A parallel tag affinity computation for social tagging systems using MapReduce

Hyunwoo Kim*, Taewhi Lee and Hyoung-Joo Kim

School of Computer Science and Engineering,
Seoul National University,
1 Gwanak-ro, Gwanak-gu,
Seoul, 151-742, Korea
Email: hwkim@idb.snu.ac.kr
Email: twlee@idb.snu.ac.kr
Email: hjk@snu.ac.kr
*Corresponding author

Abstract: Tag affinity is the relationship between tags. It is a useful information for search and recommendation in social tagging systems. Tag affinity is measured by several types of tag cooccurrence frequency. The computation of tag affinity is a time-consuming task as the tagging information is accumulated. To alleviate this problem, we propose a parallel tag affinity computation method using MapReduce. We present MapReduce algorithms for computing three types of tag affinity measures: macro, micro, and bigram tag cooccurrence frequency. Our experimental results show that the proposed MapReduce-based approach not only significantly outperforms existing methods based on a relational database but also provides high scalability. To the best of our knowledge, this approach is the first tag affinity computation on MapReduce.

Keywords: parallelisation; social tagging; MapReduce; Hadoop; tag affinity; tag cooccurrence frequency; bigram; big data.

Reference to this paper should be made as follows: Kim, H., Lee, T. and Kim, H-J. (2014) 'A parallel tag affinity computation for social tagging systems using MapReduce', *Int. J. Big Data Intelligence*, Vol. 1, No. 3, pp.141–150.

Biographical notes: Hyunwoo Kim is a PhD candidate in School of Computer Science and Engineering, Seoul National University. His research interests include social tagging, recommendation, information retrieval, and database.

Taewhi Lee is a PhD candidate in School of Computer Science and Engineering, Seoul National University. His research interests include database, semantic web, text/graph data retrieval, and large-scale data processing.

Hyoung-Joo Kim is a Professor in School of Computer Science and Engineering, Seoul National University. He received his PhD in Computer Science from University of Texas at Austin in 1988. His research interests include database, semantic web, big data, and large-scale data processing.

1 Introduction

Social tagging has been essential to the success of social media services such as Flickr, Delicious, and YouTube (Das et al., 2012). Other services also have incorporated the social tagging into their systems to utilise collective intelligence, including CiteULike, BibSonomy, Last.fm, and GroupLens. One of the main purposes of social tagging is retrieval (Ames and Naaman, 2007). A tag is a keyword, which is annotated by a user to an item such as a bookmark, movie, photo, or user. Tags play an increasing role in the information retrieval process because they describe items or express user impressions for the items. Through social tagging, folksonomy is generated and non-hierarchical categories or indexes are created for retrieval. A user is able to annotate an item with any number of tags and many users can annotate the same item on collaborative tagging systems (Smith, 2007).

Tag affinity is an important measure and is utilised by information retrieval systems. The tag affinity indicates the relationships between tags. Semantic relationship between tag generates emergent semantics and enhances user experience (García-Plaza et al., 2012; Specia and Motta, 2007). Generally, the tag affinity of two tags is measured by the tag cooccurrence frequency (Xu et al., 2006). It is also utilised in tag recommendation approach in Flickr (Sigurbjörnsson and Van Zwol, 2008). When two tags are frequently annotated to the same item, it implies that the two tags share a common concept and they are semantically related. When two tags are rarely annotated to the same item, it implies that the two tags have no relationship.

With the rise of large-scale social services, computation of the tag cooccurrence frequency has become a challenging task because it is computed over either the whole dataset or a personomy that is a user's folksonomy. The basic data structure of tagging information is a triple (Hotho et al., 2006), denoted by $<u_1, r_1, t_1>$, that represents a user u_1 annotated resource r_1 with tag t_1. When triples are stored in a relational database table, it needs to execute a SQL query that contains a self-join for computing the tag cooccurrence frequency of two tags. Therefore, as the tagging data is accumulated, computing the tag cooccurrence frequencies of all possible tag pairs is more time-consuming, and the process will be infeasible.

To resolve this scalability problem, we propose parallel computation methods for tag affinity using MapReduce (Dean and Ghemawat, 2004) that is a well-known distributed processing framework. To the best of our knowledge, this approach is the first tag affinity computation using the MapReduce framework. In this paper, we consider three types of tag cooccurrence frequency: macro, micro, and bigram tag cooccurrence frequency. We present MapReduce algorithms for each tag cooccurrence frequency. A series of experiments was conducted on a cluster with 11 nodes. The result of evaluation shows that our proposed parallel tag affinity computation is more efficient compared to an existing method using a relational database. Moreover, the proposed approach is scalable in terms of speedup and efficiency.

The remainder of this paper is as follows. Section 2 reviews background information related to this study. Our parallel tag affinity computation methods are described in Section 3. Section 4 presents the experimental evaluation. Section 5 introduces the previous research work. Finally, we summarise our proposed approach and present directions for future work in Section 6.

2 Preliminaries

In this section, we present the basic concepts related to this study. We first describe tag affinity measures including macro, micro, and bigram tag cooccurrence frequencies. Next, we briefly introduce the MapReduce framework.

2.1 Tag affinity measures

Tag affinity is measured in several ways (Markines et al., 2009). Most common measures are macro and micro tag cooccurrence frequencies (Lee et al., 2012). In the macro tag cooccurrence, two tags are counted as cooccurring when the two tags are annotated to the same resource. It reflects the general meaning of two tags. Meanwhile, the micro tag cooccurrence is counted when two tags are annotated to the same resource by the same user. It reflects the user's personal usage of two tags. When tagging information is stored in the order of user input, another measure for tag affinity is bigram tag cooccurrence frequency, which assumes that adjacent tags are more semantically related than other tags (Kim et al., 2009). This is because the process of tagging is analogous to the chain of thought. In the bigram approach, there is no macro and micro classification because the bigram tag cooccurrence for different users is meaningless. The bigram approach is a special case of micro tag cooccurrence.

