An Agent-Based Linked Data Integration System

Xuejin Li, Zhendong Niu, Chongyang Shi
School of Computer Science
Beijing Institute of Technology
Beijing, China
xuejinli7@gmail.com, {zniu, cy_shi}@bit.edu.cn

ABSTRACT: *With the advent of the Web of Linked Data, new challenges to federated query processing are emerging. Different from traditional federated database systems which do static data integration, this Web of Data is open and ever-changing. In this paper, we present a agent-based architecture providing a flexible and decoupled solution for the federated queries over Linked Data. Based on the presented architecture, a Linked Data Management System (LDMS) has been developed. LDMS manages Linked Data in a virtual way, i.e., it does not load remote data into a local data store. With an application scenario, we demonstrate the scalability and extensibility of the presented architecture.*

Keywords: Linked Data, Data Management, Web Data, Scalability, LDMS, Agent Architecture, Database Systems

Received: 11 November 2013, Revised 18 December 2013, Accepted 23 December 2013

1. Introduction

The Web of Linked Data enables new types of applications which can aggregate data from different data sources and integrate fragmentary information from multiple sources to achieve a more complete view. This Web of Data is open and has grown considerably. Today it comprises over 31 billion RDF triples, which are interlinked by around 504 million RDF links (September 2011)[1]. Traditional federated database management systems have been focused on the relatively static environments of distributed databases. In these environments, the structure of data is consistent and data query performance can be optimized using pre-computed indices. Hence, these data management systems do not scale up and do not cope well with open and dynamic environments. An effective Linked Data management system has become one key challenge for many Semantic Web applications.

Due to the open and dynamic nature of the Web, Linked Data management systems must provides flexible, extensible means to manage these data sources. To achieve this flexibility and openness, we accommodate the technologies of agent into our system and propose an agent-based architecture. In this paper a system is presented which has ability in monitoring, managing and querying data sources over the Web of Linked Data. It is not attached to any other projects, but developed as an independent system. In the current version, the proposed system requires data sources accessible via SPARQL endpoints. However, it is expect to have the ability of accessing any Linked Data sources in the following version.

[1]http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

The remainder of this paper is structured as follows. In Section 2 we review related work. We discuss the architecture of the system in Section 3. Section 4 describes the agent design and implementation. A data integration application in life science domain and cross domain is briefly presented in section 5. Finally, section 6 gives the conclusion and future work.

2. Related Work

The agent technology has been widely used in Internet applications. In the environment of web, agent technology enables a high degree of decentralization which is the key to system scalability and extensibility. The University of Michigan Digital Library (UMDL) architecture encapsulates the many functionalities required in a digital library as a population of modular, goal-oriented, specialized '*agents*' [1]. A mobile agent architecture for managing distributed information is presented in [2]. Different from the scenarios which these systems applied in, developers can not exert any influence on remote data sources over the Web of Linked Data. As the most relevant work of the presented system, the InfoSleuth [3] project extends the capabilities of the Carnot technologies developed in MCC[2] into querying distributed, dynamic information sources in the World Wide Web. In traditional Web, data formats used in information sources are various. InfoSleuth has to use additional ontologies to specify both the infrastructure underlying the agent-based architecture and characterize the information contents in the underlying data repositories. In this paper, we adjust this agent-based architecture to linked data query processing.

Garlic [4] is one of the first mediator system aiming at the integration of heterogeneous distributed multimedia data. It was developed by IBM at the Almaden research center during the nineties. [5] introduces the implementation of a federated geospatial catalogue system using the mediatorwrapper framework.

While approaches to federated query processing over linked data are still in their infancy, some prototype systems exist. SemWIQ [6] uses a mediator-wrapper architecture to provide a transparent access to multiple distributed data sources. It requires data sources register themselves at the mediator by sending HTTP POST requests with RDF documents attached. However, this requirement is infeasible in the Web of Linked Data because that we can not require data publisher to do more things than publishing their data. DARQ [7] extends the popular query processor Jena ARQ to an engine for federated SPARQL queries. It adopts architecture similar to SemWIQ with not providing ability of monitoring data sources. The presented architecture was designed to monitor the Web of Linked Data and automatically discover, register and abandon data sources.

