Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences 176 (2006) 1175–1189
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Performance evaluation of SDIAGENT, a multi-agent system for distributed fuzzy geospatial data conflation

Shahram Rahimi [a,*], Johan Bjursell [b], Marcin Paprzycki [c], Maria Cobb [d], Dia Ali [d]

[a] *Department of Computer Science, Southern Illinois University, Carbondale, IL 62901, USA*
[b] *School of Computational Sciences, George Mason University, Fairfax, VA 22015, USA*
[c] *Institute of Computer Science, SWPS, 03-815 Warsaw, Poland*
[d] *Department of Computer Science, University of Southern Mississippi, Hattiesburg, MS 39406, USA*

Abstract

A rapid growth of available geospatial data requires development of systems capable of autonomous data retrieval, integration and validation. Mobile agents may provide the suitable framework for developing such systems since this technology, in a natural way, can deal with the distributed heterogeneous nature of such data. In this paper, we evaluate SDIAGENT our, recently introduced, multi-agent architecture for geospatial data integration and conflation, and compare its model performance with that of client/server and single-agent approaches. Experimental results for several realistic scenarios, under varying conditions, are presented for these three system architectures. We analyze the performance alteration for various numbers of participating nodes, varying amount of database accesses, processing loads, and network loads.

* Corresponding author. Tel.: +1 618 453 6033; fax: +1 618 453 6044.
E-mail addresses: rahimi.@cs.siu.edu (S. Rahimi), cbjursel@gmu.edu (J. Bjursell), marcin@cs.okstate.edu (M. Paprzycki), maria.cobb@usm.edu (M. Cobb), dia.ali@usm.edu (D. Ali).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2005.07.009

Keywords: Agents; Multi-agent systems; SDIAGENT; Performance evaluation; Geospatial data; Conflation

1. Introduction

The aim of our current long-term research project is to develop an autonomous updating system that will be able to retrieve, filter, integrate, conflate and validate geospatial data from multiple heterogeneous sources, including Web-based repositories, into a single database system for subsequent access and retrieval.[1] Here, the geospatial data can be understood broadly to include vector, raster, text, pictorial, as well as multimedia. The sources of the input data are multiple governmental and private agencies with quality data. Autonomous updating subsumes several issues that must be resolved for a successful system implementation. Among these are integration of heterogeneous geospatial data formats and resolution of multiple representations (conflation), which are the most time consuming and complex tasks in this process.

At this stage of the project, we only consider vector-based geospatial data formats (but this is only a technical restriction of the GIS technology employed and does not affect the results presented in subsequent sections). Therefore, for the purpose of integration, the system converts different geospatial vector formats to a consistent object-oriented data format understandable for the system. Conflation is a higher-level concept than integration because it implies a deeper (semantic and intelligent) knowledge about the data. Conflation results in a state of agreement among various data sources in which a single, ''best'' view of multiple data representations for similar data types is presented to the user. Thus, conflation logically can occur only if integration as defined earlier has already been resolved. Integration and conflation are time consuming tasks that need significant amount of data manipulation and may generate heavy network traffic.

Automated conflation of maps is a complex process that must utilize work from a wide range of subjects and has presented several computational challenges for two decades. However, since the first successful implementation of a system based on geostatistical techniques nearly 20 years ago, very little progress has been made [5]. More recently, researchers have turned to fuzzy logic, rough sets and other methods for handling reasoning abilities under conditions of uncertainty to help solve general conflation problems [4]. These approaches

[1] Intelligent Database Agents for Geospatial Knowledge Integration and Management; BAA #NMA202-99-BAA-02 NIMA, Department of Defense; Research Area: Geospatial Information Sciences Concentration Area 4: Knowledge Development.

certainly show greater promise for producing a wider range of acceptable results; however, models for implementation have still been limited in their ability to provide a true, workable solution. Since the nature of the problem of integration and conflation is distributed, it requires a distributed model.

A client/server-based distributed model was introduced by the authors, however it turned out that in real-world applications it lacked a satisfactory performance [3]. This was due to the increase of the network traffic during the integration and conflation process. Moreover, the concentration of the conflation process in a central computer further adversely affected the performance of this model. The need for improving the performance led us to investigate the use of software agents for the implementation of such a system. In [9], we introduced SDIAGENT, an autonomous multi-agent-based application that retrieves, integrates and conflates geospatial data from multiple heterogeneous sources.