Figure 1 A tagging example for tag cooccurrence frequency

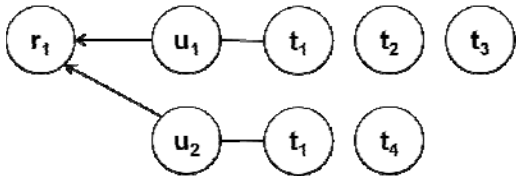


Figure 1 is a tagging example for resource r_1. It indicates that user u_1 annotates tags t_1, t_2, and t_3 to resource r_1 and user t_2 annotates tags t_1 and t_4 to resource r_1. Given this situation, the macro tag cooccurrences are all combinations of tags in the example: (t_1, t_2), (t_1, t_3), (t_1, t_4), (t_2, t_3), (t_2, t_4), and (t_3, t_4).The micro tag cooccurrences are (t_1, t_2), (t_1, t_3), (t_2, t_3), and (t_1, t_4). Note that the pairs (t_2, t_4), and (t_3, t_4) are excluded in the micro tag cooccurrence because they are not annotated by the same user. The bigram tag cooccurrences in this example are (t_1, t_2), (t_2, t_3), and (t_1, t_4). In contrast to the micro tag cooccurrences, (t_1, t_3) is excluded because the two tags are not adjacent.

2.2 MapReduce

To alleviate the scalability problem in social tagging systems, we introduced MapReduce to our approach. The MapReduce framework is a programming model for parallel and distributed processing of large datasets (Dean and Ghemawat, 2004). A MapReduce job consists of a map and a reduce phases. In the map phase, the framework splits input files and the files are read by map workers that is called mappers. The mappers execute a map function and then generate intermediate files on local disks. The intermediate files are remotely read by reduce workers that is called reducers. In the reduce phase, the reducers execute a reduce function and write output files as results. In our proposed approach, mappers split data into (key, value) pairs, then reducers merge the tag cooccurrence frequency and store the resulting output files. In the following section, we describe our approach in detail.

3 Parallel tag affinity computation

In this section, we describe MapReduce algorithms for counting micro, macro and bigram tag cooccurrence frequencies. We assume that each line of input data files consists of a triple (*user, resource, tag*) as shown in Table 1. The input data is stored in an underlying distributed file system. We utilise Hadoop that is an open source implementation of the MapReduce framework with Hadoop distributed file system (HDFS).

Table 1 Triples in input data files

User	*Resource*	*Tag*
u_1	r_1	t_1
u_2	r_2	t_2
…	…	…

3.1 Micro tag cooccurrence frequency

Micro tag cooccurrence frequency is computed in two MapReduce jobs. Job 1 splits the input data and generates the tag pairs for the same (*user*, *resource*) key. Job 2 sorts the tag order and merges the partial tag cooccurrence frequency.

Figure 2 shows MapReduce job 1 for counting micro tag cooccurrence frequency. In the map phase, each mapper reads an input data split and generates the intermediate results after converting each line to a key-value pair of <(*user*, *resource*), *tag*>. For instance, the first row of input record (u_1, r_1, t_1) is converted to the key (u_1, r_1), value (t_1) pair. The intermediate results are read by reducers. In the reduce phase, each reducer obtains all tag values for the same key (*user*, *resource*). Then the reducer generates combinations of tag pairs for the same key (*user*, *resource*) with a single count. In Figure 2, when the first reducer obtains tags t_1, t_3, and t_2 for the key (u_1, r_1), the reducer generates (t_1, t_3, 1), (t_3, t_2, 1), and (t_1, t_2, 1) as results. The output files of the MapReduce job 1 are then utilised as input files of MapReduce job 2.

Figure 3 shows MapReduce job 2 for counting micro tag cooccurrence frequency. In the map phase, each mapper reads an input split that is generated by the MapReduce job 1. The mappers sort two tags by ascending order of tag id. For example, when the input data is (t_3, t_2, 1), the mapper stores the intermediate result as <(t_2, t_3), 1> . If the system omits this sorting process, the tag cooccurrences <(t_3, t_2), 1> and <(t_2, t_3), 1> are not merged into the same tag cooccurrence frequency and the wrong result will be obtained. Both <(t_3, t_2), 1> and <(t_2, t_3), 1> indicate that tag t_2 and tag t_3 cooccur once. The sorted intermediate results are read by the reducers. In the reduce phase, each reducer merges the tag cooccurrence frequency for the same key (*tag1*, *tag2*). Then, reducers store the micro tag cooccurrence frequencies for every tag pair in the dataset. Given example, the result file consists of four tag pairs for micro tag cooccurrence. We will compare this result to macro and bigram tag cooccurrence for the same input file in the following sections.