3. Agent-based Architecture

The agent technology has been widely used in internet applications. In the environment of web, agent technology enables a high degree of decentralization which is the key to system scalability and extensibility. InfoSleuth [3] provides capabilities of querying distributed, dynamic information sources in the World Wide Web. While data formats used in information sources being various in traditional web, the Web of Linked Data is in different case. Data published in this Web of Data must comply with Linked Data principles. In this paper, we adjust the agent-based architecture of InfoSleuth to linked data query processing.

3.1 Architectural Overview

The proposed architecture is comprised of a network of cooperating agents communicating by means of FIPA ACL[3]. Users submit SPARQL queries via user agents which provide user interfaces. The queries are routed by brokerage agents to specialized agents for data retrieval from remote data sources and integration of intermediate results.

Agents register their network addresses and services in the broker agent. When an agent has accomplished its work or needs help from other agents, it will send a request to the broker agent which routes this request to an appropriate agent or send back an address of the needed agent. The overall architecture of our prototype is shown in Figure 1, in terms of its agents. Brief descriptions of the functionalities of each of agent are given below.

• **Management Agent:** Manages data sources in system. It provides an interface for system managers to add, update or delete data sources.

[2] http://www.mcc.com/projects/infosleuth

[3] http://drogo.cselt.stet.it/fipa/spec/fipa97.htm

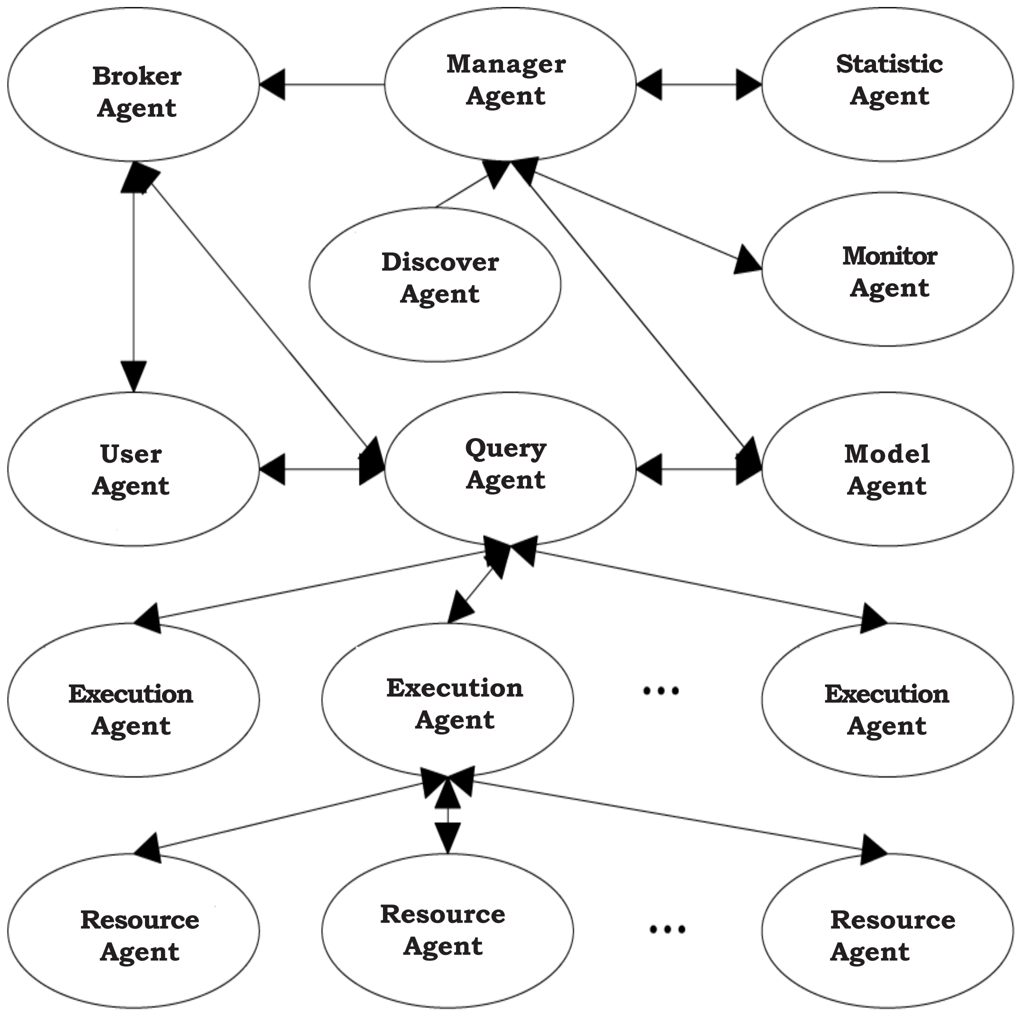

Figure 1. The presented architecture

• **Monitor Agent:** Monitors all known data sources. If a data source is updated or inaccessible, monitor agent will inform the management agent updating or deleting it.

• **Discovery Agent**: Discovers new data sources. Traditional www search engines navigate the overall network starting by some seed information sources. The similar way can be used to discover new data sources. Unfortunately, we can not decide whether a data source provide the SPARQL endpoint when we enter into it by looking up IRIs. In the current version of the system, we periodically access the raw data of Linked Data cloud via the CKAN API[4].

• **Statistic Agent:** Generates the statistic model for data sources. When a new data source is detected, it collects the relevant information about this data source, such as the SPARQL endpoint and the statistical model, and transforms them to the model agent.

• **Model Agent:** Maintains the local statistic model store.

• **Broker Agent:** Routes requests of other agents. It is the central agent in the architecture.

• **User Agent:** Receives user queries. It verifies user queries, transfers them to query agent and shows query answers.

[4]http://docs.ckan.org/en/latest/api.html

• **Query Agent:** Decomposes original queries into some sub-queries, makes query plans and integrates the intermediate results.

• **Execution Agent:** Executes query plans. It interacts with resource agents for executing join operations.

• **Resource Agent:** Retrieves data from remote data sources.

3.2 Sample Scenarios

In the following, two sample scenarios are used to demonstrate how agents in the presented architecture interact with each other.