In this paper, we briefly describe SDIAGENT architecture and then concentrate on the performance of its conflation process. We compare the performance of our conflation model with the client/server approach. Moreover, we consider a single mobile agent-based conflation to illustrate the positive effect of dividing the tasks among multiple agents. We observe crucial parameters in the model that may affect the performance of the conflation process and compare the different approaches for a variety of settings. In this performance study, we show how our multi-agent architecture improves the conflation by reducing the size of the data transfer, distributing the actual process, and improving scalability of data manipulation and integration.

Let us note that this paper follows [8] and is in line with related works such as in [1,2,6,7,11]. In the later three articles, the issue of agent platform scalability is considered to be a very practical one, rather than a source of general reflection. Thus, we study performance empirically in the context of the real-world problem that we have to solve. Finally, let us note that a slightly different approach to analysis of the performance of the SDIAGENT can be found in [2]. There the details of the conflation process were disregarded and the conflation scenario (agents, their mobility and interactions) was used to show that it is possible to linearly scale system like SDIAGENT to more than 2000 agents implemented using JADE agent platform.

The remaining parts of this paper are organized as follows. First, we describe the architecture, as well as the parameters that are heavily relied upon by the model. Thereafter, the results for numerous tests are presented and discussed. Finally, we conclude the paper with a short summary.

2. Architecture overview

Fig. 1 depicts the general multi-agent architecture of SDIAGENT. Before proceeding to the conflation process, we list the types of agents (together with

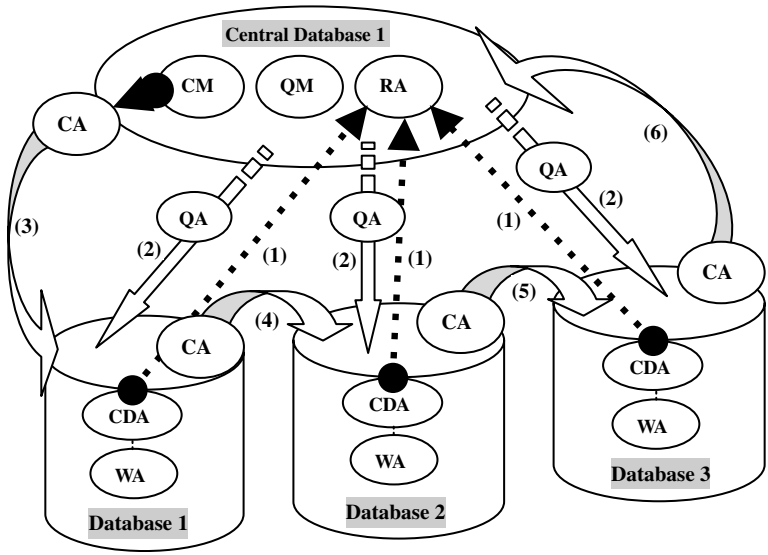

Fig. 1. The general multi-agent architecture of SDIAGENT.

a brief description of each) that currently are used in the system. Detailed information can be found in [9]. We begin with the agents that reside in the centralized database:

- ROI Agents (RA): Each RA is responsible for managing updates for a particular Region of Interest (ROI). For instance, the United Nation divides the world into 10 such regions. In our design, we follow this strategy. RAs are static and remain on the central database during the entire process.
- Queue Manager (QM): The QM is responsible for supervising a priority queue of updates generated by the RAs. The QM plays two roles. First, in case of an emergency, such as a natural disaster or political/military action, the agent assures that the pertinent information is promptly updated. Second, in the case of standard operational mode, it assures that all information sooner-or-later becomes updated (no starvation).
- Conflation Manager (CM): This is a static agent, which is located in the central database and is responsible for generating a conflation agent for each update request entry in the queue and initiating the conflation process.
- Conflation Agents (CA): Each CA, generated by the CM, is responsible for a single update request. The CA is a superclass of many specialized agent classes that have extensive knowledge about their domain relevant to the conflation process (Fig. 2). CAs are intelligent mobile agents, traveling to the data repositories (feeder databases) to perform conflation in a round robin fashion (described below).