Figure 2 Computation for micro tag cooccurrence frequency: MapReduce job 1

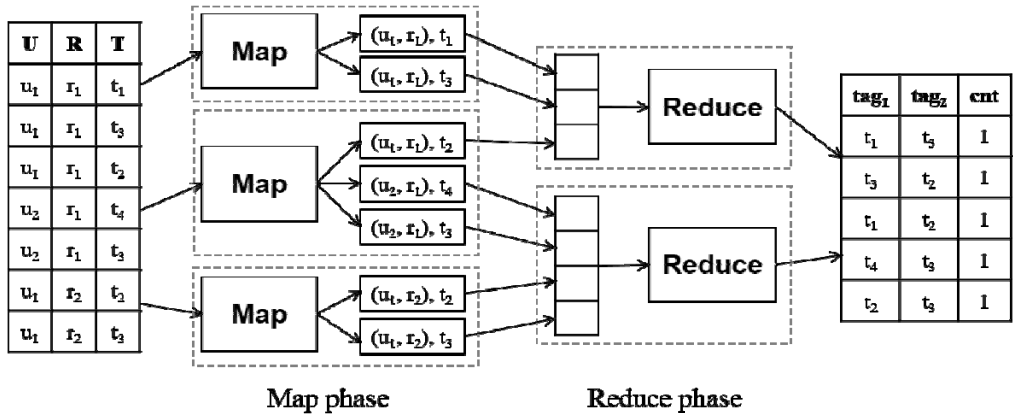
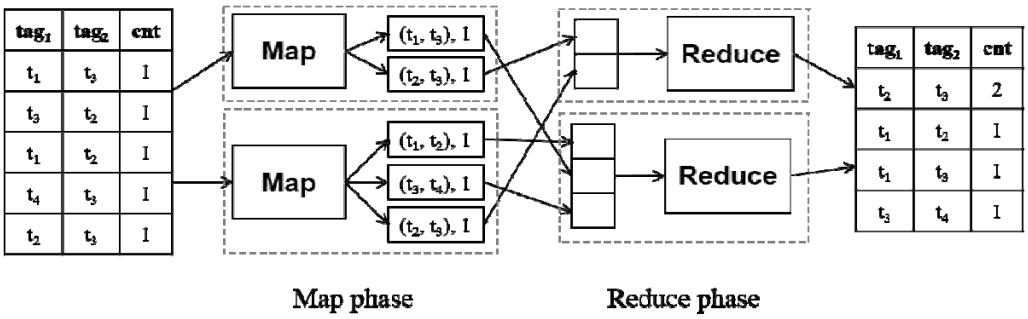


Figure 3 Computation for micro tag cooccurrence frequency: MapReduce job 2



3.2 Macro tag cooccurrence frequency

Analogous to micro tag cooccurrence frequency, macro tag cooccurrence frequency is computed in two MapReduce jobs as well. However, the MapReduce job 1 for macro tag cooccurrence differs from micro tag cooccurrence in the key of the map phase. For counting micro tag cooccurrence frequency, the map output records are generated with the key of (*user*, *resource*) because it is counted when two tags are annotated to the same resource by the same user. On the other hand, the map output records are generated with the key of (*resource*) because macro tag cooccurrence frequency is counted when two tags are annotated to the same resource, including the tags annotated by different users. In addition, the duplication of the tag id should be eliminated for counting macro tag cooccurrence frequency.

Figure 4 shows MapReduce job 1 for counting macro tag cooccurrence frequency. In the map phase, each mapper reads an input split and generates intermediate results after converting each line to a key-value pair of <(*resource*), *tag*>. In Figure 4, the first row of input record (u_1, r_1, t_1) is converted to the key (r_1), value (t_1) pair. The intermediate results are read by reducers. In the reduce phase, each reducer obtains all tag values for the same key (*resource*). Then the reducer generates all combination of tag pairs for the same resource with a single count, similarly to counting micro tag cooccurrence frequency. However, the duplication of the tag id should be eliminated because more than two users may annotate the same resource with the same tag. For example, when initial triples are (u_1, r_1, t_1), (u_2, r_1, t_1), and (u_3, r_1, t_2), the corresponding intermediate results are <(r_1), t_1>, <(r_1), t_1>, and <(r_1), t_2>. Then the reducer generates (t_1, t_2, 1) as a result. If this process is identical to counting micro tag cooccurrence frequencies, i.e., without duplication elimination, a reducer for key (r_1) will generate all combinations of tags t_1, t_1, t_2: (t_1, t_1, 1), (t_1, t_2, 1), and (t_1, t_2, 1). These results lead to incorrect macro cooccurrence frequencies of the value 2 for the pair t_1 and t_2, and the value 1 for the pair $t_1$and t_1. Given this situation, the correct answer is that the macro tag cooccurrence frequency between tags t_1 and t_2 is one. The frequency of the other tag pair is zero because, from the viewpoint of resource r_1, tag t_1 and t_2 are annotated to r_1. To calculate the correct result, the reducers should eliminate duplicated tag ids for the same resource.