During system start-up, all other agents register their addresses and capabilities in the broker agent at a wellknown address. The model agent loads statistic models and IP addresses of SPARQL endpoints of all known data sources. System managers can manually add or delete data sources through the interface provided by the management agent.Besides, the management agent maintains a list of messages received from the discovery agent. System managers decide whether a new data source needs to be added into the system. When a new data source is selected, the management agent triggers the statistic agent which constructs the statistic model of the data source. A completed statistic model is transferred to the model agent, and then is stored into a local model store.

A user submits a query by the user interface provided by the user agent. After being verified, the query is sent to the query agent. Then, the original query is decomposed into some sub-queries according to the information provided by the model agent. Each of decomposition triggers an execution agent. According to the statistic information, an optimal way of executing join operations is selected. Execution agents distribute sub-queries to their respective relevant resource agents which retrieve data from remote data sources and interact with other resource agents for join operations. All intermediate results returned from all execution agents are integrated into the final query answers by the query agent. The user agent shows query answers according to the format selected by the user.

4. Agent Design and Implementation

We have implemented the presented architecture in a Linked Data Management System(LDMS[5]), which has ability of adding, updating and abandoning data sources and provides efficiently query service for end users. The agent-based system is developed by JADE[6], a well-established multi-agent system development framework, allowing us to put our main attention on the business layer of the system. For example, communications between agents are simplified to send ACL Messages (enclosed in the ACLMessage class of JADE). In this section, we describe the functionality, design rationale, and implementation of each of the LDMS agents.

4.1 User Agent

The User Agent is the user's intelligent interface to the LDMS network. It assists the user in committing queries, and in displaying the results of queries in a manner sensitive to the user's context.

After initialization, the User Agent advertises itself to the Broker Agent, so that other agents can find it based on its capabilities. It provides users with options of the displaying format of query results. When the query agent has obtained a result, it engages in an ACL "*conversation*" with the user agent, in which the results are incrementally returned and displayed.

The User Agent is autonomous and maintains the user's context between browser sessions. It supports concurrent ACL interactions with other agents. Hence, the User Agent does not suspend its activity while waiting for the result of one query to be returned. It is implemented via Java applets which provide a flexible, platform-independent, and context-sensitive user interface. The User Agent is capable of saving user preferences and displays different interfaces for different users according to user query histories.

