A Semantic Double-Buffer Based Approach to Enhance Semantic Web Search*

Ling Chen, Hai Jin, Sheng Di
Cluster and Grid Computing Lab
Services Computing Technology and System Lab
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@hust.edu.cn

Abstract

With the development of semantic web, search efficiency is becoming a challenging issue. To address this problem, we design an approach which mainly has two features. First, we explore potential semantic relationship between different objects and make full use of them to build a semantic buffer on server end, enhancing search speed. Second, we put another relatively small semantic buffer on each client so as to adapt to each individual user's interests, further improving the whole search efficiency and reducing server's load. Through testing on a practical platform, we testified that this Semantic Double-Buffer based search approach can not only effectively reduce user's request response time, but considerably release server's load, bringing a high scalability.

1. Introduction

With the emergence and development of semantic web [1], search methods based on semantic web techniques promise advantages compared to conventional approaches. Semantic web is an extension of the current web, based on the idea of exchanging information with explicit, formal and machine-accessible descriptions of meaning. Semantic web contains resources corresponding not only to real world objects (e.g., texts, images, people, places, organizations), but also to relationships between objects [2]. Semantic web search can reveal relationship and semantic related information.

There are some solutions to address how to store and search objects and relationship in semantic web encoded with RDF [3], and the most famous one is Sesame [4], an RDF data repository. Sesame can store and query metadata and relation in RDF(S). Yet, its running efficiency is comparatively low, especially when the data quantities scale up, in that it has to traverse all data every time users query information. As a matter of fact, we notice that some information and their semantic related information would always be searched more frequently than others. Taking literature retrieval for example, because of qualities of various papers, some classic papers and their semantic related information might often be queried, while in contrast, the access probability of comparatively plain ones would be much less. Furthermore, for an individual user, because of his/her specific interests, some information and their related information may also be accessed more frequently. So we believe that sufficiently mining relationship among objects and buffering them in a suitable way would immensely improve search efficiency. Based on this thought, we explore potential semantic relationship between objects and design an algorithm to buffer them on server-end, and design another buffering algorithm on client-end to adapt to every individual's interests, for further improving search efficiency.

It is necessary to note that to explain the applicability of our approach, we select scientific literature as research domain in this paper. But it is also appropriate to extend it to other application domains, such as document search and music search.

The rest of this paper is organized as follows. In section 2, we introduce the related work. Section 3 introduces the architecture of semantic double-buffer based search approach. Section 4 discusses the algorithms of semantic double-buffer based search. In section 5 we test the performance of our approach. Finally, we present some concluding remarks.

* This work is supported by National Basic 973 Research Program of China under grant No.2003CB317003, and the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China under grant 705034.

0-7695-3087-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDS.2008.13

2. Related Work

To our knowledge, semantic relationship of almost all the current search engines are mainly used to present the relationship between different concepts, making related metadata more readable or clear. Presently, some related systems includes EKOSS [5], CiteSeer [6], Flink [7, 8], and so on.

EKOSS (*Expert Knowledge Ontology based Semantic Search*) is a web-based system developed for sharing expert knowledge from research fields related to bioscience, engineering and environment. Since there are some semantic-inference methods used in this system, a user can submit semantically meaningful queries and the system can return more accurate results that matches the user's search target. But EKOSS just consider semantic relationships from inference's perspective to express terms precisely. Instead, we make full use of these relationships to enhance search efficiency from many aspects.

CiteSeer is an autonomous citation indexing system which indexes academic literature in electronic format. It can not only understand how to parse citations, but identify citations to the same paper in a variety of formats and the context of citations in the body of articles. In CiteSeer, papers related to a given paper can be located using common citation information or word vector similarity. However, this implementation mainly focuses on the method of indexing.