Figure 4 Computation for macro tag cooccurrence frequency: MapReduce job 1

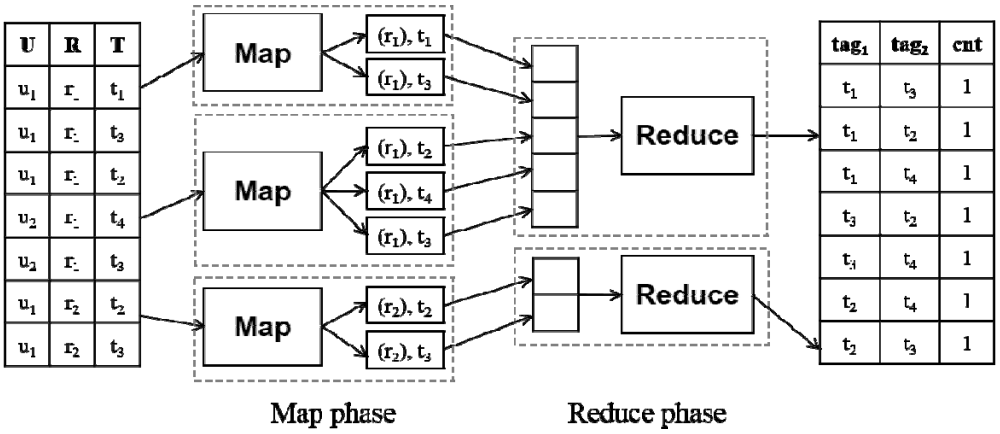


Figure 5 Computation for macro tag cooccurrence frequency: MapReduce job 2

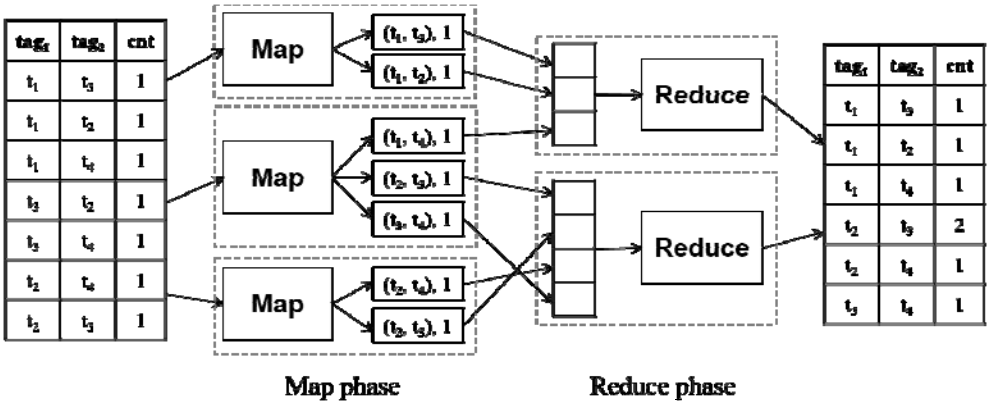


Figure 5 shows MapReduce job 2 for counting macro tag cooccurrence frequency. This process is identical to that of counting micro tag cooccurrence. Each mapper reads input data that was generated by the MapReduce job 1. The mappers sort the tags by ascending tag id and store the intermediate results. The results are read by the reducers and each reducer merges the tag cooccurrence frequency for the same key (tag_1, tag_2). Then, the reducers store the macro tag cooccurrence frequencies for every tag pair in the dataset. The result file consists of six tag pairs for macro tag cooccurrence. Given the identical input file with micro tag cooccurrence, the number of resultant tag pairs of the macro tag cooccurrence is larger than that of micro tag cooccurrence. Macro tag cooccurrence considers more combinations of tag pairs and stores a larger number of resultant tag pairs. This is why the computational cost of macro tag cooccurrence is higher than that of micro tag cooccurrence in the experiments.

3.3 Bigram tag cooccurrence frequency

In bigram tag cooccurrence, the sequence of tagging information is important because it only counts adjacent tags. We assume that the dataset preserves the tag order and its tagging information have been stored in the order that is annotated by users.

Unlike macro and micro tag cooccurrence frequencies, bigram tag cooccurrence frequency is computed in a single MapReduce job. Counting micro or macro tag cooccurrence frequency requires a merging phase for the same resource or both the same resource and user, respectively. However, bigram tag cooccurrence counting does not require the process. It sequentially reads input files and generates sorted tag pairs with a single count.

Figure 6 shows the single MapReduce job for bigram tag cooccurrence counting. In the map phase, each mapper sequentially reads lines of input data, verifies the identical (*user*, *resource*) pair, and stores tags with the same (*user*, *resource*) pair in adjacent rows. The intermediate results consist of (tag_1, tag_2, 1) triples. In the reduce phase, each reducer merges the partial frequencies for the same key (tag_1, tag_2) and then produces bigram tag cooccurrence frequencies for every tag pair in the dataset. The result file for bigram tag cooccurrence consists of three tag pairs. Given the identical input file with micro and macro tag cooccurrence, the number of resultant tag pairs of the bigram tag cooccurrence is smaller than that of micro and macro tag cooccurrence. This is because bigram tag cooccurrence considers only adjacent tag pairs and stores a smaller number of resultant tag pairs.

4 Evaluation

In this section, our experimental results are presented. A series of experiments was conducted on three types of datasets from Delicious that is one of the popular social tagging systems. It provides the social tagging to save, organise, and discover interesting links on the web. We compared the performance of our proposed method to that of existing methods in RDBMS. To confirm the scalability of our methods, we also conducted experiments varying the number of cluster nodes.

4.1 Datasets

Experiments were conducted to evaluate the proposed approaches on Delicious datasets. Delicious does not provide the whole dataset of tagging information for research purpose. We used three datasets crawled from the Delicious website, which are denoted as DEL, PINTS, and SocialBM, respectively. Table 2 shows the statistics of the datasets.