[5]LDMS is only available as Java source code (eclipse project) from the SVN repository:
https://svn.code.sf.net/p/semwldms/code/LDMS/trunk

[6]http://jade.tilab.com/

4.2 Manager Agent

The Manager Agent is the system manager's interface to the set of available resource agents in LDMS. With the Manager Agent, the system manager can add, delete or update data sets used to evaluate user queries.

When the Discover Agent finds new data sources, it sends an ACL message to the Manager Agent which maintains a list of new found data sources. System managers decide to whether adding or omitting a data source. If a new data source is selected to add to LDMS, the Manager Agent triggers the Statistic Agent which collects the relevant information about this data source, such as the SPARQL endpoint and the statistical class graph, and transforms them to the model agent. When the Monitor Agent detects a data source being changed or unavailable, it also sends a ACL message to the Manager Agent. In the window of the list of all resource agents existed in LDMS, different labels are used to label changed data sources and unavailable data sources. Then, system managers decide to update or delete a resource agent.

Same to the User Agent, the Manager Agent is implemented as a stand-alone java application. The interface is also provided via Java applets.

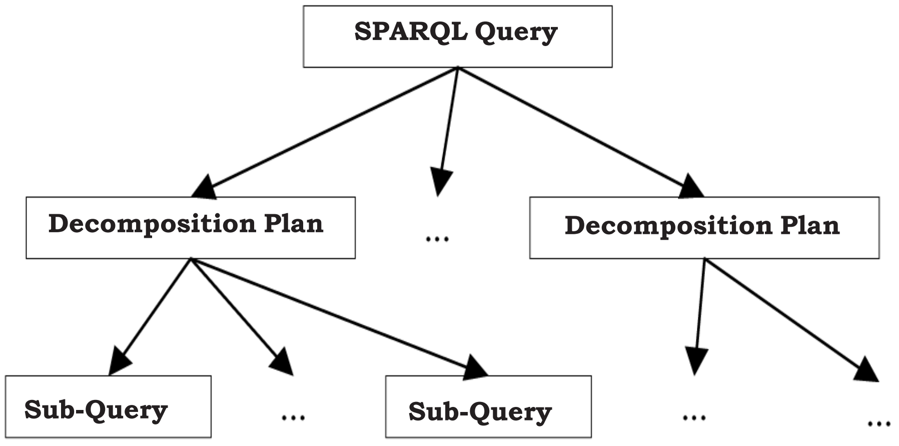

Figure 2. The hierarchy of query decomposition

4.3 Query Agent

The Query Agent has a familiar capability to traditional federated systems. It decompose a user query into some subqueries based on a local statistical model dataset maintained by the Model Agent; sub-queries are respectively sent to their relevant data sources, and then integrates the intermediate results into the final query answers.

The Query Agent is implemented in java based on Jena API. The parser shipped with ARQ is used to parse the original query string into some query components (e.g. BGPs, result vars, the limit to the number of query answers). Based on statistical models of known data sources, each triple pattern is decided to be relevant to some data sources. All triple patterns relevant to the same one data sources are used to construct a sub-query. Because a triple pattern may have more than one relevant data sources, decomposition plan may be more than one. The hierarchy of query decomposition is shown in Figure 2.

Each of decomposition plan is sent to a execute agent. The final query answers are the union of intermediate results returned by all Execute Agents.

4.4 Execute Agent

The Execute Agent coordinates the execution of information gathering tasks. It estimates the cardinality of all subqueries. The sub-query with the least number of intermediate results is evaluated firstly. To avoid cartesian product of results for two sub-queries, the second sub-query evaluated should join with the first query. That is, BGPs contained by these two sub-queries should share at least one variable. For the same reason, the third sub-query evaluated should join with the first sub-query or the second one, and so forth.

Join operations are executed in the Execute Agent. Currently, there are mainly three ways to execute join operations: nested-loop, pipe line and semijoin. Nested-loop needs download all intermediate results to the mediator. The nestedloop does not fit for coping with distributed join operations. Semijoin uses the way of value constraints to execute join operations over remote datasets. Pipelining join execution involves a great number of initiations of transaction (one for each solution of prior sub-query). LDMS uses group-join execution which restrains the number of the cached solutions in the mediator. In contrast to cache all solutions of the prior sub-query in Semi-Join, these solutions are divided into some groups, each group of solutions is used to construct value constraints of the next sub-query.