Flink employs semantic technology for reasoning with personal information extracted from a number of electronic information sources including web pages, emails, publication archives and *Friend-Of-A-Friend* (FOAF) [9] profiles. It can also be regarded as a presentation of the professional work and social connectivity of semantic web researchers. Its architecture can be divided in three layers concerned with metadata acquisition, storage and visualization. The storage layer is implemented through RDF Sesame repository which is in terms of semantic web standard, but with a comparatively low search speed. That is mainly because it has to traverse through all data every time users query any information.

Just as mentioned above, all the three related systems can reveal in-depth semantic information and semantic relationships between objects. Moreover, they are able to do not only key-based searches, but relationship-based searches. It is possible for their users to search a paper via the normal way, inputting keywords, and then browse the related information on the basis of these previous search results, say, a series of cited papers. However, since there is not any buffer-related module in their architectures, the response time when querying some information, whether it is key-based or relationship-based, will definitely be more or less over-wasted.

3. Architecture

The architecture of the Semantic Double-Buffer based search is shown in Fig.1. After a user submits a request, it will be sent to a buffer-agent of user's client before being transferred to server. As long as there is target information in the client's buffer, the information will be popped out and presented straightforwardly to users. Otherwise, the request will be submitted to servers, and the result from server will be cached into the client's buffer for future use. So far, the first-level buffer which is on client has been searched.

For server-end, as soon as it receives a request, it will first search server-buffer via an agent, in that the buffer would have cached many results activated by other users' requests in the near period. If there is needed information in the buffer, then the buffer takes out the information immediately and responds back. If there is not any information matched, then querying database (Sesame) can not be avoided. After finishing searching, the results will be analyzed and structured and then put into the server-buffer. So far, the second-level buffer has been searched and structured.

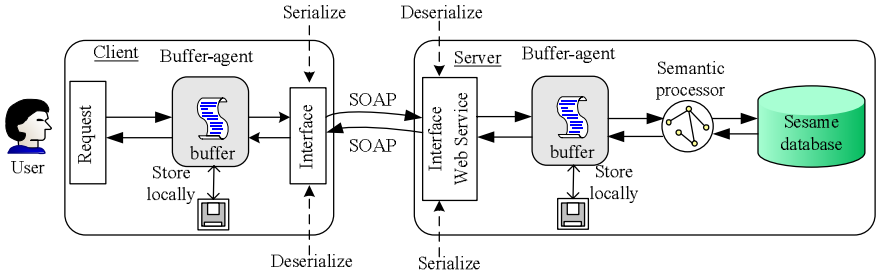

Figure 1. Architecture of Semantic Double-Buffer-Based Semantic Web Search

4. Semantic Double-Buffer Algorithm

4.1 Semantic Model

We use ontology [11-13], which is defined as shared formal conceptualizations of particular domains, to reveal objects and their relationship. We design an ontology model (simplified RDF graph) of scientific literature. This ontology model integrates all valuable objects, metadata and semantic related information of scientific literature field. As shown in Figure 2, there are two classes in the ontology model: publication and author. Each class has several properties which describe the class in detail, for example, name of the author and title of the publication. There are several relationships between classes. The relationships include write (between author and literature), co-write (between author and author), cited by (between literature and literature), cite (between literature and literature), and similar (between literature and literature). If we provide information to users according to this model, it would be of great help for searching.