Figure 6 Computation for bigram tag cooccurrence frequency: MapReduce job

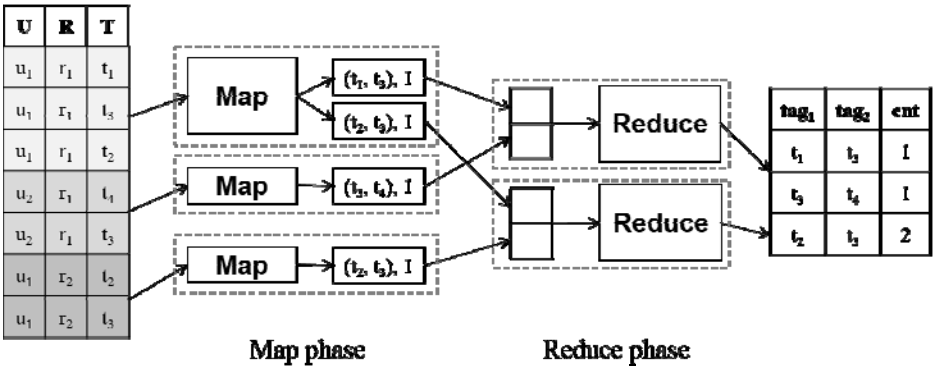


Table 2 Datasets for parallel tag cooccurrence counting

Dataset	*User*	*Resource*	*Tag*	*Triples*
DEL	638,013	10,826	311,590	13,510,165
PINTS	532,924	2,481,698	17,262,480	140,126,586
SocialBM	1,951,207	118,520,382	14,723,731	1,026,146,334

DEL is a dataset crawled by us. Our crawler starts from a webpage of popular tags on the website. When the crawler selects a popular tag, it obtains a list of resources that are annotated by this tag. Then, the crawler collects the social tagging information of the resources in the list. The popular tags includes blog, design, programming, software, music, and art. As we mentioned, tagging information is stored as a triple (*user*, *resource*, *tag*). The DEL dataset includes 638,013 users, 10,826 resources, 311,590 tags, and 13,510,165 tag assignments. In the DEL dataset, the number of users is relatively large and the number of resources is relatively small compared to other datasets. This situation arose due to the crawling process. Because our crawler collected tagging information based on the list of resources, the number of resources is not large. Furthermore, a large number of users were crawled because the crawler stored the whole user tagging information of the resource. If the crawler collects information not based on resources but on users, the number of users will be relatively small and a large number of resources will be crawled. PINTS represents the dataset crawled by Görlitz et al. (2008) and SocialBM represents the dataset crawled by Zubiaga et al. (2012). The SocialBM dataset includes 339,897,227 bookmarks and more than one billion triples.

4.2 Comparison with existing methods

To verify the difficulties of the computation on a relational database, we performed experiments which compare the performance of our proposed methods to that of the existing methods using SQL queries on RDBMS (Lee et al., 2012). In their comparison, macro and micro tag cooccurrence frequencies were computed with millions of queries, which incur excessive overhead. In our comparison, the macro and micro tag cooccurrence frequencies were computed with a single SQL query. We stored triples in a table on an Oracle DBMS on a standalone server (Intel® Xeon® 3.00 GHz CPU with 16 GB memory). All possible indices were generated for optimal performance. Our proposed methods were implemented on Hadoop (Version 1.1.2). The specifications of the node in a cluster were Intel® Core™ i5 3.10 GHz CPU with 4 GB memory.

First experiments were conducted in a single node environment. We measured the execution times for computing macro, micro, bigram tag cooccurrence frequencies for the DEL dataset, which is the smallest dataset among the test datasets. Other datasets are much larger than the DEL dataset, so macro tag cooccurrence cannot be computed within feasible execution times.

Table 3 shows the execution times for tag affinity computation. The first column specifies the tag affinity measures. The second and third columns show the execution times on RDBMS and Hadoop, respectively, formatted as *hours:minutes:seconds*. The last column shows the number of tag pairs in the results. It is observed that computation of macro tag cooccurrence frequency took more execution time than micro one. As we mentioned, this is because the number of tag pairs for macro tag cooccurrence is much larger than the number of tag pairs for micro tag cooccurrence. This is a result of the crawling method, where we collected information based on resources so that the number of tags annotated to a single resource is significantly large. Compared to the execution times on RDBMS, the execution time on Hadoop is significantly reduced. For macro tag cooccurrence frequency, the computation took about four hours on RDBMS and the execution time on Hadoop is 17 minutes 18 seconds.

Table 3 Execution times for tag affinity computation on the DEL dataset

Tag affinity	*Existing methods on RDBMS*	*Our proposed methods on Hadoop*	*No. of tag pairs in results*
Macro	03:54:43	00:17:18	180,540,339
Micro	00:03:41	00:02:14	3,123,009
Bigram	04:12:09	00:00:37	1,548,240

From the above experimental evaluations, we conclude that the parallel macro, micro, and bigram tag cooccurrence computations are much faster than corresponding methods on a relational database. They are scalable approaches and the processes are terminated within feasible execution times.

4.3 Performance on large-scale datasets

To verify speedup and efficiency of the proposed parallel tag affinity computation on large-scale social tagging datasets, experiments were conducted on the PINTS and SocialBM datasets on a cluster of 11 machines that consists of one jobtracker and ten task trackers.

Table 4 shows the results for tag affinity computation on the PINTS and SocialBM datasets. On the PINTS dataset, tag affinity computations for 140 million triples terminate within 12 minutes. On the SocialBM dataset, one billion triples are processed for bigram tag cooccurrence within eight minutes and for macro tag cooccurrence within three hours. On both datasets, macro, micro, and bigram tag cooccurrence counting is efficiently conducted in feasible execution times.

Table 4 Execution times for tag affinity computation on the PINTS and SocialBM datasets

Dataset	*Tag affinity*	*Execution time*	*No. of resultant tag pairs*
PINTS	Macro	00:11:22	391,364,682
	Micro	00:05:17	87,501,457
	Bigram	00:01:13	12,893,504
SocialBM	Macro	02:53:27	4,526,977,222
	Micro	00:30:04	246,717,638
	Bigram	00:07:29	86,809,890

4.4 Performance with various number of nodes

We performed a series of experiments, varying the number of task trackers from one to ten. Similarly to the first experiments, the specification of each node in the cluster is an Intel® Core™ i5 3.10 GHz CPU with 4 GB memory. We evaluated the execution time of macro tag cooccurrence counting on the Delicious dataset. For evaluation, we selected the macro tag cooccurrence counting, which is the most time-consuming task in Table 3 on the DEL dataset on Hadoop. Table 5 shows the results of the evaluation.