The Execute Agent evaluates sub-queries in parallel way. In a producer thread, the results of the first sub-query are fetched and written to a data stream one by one. A consumer thread monitors the data stream and takes away the previous results which are used to join with the results of the second query and start another producer thread.

4.5 Broker Agent

The Broker Agent is the core agent of LDMS. It pairs agents seeking a particular service with agents that can perform that service. The Broker Agent is running on a well-known address and firstly started. Other agents advertise their services to the broker agent via ACL. When an agent needs assistant of other agents, the Broker Agent responds to it with information about the other agents that have previously advertised relevant service. Due to the existence of the Broker Agent, other agents need not store information of all agents in the system, thus lower costs of network traffic required to accomplish an agent's task.

An agent advertisement minimally includes its location, name and capability. Additionally, agents may advertise some special information. For example, a resource agent need tell the Broker Agent the name of the data source it connected. The execute agent distributes sub-queries into resource agents according by these information.

4.6 Resource Agent

The Resource Agent retrieves data from remote data sources. It acts as a wrapper of heterogeneous data sources, providing a common interface, i.e. a SPARQL endpoint. Currently, LDMS is used integrating Linked Data, requiring data sources providing a query service.

The Execute Agent needs the help of the Resource Agent for answering SPARQL queries. The Statistic Agent obtains the statistical model of a remote data source through the Resource Agent. Hence, the most importance information the Resource Agent telling the Broker Agent is the address of its SPARQL endpoint.

Data set	Triples(k)	Model size(KB)	Time(m)
DBpedia	43.6M	13,126	235.3
NYTimes News	335k	103	1.9
LinkedMDB	6.15M	368	27.6
Jamendo	1.05M	33	5.2
Geo Names	108M	68	523.7
SW Dog Food	104k	646	0.5
KEGG	1.09M	42	5.5
Drugbank	767k	195	2.2
ChEBI	7.33M	23	25.2

Table 1. Benchmark Datasets

5. Experiments

In this section, we demonstrate the use of LDMS in data integration applications, and we present a data integration example from the life science application domain and the cross domain.

We use data from FedBench[7] [8] to simulate a real word application. FedBench includes two subsets of data sources in the

[7] FedBench can be downloaded at http://code.google.com/p/fbench/

Query	BGPs	Patterns	Answer size
Q 1.1	2	1,2	90
Q 1.2	1	3	1
Q 1.3	1	5	2
Q 1.4	1	5	1
Q 1.5	1	4	2
Q 1.6	1	4	11
Q 1.7	1	4	1
Q 2.1	2	1,1	1159
Q 2.2	2	1,2	333
Q 2.3	1	5	9054
Q 2.4	1	7	3
Q 2.5	1	7	393
Q 2.6	1	5	28

Table 2. Benchmark Queries

Linked Data cloud: cross-domain and life sciences. While the former comprises of six data sources, the later includes four data sources. For each data set, it defined seven queries. Answering anyone query needs access more than one data sources. For the limits of this paper on graph patterns,thirteen of fourteen queries are adopted in our experiments.The overview of the data sets is shown in Table I in terms of the number of triples and the size of statistical models and the time taken to create them. Queries are shown in Table 2 in terms of the number of BGPs and patterns in the WHERE clause and the answer size.

LDMS provides a general interface for accessing multiple data sources. Adding a new data source to the system simply entails adding a new Resource Agent, along with its statistical model. The brokering and query decomposition capabilities increase the efficiency of queries by committing local queries only to the databases that are likely to contain the matched data.

Query	Pipe line(s)	Nested-loop(s)	Semi-join(s)	Group-join(s)
Q 1.1	3.1	Time out	3.1	3.1
Q 1.2	0.1	0.2	0.1	0.1
Q 1.3	15.6	33.5	7.8	7.8
Q 1.4	1.6	40.2	1.5	1.5
Q 1.5	6.2	113.2	1.7	1.6
Q 1.6	117.2	Time out	128.7	33.0
Q 1.7	70.2	Time out	3.1	4.6
Q 2.1	11	213.5	7.9	10.8
Q 2.2	7.0	Time out	3.2	3.2
Q 2.3	Time out	Time out	Time out	553.3
Q 2.4	0.7	335.2	1.4	0.1
Q 2.5	Time out	Time out	173.6	93.9
Q 2.6	31.3	112.6	1.6	1.6

