distributed memory," in *HPCC*, 2013, pp. 1519–1527.

[12] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar, "X10: an object-oriented approach to non-uniform cluster computing," in *OOPSLA*, 2005, pp. 519–538.

[13] L. Cheng, S. Kotoulas, T. E. Ward, and G. Theodoropoulos, "Robust and efficient large-large table outer joins on distributed infrastructures," in *Euro-Par*, 2014, pp. 258–269.

[14] L. Cheng, S. Kotoulas, T. E. Ward, and G. Theodoropoulos, "Robust and skew-resistant parallel joins in shared-nothing systems," in *CIKM*, 2014.

[15] D. Kossmann, "The state of the art in distributed query processing," *ACM Comput. Surv.*, vol. 32, no. 4, pp. 422–469, Dec. 2000.

[16] P. W. Frey, R. Goncalves, M. Kersten, and J. Teubner, "Spinning relations: high-speed networks for distributed join processing," in *DaMoN*, 2009, pp. 27–33.

[17] B. Liu and E. A. Rundensteiner, "Revisiting pipelined parallelism in multi-join query processing," in *VLDB*, 2005, pp. 829–840.

[18] L. Cheng, S. Kotoulas, T. E. Ward, and G. Theodoropoulos, "Efficiently handling skew in outer joins on distributed systems," in *CCGrid*, 2014, pp. 295–304.