First column of the table is the number of nodes in the cluster. Other columns in the table indicate the execution times for each task in an *hours*:*minutes*:*seconds* format. The second to fourth columns give times for MapReduce job 1 of Figure 4. The fifth to seventh columns show times for MapReduce job 2 of Figure 5. Mapper and reducer columns show the maximum execution time among all the mappers and reducers that participated in the process, respectively. The total column gives the execution time to terminate the job. The results in the table indicate that the total execution time of the process, in the last column, decreases as the number of nodes increases.

Figures 7 and 8 show the execution times of the map and reduce phases separately.

Table 5 Execution times of macro tag cooccurrence counting on the Delicious dataset

No. of nodes	*Job 1*			*Job 2*			*Total execution time*
	Mapper	*Reducer*	*Total*	*Mapper*	*Reducer*	*Total*	
1	00:00:11	00:03:14	00:03:28	00:00:29	00:13:13	00:13:50	00:17:18
2	00:00:16	00:02:18	00:02:26	00:00:55	00:09:59	00:00:23	00:12:49
3	00:00:10	00:02:06	00:02:13	00:00:42	00:06:51	00:07:15	00:09:28
4	00:00:11	00:01:34	00:01:41	00:00:38	00:04:48	00:05:08	00:06:49
5	00:00:10	00:01:27	00:01:33	00:00:31	00:03:59	00:04:19	00:05:52
6	00:00:10	00:01:05	00:01:12	00:00:20	00:02:14	00:02:34	00:03:46
7	00:00:10	00:01:10	00:01:16	00:00:21	00:01:58	00:02:18	00:03:34
8	00:00:11	00:00:59	00:01:05	00:00:22	00:01:30	00:01:50	00:02:55
9	00:00:10	00:00:50	00:00:57	00:00:22	00:01:23	00:01:43	00:02:40
10	00:00:10	00:00:41	00:00:47	00:00:22	00:01:11	00:01:32	00:02:19

Figure 7 Execution times for MapReduce job 1

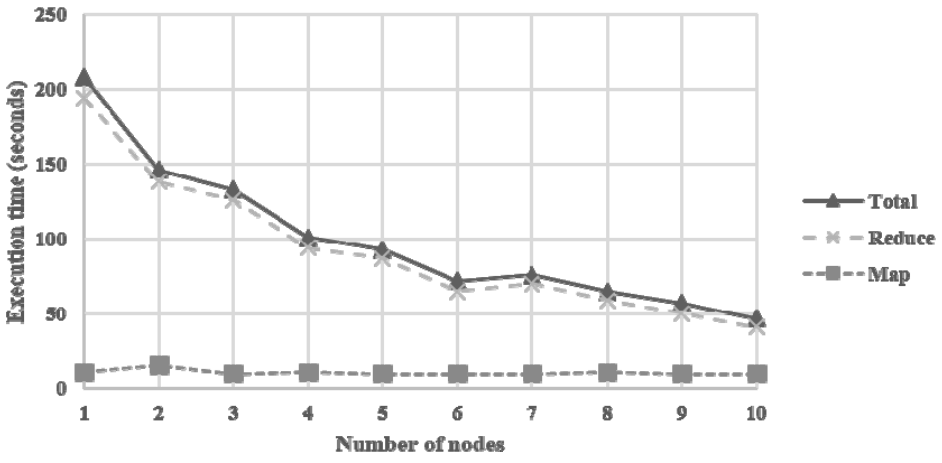


Figure 8 Execution times for MapReduce job 2

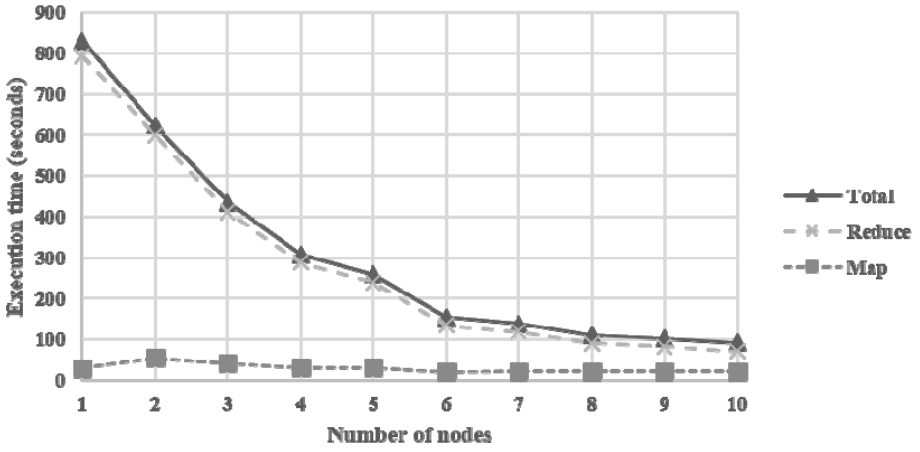
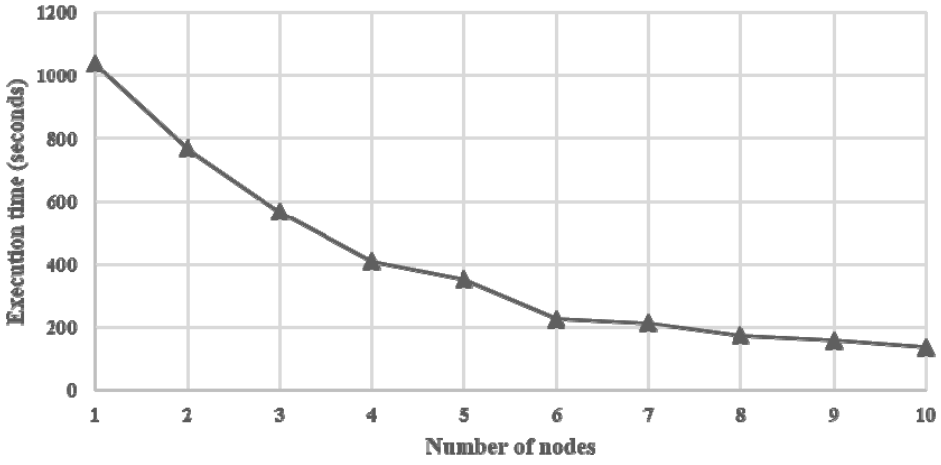