Table 3. The Comparison Of Returning Completed Answers

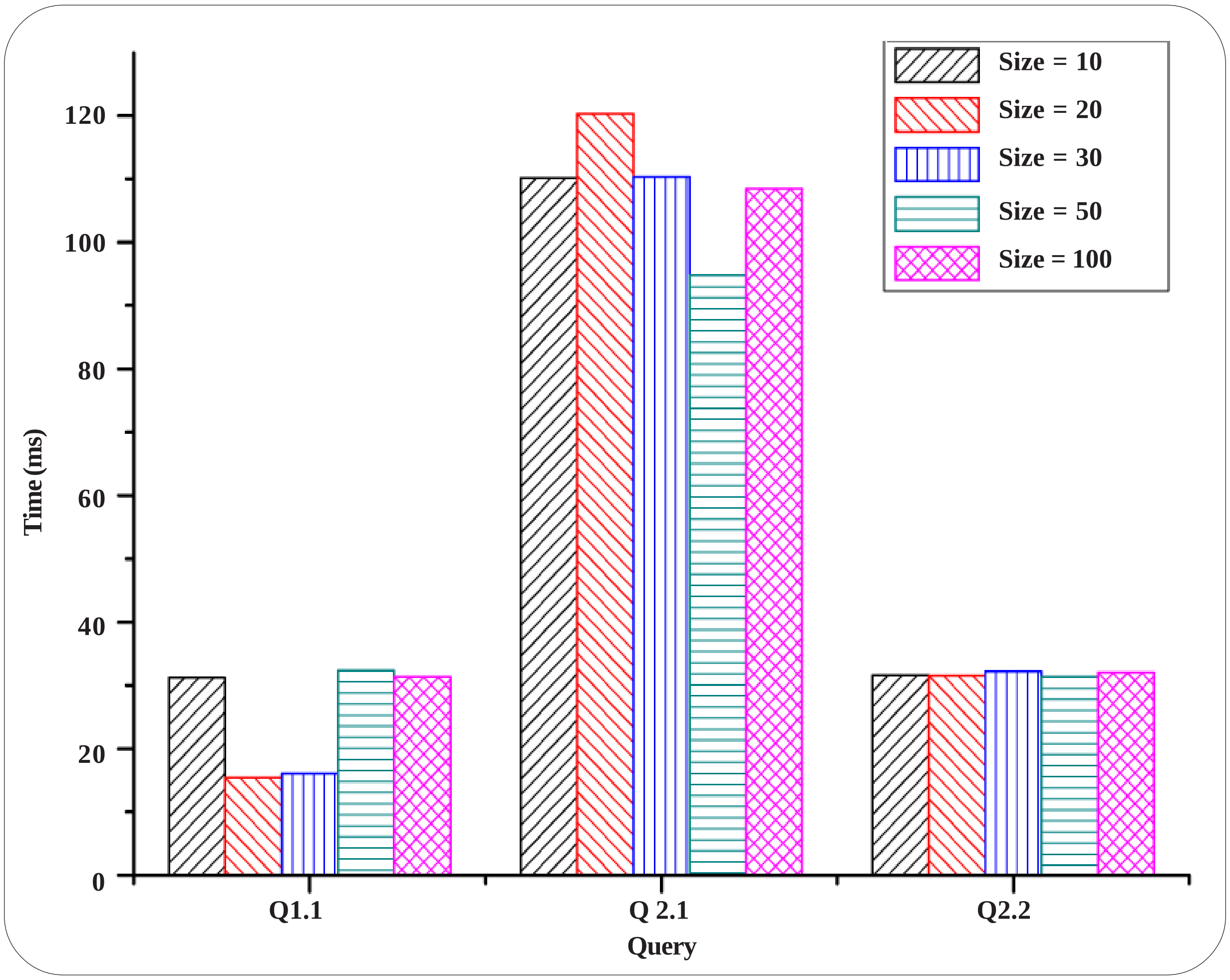

Figure 3. The Comparison of Returning Complete Answers with Different Group Sizes

5.1 Evaluation of Returning Completed Answers

Benchmark queries were respectively evaluated with four ways of execution of join operation: pipe line, Nested-loop, semi-join and group-join. For group-join, the size of group was set to 100. The value of time out was set to 10s. Table III shows the time for returning completed answers.