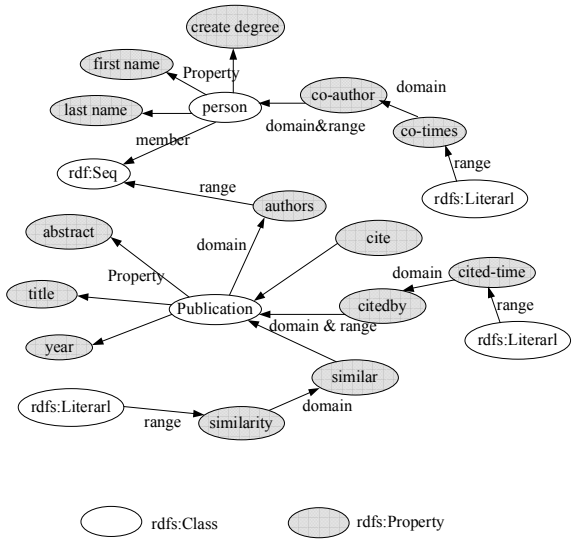

Figure 2. Ontology Model of Scientific Literature

The algorithm discussed in sessions 4.2 and 4.3 are based on this ontology model. Actually, as long as any two related objects are equivalent to each other, the relationship, defined as peer-to-peer-relationship (p2p-relationship), between them can be used in this algorithm. For instance, cite (or cited by) is between papers, co-write is between authors. It is clear that there are two key common points in the above two relationships: 1) Objects are equivalent to each other, such as paper1 and paper2 are both papers; 2) The relationship between objects is recursive. For example, any paper cited by other papers can also cites some papers else. In comparison, the relationship between author and paper is not a p2p-relationship.

4.2 Server-Buffer Algorithm

It is always the case that users not only pay attention to semantic related information of an object, but also are interested in its semantic related information's semantic related information, which is defined as two layer semantic related information in this paper. For example, a user who pays attention to an author called Denial may probably be interested in Denial's co-author Sam, and Sam's co-author may probably write valuable paper that the user wants. So it is useful to link these authors together and buffer author's co-author's co-author in a buffer. We call it multi-layer semantic relationship.

Because we not only buffer objects themselves, but also link them together and buffer multi-layer semantic relationships. Hence, users can always immediately retrieve needed information on server-end indirectly instead of directly. *Indirectly* means that users can retrieve information immediately even though no previous users requested the same author.

Based on this idea, we design a structure which can integrate objects and semantic relationships together, and propose a server buffer algorithm which describes a process on how to build this structure.

Now we define Semantic Server-Buffer Structure. This structure includes some sub-structures.

Define *Author Association Information Object* (AAIO) as follow:

AAIO = (Static Property Information, Index)
= (UID, Name, CreateDegree, index of coAuthor List)

AAIO consists of two parts: *Static Property Information Field* (SPIF) and *Index Field* (IF). *SPIF* includes three sub-fields: *UID, Name, CreateDegree*. *UID* is unique ID of author. *CreateDegree* is a coefficient used to evaluate the authority of an author. *IF* is the *index of coAuthor List*.

Define a buffer-carrier - *Link Information Buffer Structure* (LIBS), composed by a series of *AAIO*.

LIBS = (AAIO)

Define *Co-Author Association Information Object* (C-AAIO) as follow:

C-AAIO = (Degree of co-write relationship, UID of co-author) = (Cooperate Times, UID)

Cooperate Times are referred to the times of two authors co-writing papers.

Define *Co-Author Object List* (C-AOL) which is composed by a series of *C-AAIO*.

C-AOL = (C-AAIO)

In addition, define *Link Information Structure* (LIS) as follow: *LIS = (AAIO)*

LIS is only used to keep a set of *AAIO* queried based on the current request. That is to say, after processing a submitted request each time, no matter whether or not database needs to be searched, the server-buffer-agent would construct a *LIS* transmitted to clients as response. Obviously, *LIS* ⊂ *LIBS*

Here, a Semantic Server-Buffer Structure example is shown in Figure 3. In this example, Alice has three co-authors: Tony, Frank and Jane, and Tony has two co-authors: Alice and Ross.

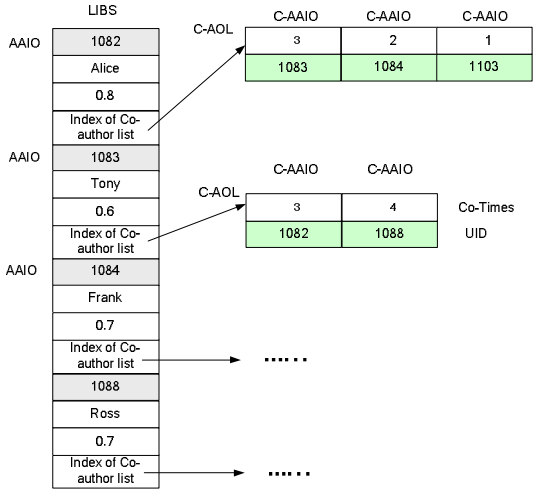

Figure 3. An Example of Semantic Server-Buffer Structure

After defining the structure, we use an iterative method to build our Semantic Server-Buffer Structure. We construct every *AAIO* and store them in *LIBS*. *LIS* is built to be transmitted to client. The detailed procedure of the algorithm is shown below.