Figure 9 Total execution times for macro tag cooccurrence on the Delicious dataset



In the map phase, because the size of input data is constant and the number of mappers does not change, the map execution time does not change as the number of nodes increases. However, in the reduce phase, the execution time decreases as the number of nodes increases. Incrementing the number of reducers improves the performance.

Figure 9 shows the performance gain of a cluster environment. The more nodes are added to the cluster, the faster the execution is conducted. It is reasonable to expect that the execution time of ten-node cluster is smaller than that of 1-node cluster. We evaluate the speedup of the proposed parallel approach, which can be defined as follows (Eager et al., 1989).

$$S(n) = \frac{T_1}{T_n}$$

where $S(n)$ is the speedup of an n-node cluster, T_1 is the execution time of a 1-node cluster and T_n is the execution time of an n-node cluster.

Figure 10 shows the experimental results for speedup of the proposed parallel approach. The proposed approach does not achieve linear or ideal speedup. Data transfer and communication costs among the tasks hinder the linear speedup. However, the ratio of speedup is relatively close to the ideal speedup as the number of nodes increases. The ratio of speedup is the efficiency of the process, where the efficiency of an n-node cluster $E(n)$ is the average utilisation of the cluster and is defined as follows (Eager et al., 1989).

$$E(n) = \frac{S(n)}{n}$$

where n is the number of nodes in the cluster and $S(n)$ is the speedup of the n-node cluster. Figure 11 gives the experimental results for the efficiency of clusters from one to ten nodes, showing that the efficiency is not reduced as the number of nodes increases; even the efficiency increases.

From the series of experimental evaluations above, we conclude that the proposed parallel tag affinity computation is an efficient and scalable approach.

Figure 10 Speedup of parallel tag affinity

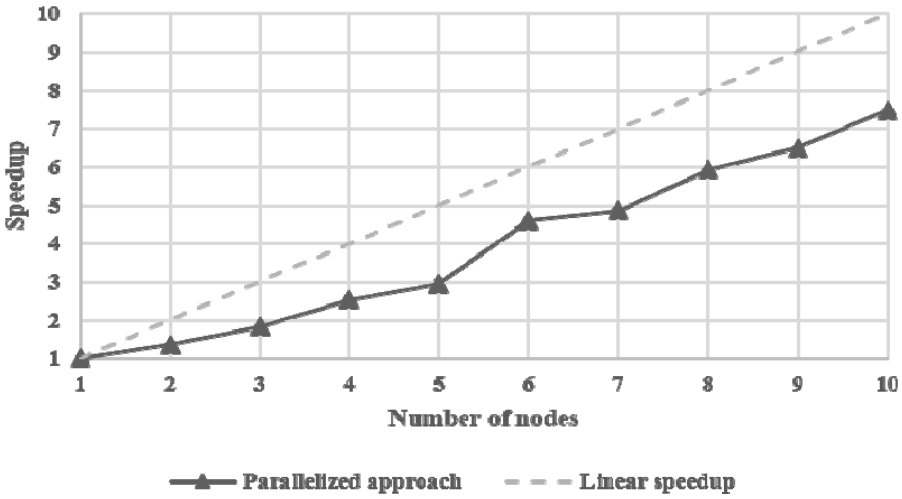
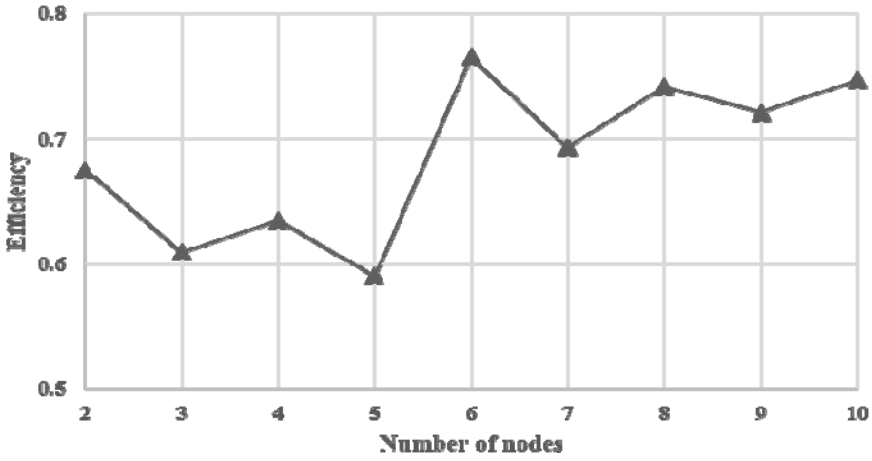


Figure 11 Efficiency of the parallel tag affinity calculation



5 Related work

To the best of our knowledge, there is no previous research study on the computation of tag affinity computation using the MapReduce framework. However, there exist previous research studies on parallel approaches in social tagging systems. Liang et al. (2010) proposed a parallel user profiling approach based on folksonomy information. The scalable recommender systems were implemented based on cascading. Zhao and Shang (2010) proposed a user-based collaborative filtering algorithm for the MapReduce framework. Jiang et al. (2011) described the limitations of Zhao and Shang's method and proposed a better scalable item-based collaborative filtering algorithm on the MapReduce framework. De Pessemier et al. (2011) provided details about how to calculate collaborative filtering and pairwise similarities on a MapReduce framework.