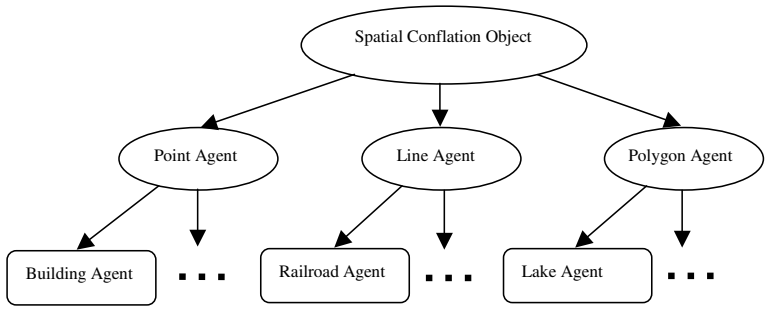

Fig. 2. The type of feature to be updated and its class determine the type of conflation agent to be generated.

- Query Agents (QA): These are released from the central database by the conflation agents (CA) to gather information for conflation-related queries. Prior to the CA, the QAs arrive in all pertinent databases (for each request, a sub-set of all feeder databases), perform initial queries and post the result to be used in the conflation process by the CA. The query agent cooperates with the wrapper agent (below) to translate the data into the common data format of the system.

Next, we list the agents that reside on the feeder databases:

- Change Detection Agents (CDA): These are relatively small and unintelligent agents that simply log changes to the database, query for necessary information and interact with the wrapper agents to create an update object for transfer to the region of interest agent (RA) on the central database.
- Wrapper Agents (WA): These agents are responsible for translating different geospatial data formats to the common data model of the system. At this time, we have two versions of WAs, for VPF (Vector Product Format-NIMA) and for ARCGIS Map data model.

We now describe a typical scenario that utilizes the above-described agents for autonomous management of geospatial data. Before we proceed, we will assume that at initial time the data in the feeder databases is almost synchronized with the central database. This means that only a very small amount of new information is available in the feeders. Suppose, from Fig. 1, that an update to an attribute value of a spatial feature is performed on database 1 (DB1). The resident CDA notes the change and queries the database for the information needed for an update object, namely, the database identifier, the feature ID and type information, bounding box coordinates, and type of update performed (metadata, attribute, topology, geometry or unknown). This

information is given to the WA to create the update object, which is then sent to the proper RA on the centralized database (flow 1, Fig. 1). At this moment, the WA can self-suspend, while the CDA returns to watch the database updates.

When the update object arrives at the central database, it is placed in a special data structure known as a ROI-tree. The ROI-tree is a quad-tree that contains predefined region objects represented by the RAs. For more information on the topic of spatial indexing, the reader is referred to [10]. Each RA manages updates for a particular area of the earth, represented by bounding box coordinates, and each RA has knowledge of all feeder databases that contain information within its domain. The update object is placed in the ROI-tree at a place under the domain of the RA for the region in which the update occurred. In our approach, as a default, we pursue 10 regional divisions of the world; however, other spatial arrangements are also possible. The hierarchical concept of sub-regions is also supported, and users can define their own regions of interest.

When the timing/frequency of updates is considered, users are allowed to set priorities for selected regions and sub-regions, or may use system-generated defaults. These specified priorities dictate the rate at which the gathered updates must be conflated and incorporated into the centralized database. As an example, an update for a low-priority region may be allowed to remain in the tree for several hours or even days, while high-priority region updates may be processed several times an hour, or even immediately upon arrival.

Whenever the priority scheme of the supervising RA determines that it is time to check for updates, a traversal of the sub-tree below the RA is performed, and all the update objects are placed in a conflation queue in order of their respective priorities. The QM is responsible for supervising this priority queue. The QM has two main responsibilities; first, for standard operation, it assures that all information sooner-or-later is processed (no starvation condition) while the priorities are observed. Second, in emergency cases, when a given region becomes a ''hot spot'', it will assure that the pertinent information is promptly updated by moving its request to the top of the queue.