Server-Buffer Algorithm—construction(*n*,A,*l*):
Static *n*; //Current layer of co-author-iteration
Input: A:Requested author A,
l:the layer of *multi-layer semantic relationship*.
Output: the author A which has been constructed according to LIS
Step1: 1 **if** A exists in LIBS
2 Get the object, Author A from LIBS;
3 Put A's AAIO into LIS;
4 **else**
5 Construct A's corresponding AAIO, denoted as A_AAIO;
6 Search Sesame, retrieve A's Static Property Info.;
7 Search Sesame and retrieve A's co-authors;
8 Fill out A_AAIO;
9 Put A_AAIO into LIBS and LIS;
10 **endif**
Step2: 11 **if** *n* < *l*
12 n ++; //Increase one layer.
13 Get A_AAIO's C-AOL;
14 **if** C-AOL == NULL
15 Initialize C-AOL;
16 Query A's co-authors, result is denoted as QR-A;
17 **for** Traverse each co-author of QR-A **do**
18 Take Author Name B and construct B's C-AAIO;
19 Denote B's AAIO as B_AAIO, B_AAIO = construction(*n*,B,*l*); //Iteration
20 Put B_AAIO into LIS;
21 **enddo**
22 **else** // C-AOL !=NULL
23 **for** Traverse each object of C-AOL **do**
24 Get object of one Author : c;
25 Get Name of c as C and construct C's C-AAIO;
26 Denote C's AAIO as C_AAIO, C_AAIO = construction(*n*,C,*l*);
27 Put C_AAIO into LIS;
28 **enddo**
29 **endif**
30 **endif**
31 return A_AAIO;
END

4.3 Client-Buffer Algorithm

In order to further improve the client's querying time by adapting to individuals' interests, we organize another buffer on client-end similar to that of server-end.

There are totally three kinds of data structures in client buffer. The first two are *AAIO* and *LIBS*, whose structures are the same as those of server buffer. The third one, *History*, is used to record all authors' UIDs that the current user submitted to server in the past. So this structure can be depicted as follow:

History = (Author's UID)

The construction of client buffer is relatively easier than that of server buffer. This is because the author-objects kept in client buffer are right in *LIS* that have already been built-up and transmitted from server-end, no need to be created again, just put in client buffer.

As soon as the client-buffer-agent receives a *LIS*, it would get the *LIS*'s *AAIO* (information carrier) one by one and judge whether there are corresponding carriers in client buffer. Each *AAIO* will be put into client buffer unless there is already one in it. Finally, add the current requested UID into *History*.

Following is the pseudo-code of the algorithm.

```
Client-Buffer Algorithm
Step1: 1   User submits an author A and UID to Client;
       2   if A exists in History
       3       return;
       4   else
       5       Submit request to Server;
       6       Waiting for response……;
Step2: 7       Receive querying result: LIS from server;
       8       Record A in History Map;
       9       for Traverse each LIS's AAIO do
      10           if the AAIO does not exist in client's LIE
      11               add it into LIBS;
      12           endif
      13       enddo
      14       Free LIS;
      15   endif
END
```

5. Performance Analysis

We test our design from three aspects: response time when one user requests information, average response time when lots of users request information simultaneously, the comparison of time cost among different parts in the system.

5.1 Single Request Situation

We test response time for single user to request semantic information.