6 Conclusions

In this paper, we proposed MapReduce algorithms for computing three types of tag affinity measures: macro, micro, and bigram tag cooccurrence frequency. Macro and micro tag cooccurrence frequencies are computed in two MapReduce jobs similarly but de-duplications of intermediate results is required for the macro tag cooccurrence. Bigram tag cooccurrence is computed in one MapReduce job assuming that the tag order is preserved in input files. The experimental results present that the proposed approach shows better performance than existing methods on a relational database. Furthermore, the proposed approach is scalable from the perspective of speedup and efficiency. In future work, we will investigate tag frequency and normalised tag cooccurrence frequency computation using the MapReduce framework simultaneously.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 20120005695).

References

Ames, M. and Naaman, M. (2007) 'Why we tag: motivations for annotation in mobile and online media', *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp.971–980, ACM.

Das, M., Thirumuruganathan, S., Amer-Yahia, S., Das, G. and Yu, C. (2012) 'Who tags what? An analysis framework', *Proceedings of the VLDB Endowment*, Vol. 5, pp.1567–1578.

De Pessemier, T., Vanhecke, K., Dooms, S. and Martens, L. (2011) 'Content-based recommendation algorithms on the Hadoop MapReduce framework', *Proceedings of the 7th International Conference on Web Information Systems and Technologies*, pp.237–240.

Dean, J. and Ghemawat, S. (2004) 'MapReduce: simplified data processing on large clusters', *Proceedings of the 6th Symposium on Operating System Design and Implementation*, pp.137–149.

Eager, D.L., Zahorjan, J. and Lazowska, E.D. (1989) 'Speedup versus efficiency in parallel systems', *IEEE Transactions on Computers*, Vol. 38, No. 3, pp.408–423.

García-Plaza, A., Zubiaga, A., Fresno, V. and Martínez, R. (2012) 'Reorganizing clouds: a study on tag clustering and evaluation', *Expert Systems with Applications*, Vol. 39, No. 4, pp.9483–9493.

Görlitz, O., Sizov, S. and Staab, S. (2008) 'PINTS: peer-to-peer infrastructure for tagging systems', *Proceedings of the 7th International Conference on Peer-to-peer Systems*, p.19.

Hotho, A., Jäschke, R., Schmitz, C. and Stumme, G. (2006) 'Information retrieval in folksonomies: search and ranking', *Proceedings of the 3rd European Semantic Web Conference*, pp.411–426, Springer.

Jiang, J., Lu, J., Zhang, G. and Long, G. (2011) 'Scaling-up item-based collaborative filtering recommendation algorithm based on Hadoop', *IEEE World Congress on Services*, pp.490–497, IEEE.

Kim, H., Lee, K., Shin, H. and Kim, H-J. (2009) 'Tag suggestion method based on association pattern and bigram approach', *Proceedings of the 10th ACIS International Conference on Software Engineering*, *Artificial Intelligences, Networking and Parallel/Distributed Computing*, pp.63–68, IEEE.

Lee, K-P., Kim, H-G. and Kim, H-J. (2012) 'A social inverted index for social-tagging-based information retrieval', *Journal of Information Science*, Vol. 38, No. 4, pp.313-332.

Liang, H., Hogan, J. and Xu, Y. (2010) 'Parallel user profiling based on folksonomy for large scaled recommender systems: an implementation of cascading MapReduce', *IEEE International Conference on Data Mining Workshops*, pp.154–161, IEEE.

Markines, B., Cattuto, C., Menczer, F., Benz, D., Hotho, A. and Stumme, G. (2009) 'Evaluating similarity measures for emergent semantics of social tagging', *Proceedings of the 18th International Conference on World Wide Web*, pp.641–650, ACM.

Sigurbjörnsson, B. and Van Zwol, R. (2008) 'Flickr tag recommendation based on collective knowledge', *Proceedings of the 17th International Conference on World Wide Web*, pp.327–336, ACM.

Smith, G. (2007) *Tagging: People-powered Metadata for the Social Web*, New Riders.

Specia, L. and Motta, E. (2007) *Integrating Folksonomies with the Semantic Web. The Semantic Web: Research and Applications*, Springer, Austria.

Xu, Z., Fu, Y., Mao, J. and Su, D. (2006) 'Towards the semantic web: collaborative tag suggestions', *Collaborative Web Tagging Workshop*, Scotland.

Zhao, Z-D. and Shang, M-S. (2010) 'User-based collaborative-filtering recommendation algorithms on Hadoop', *Proceeding of the 3rd International Conference on Knowledge Discovery and Data Mining*, pp.478–481, IEEE.

Zubiaga, A., Fresno, V., Martínez, R. and García-Plaza, A. (2012) 'Harnessing folksonomies to produce a social classification of resources', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 25, No. 8, pp.1801–1813.