Due to all intermediate results being transformed over network, the time performance of materialization is in the worst situation. However, if all triple patterns in a query have good selectivity, it still can be comparable to other ways, i.e. Q 1.2. Group-join can always answer these queries in active time.

5.2 Different Group Size

We also report the time for returning complete answers when the size of group is set to other number. Q 1.1, Q 2.1 and Q 2.2 were randomly selected as the testing queries. The comparisons are illustrated in Figure 3. An interesting observation is that the time performance is insensitive to the group size for some queries like Q 2.2. When the group size being larger than the maximum size of intermediate result sets, Group-join will always be equivalent to Semi-join. In other words, different sizes of group can have influence on the time performance only if they can lead to different times accessing remote data sources or different amount of network transmission. How to find the optimal size of group for each query is in our future work.

6. Conclusion and Future Work

In this paper, we have presented an agent-based architecture and introduce the functionality of all agents. A Linked Data management system (LDMS) has implemented the presented architecture. LDMS is scalable and flexible. This is because of the use of collaborative agents, and the Java based development platform JADE. LDMS can be easily deployed in different environment. All agents are platform-independent. Besides, the user interface and the manager interface GUIs are written as Java applets, which can be executed from any browser on any platform.

The agent-based design of LDMS is well suited for managing data sources in a dynamic environment. Additionally, it integrates lined data sources while maintaining their local autonomy. We have demonstrated that the use of agent technology is a key element for system scalability and extensibility.

With the ever-increasing amount of data sources accessible via SPARQL endpoints, the linked data integration has attracted more and more attentions [9]–[11]. However, federated query systems for Linked Data are still in their infancy. Improving the query performance of these systems is always in the center of their work. We outline two key factors concerning the performance of federated query systems: Firstly, the decomposition of original queries must be accurate as far as possible. Secondly, distributed join operations should be effectively executed. We focus on improving the query performance of LDMS in our future work.

7. Acknowledgment

This work was supported by the National Natural Science Foundation of China (#61272361).

References

[1] Durfee, E. H., Kiskis, D. L., Birmingham, W. P. (1997). The agent architecture of the university of michigan digital library, *IEE Proceedings- Software*, 144 (1) 61–71.

[2] Dale, J. (1997). A mobile agent architecture for distributed information management, Ph.D. dissertation, University of Southampton. [Online]. Available: http://eprints.ecs.soton.ac.uk/00000849/05/thesis. ps.gz

[3] Bayardo Jr, R. J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., *et al*. (1997). Infosleuth: agent-based semantic integration of information in open and dynamic environments, in *ACM SIGMOD Record*, 26 (2) 195–206. ACM.

[4] Haas, L., Kossmann, D., Wimmers, E., Yang, J. (1997). Optimizing queries across diverse data sources.

[5] Shao, Y., Di, L., Bai, Y., Wang, H., Yang, C. (2013). Federated catalogue for discovering earth observation data, *Photogrammetrie-Fernerkundung- Geoinformation*, (1) 43–52.

[6] Langegger, A., Woß, W., Blochl, M. (2008). A semantic web middleware for virtual data integration on the web, in *The Semantic Web: Research and Applications*. Springer, p. 493–507.

[7] Quilitz, B., Leser, U. (2008). Querying distributed rdf data sources with sparql, in *The Semantic Web: Research and Applications*. Springer, p. 524–538.

[8] Schmidt, M., Gorlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T. (2011). Fedbench: a benchmark suite for federated semantic data query processing, *in The Semantic Web–ISWC*. Springer, p. 585–600.

[9] Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M. (2011). Fedx: optimization techniques for federated query processing on linked data, *in The Semantic Web–ISWC*. Springer, p. 601–616.

[10] Gorlitz, O., Staab, S., Splendid: Sparql endpoint federation exploiting void descriptions. *in COLD*.

[11] Hoefler, P., Granitzer, M., Sabol, V., Lindstaedt, S. (2013). Linked data query wizard: A tabular interface for the semantic web, *The Semantic Web: ESWC* 2013 *Satellite Events*. Springer, p. 173–177.