We design a method to simulate a user's search behavior, which is described as follows: Step 1): Randomly select an author from database and submit the request, putting its multi-layer co-authors (returned result) into a set called Author-SET. Step 2): Select an author from Author-SET, submit the request, and add its multi-layer co-authors into the Author-SET. In the meantime, record the response time and the number of *AAIO*s to be transmitted.

Figure 4 shows the testing result. From the figure, if there is no semantic double-buffer, the response time will increase in a linear trend. This is not ideal, especially when the number of *AAIO* to be transmitted from server is up to 50, the waiting time for users would be over 25 seconds, which is because all *AAIO*s have to be queried from database. Comparatively, let us take a look at the situation with semantic double-buffer. In this case, when a user submits a request of a new author, the increased-rate of the response time is clearly smaller than the previous situation, in that the more AAIOs to be retrieved, the larger the possibility they overlap, and the more frequently the server-buffer will be accessed. Furthermore, when this user submits the same request, the corresponding response time would be down to less than 1s because of client-buffer. Hence, semantic double-buffer design does have a good efficiency in single request situation.

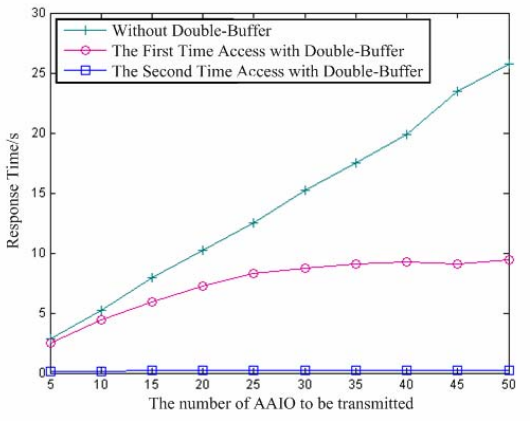

Figure 4. Response Time of Single User

5.2 Multiple Requests Situation

We use 1~11 computers (nodes) to simulate the behaviors of 10~110 users, and there are 10 suspended threads on each computer to simulate 10 users. As for the testing result, we use the same testing method above to record the response time for each virtual user, and calculate their mean value as the average response time. Figure 5 shows the logarithm of the average response time when quite a few users submit requests to the server in the meantime.

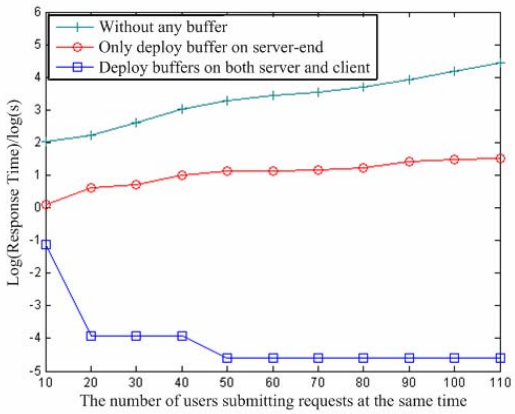

Figure 5. Logarithm of Response Time of Multi-Users

Through the testing, we can clearly see that our approach really has a high scalability. If there is no any buffer set, neither in server nor in client, the average response time will increase very fast with the increasing number of users. If there is no buffer on client but server, the average time would be reduced to about 3 seconds. As soon as we adopt semantic double-buffer, the corresponding time will be controlled far below 1 second.

5.3 Comparison of Time Cost

Through this part of testing, we testify that the time cost by processing either semantic server-buffer or client-buffer can be ignored comparing to that of searching Sesame database. In fact, Sesame really is able to do semantics/ontology-based query-operation, but all the data have to be stored in several files on disk. That is why the database is so relatively slow, that is, each querying operation for Sesame has be involved with a lot of I/O operations. However, semantics/ontology-based query operation via Sesame database provides quite a few conveniences for us to build semantic relationships and double-buffer. So, the design of our semantic-double-buffer makes full use of its efficiency, to avoid frequently accessing semantic-database to a certain extent. The detailed testing data is shown in Figure 6.

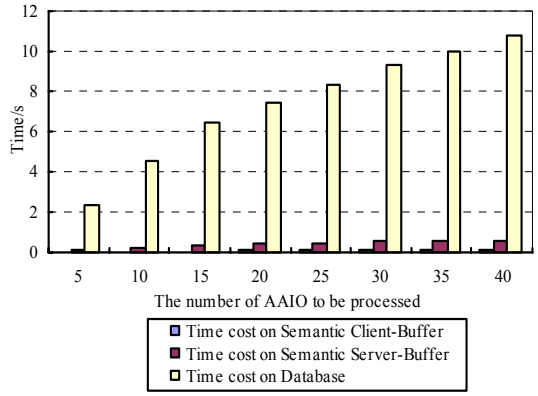

Figure 6. Cost Time Comparison of Different Parts in System

6. Conclusion

In this paper, we present the design and implementation of semantic double-buffer based search approach. We reveal semantic relationship according to ontology and design an algorithm that buffer multi-layer semantic relationship on server-end, and design another algorithm on client-end to adapt to every individual's interests, further improving search efficiency.

References

[1] S. Decker, S. Melnik, F. V. Harmelen, D. Fensel, M. Klein, J. Brockstra, M. Erdmann, and I. Horrocks, "The Semantic Web: The Roles of XML and RDF", *IEEE Internet Computing*, 15(3), 2000, pp.63-74.

[2] R. Guha and R. McCool, "TAP: A semantic web platform", *Computer Networks: The International Journal of Computer and Telecommunications Networking, Special Issue: The Semantic Web: An Evolution for a Revolution*, 42(5), 2003, pp.557-577.

[3] O. Lassila and R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, Feb. 1999: Online at http:// www.w3.org/TR/REC-rdf-syntax/.

[4] J. Broekstra, F. Van Harmelen, and A. Kampman, "Sesame: a generic architecture for storing and querying RDF and RDF Schema", *Proceedings of the First International Semantic Web Conference (ISWC'02)*, 2002, pp.54-68.

[5] S. Kraines, W. Guo, B. Kemper, and Y. Nakamura, "EKOSS: A knowledge-user centered approach to knowledge sharing, discovery and integration on the Semantic Web", *Proceedings of 5th International Semantic Web Conference*, Nov. 2006, pp.833-846.

[6] C. L. Giles, K. Bollacker, and S. Lawrence, "CiteSeer: An Automatic Citation Indexing System", *Proceedings of Third ACM Conf. on Digital Libraries*, 1998, pp.88-98.

[7] P. Mika, "Flink: Semantic Web Technology for the Extraction and Analysis of Social Networks", *Journal of Web Semantics*, 3(2), 2005, pp.211-223.

[8] Flink Website: Online at http://www.aduna-softw are.com/technologies/autofocus_server/overview.view.

[9] FOAF project: Online at http://www.foaf-project.or g.

[10] SOAP Version 1.2 Part 1: Messaging Framework: Online at http://www.w3.org/TR/soap/.

[11] F. Baader, I. Horrocks and U. Sattler, "Description logics as ontology languages for the semantic web", *Lecture Notes in Artificial Intelligence*, Springer, 2003.

[12] B. Smith and C. Welty, "Ontology: Towards a New Synthesis", *Proceedings of Formal Ontology and Information Systems (FOIS'01)*, 2001, pp.3-9.

[13] P. Borst, H. Akkermans, and J. Top, "Engineering Ontologies", *International Journal of Human-Computer Studies*, 1997, Vol.46, No.2-3, pp.365-406.

[14] X. Ning, H. Jin, and H. Wu, "SemreX: Towards Large-scale Literature Information Retrieval and Browsing with Semantic Association". *Proceedings of 2nd IEEE International Symposium on Service-Oriented Applications, Integration and Collaboration (SOAIC'06)*, Shanghai, China, 2006, pp.602-609.