The CM agent removes the highest priority update request from the queue and initiates the conflation process by generating an appropriate CA, based on the information in the update request object related to the feature type (point/line/area) and its class (e.g., building, railroad, vegetation). The CM agent also directly accesses the ROI-tree to retrieve other required information to complete the CA generation process. Now, the CA is ready to carry out the conflation process on the object. The CA uses its specialized knowledge to generate QAs to retrieve all available data needed for performing conflation on that object. The QAs are sent in parallel to all the feeder databases in which a potential conflict with the update exists (flow 2, Fig. 1), and begin the data extraction process. The QA communicates with the wrapper agent to translate the data

to the common data format of the system. After this step, the data in all the participating databases are ready for CA to start the conflation process.

The CA first moves to the database from which the update originated (flow 3, Fig. 1). The QA then passes the already-collected/prepared conflation data to it. The CA traverses all of the relevant databases, collecting the information and executing the knowledge-based conflation algorithm described in the following section (flow 4 and 5, Fig. 1). It assembles the conflated data in a Round Robin fashion and then brings the results back to the central database for updates (flow 6, Fig. 1). A detailed discussion of this process is presented in the following section.

We now briefly describe a typical conflation scenario. Conflation Agents are the most intelligent agents in this system. The type of the feature to be updated (point, line or area) and its class (i.e., railroad, highway, building) have a direct effect on the type of the CA to be generated (Fig. 2). For instance, if the feature is a point, a Point Conflation Agent (PCA), which contains a knowledge-based conflation algorithm for point features, will be generated.

Moreover, the type of the update performed on the feature is a significant factor in the conflation process. There may be five types of updates: metadata, attribute, topology, geometry, or some combination of these. In this system, we have a special CA for each one of these types. If just one of these updates occurs, then a CA for that particular type will be released. If more than one of these types of updates takes place then a team of CAs, one for each type, will be generated and launched. In the latter case, the combination of information is managed by the lowest-level common ancestor to all CAs spawned.

The customized CA travels to the feeder databases one by one, receives the already-collected conflation data from QAs, and places the data in a matching feature set. Each object in the matching feature set is given a similarity score. The similarity score is a weighted combination of various criteria, which represents the degree of similarity between an object in the matching feature set and the feature for which the conflation process is taking place. All the scores for different criteria range from 0 to 1, and the total similarity score is the weighted average of the scores. The features with a similarity score higher than a threshold will be considered for the conflation process.

The conflation algorithm design incorporates both fuzzy logic and a rule-based programming paradigm. *Feature matching*, one of the most substantive components of conflation, is a problem particularly well-suited to modeling through such human decision-making techniques. Rules based both on objective criteria, which incorporate constraint relaxation, and intelligent techniques based on subject matter expert input are part of the overall conflation processing scheme.

Since feature matching is an inexact process, we selected a fuzzy set paradigm for modeling the matched results. While a significant number of objective criteria are used to determine matches, the actual results are membership

values for fuzzy sets. For example, consider a railroad line feature that is a current candidate for matching with the original feature. The railroad line's spatial proximity to the original feature, its degree of matching on a feature type code, and equivalence of attribute sets and corresponding values all play a role in the determination of a match/non-match, which is presented as a fuzzy value.

The rule sets for conflation are based on a hierarchical classification of feature data. Because of our desire to keep the mobile agents' load as light as possible, we concentrate the bulk of the rules at the central database, and distribute with the mobile agents only the necessary "screening" rules to return potentially viable results. It is beyond the scope of this paper to detail the actual conflation process; however, it is important to note that the length of the conflation process varies for different cases. For more information please refer to [3].

3. Performance evaluation

Before reviewing the results, we briefly describe the client/server and the single-agent approaches. For the client/server system we use RMI and multithreading where each thread is responsible for the queries of a single database server. The client initiates the conflation process by accessing the servers and requesting data. After receiving the query results from every server, the conflation process takes place locally on the client computer (for more information on the details of this design, please refer to [3]). In the single-agent architecture, an agent is sent out from the central site to the data repositories with an itinerary of IP-addresses. On each node, the agent first queries the database and, thereafter, conflates the data. Subsequently, the agent proceeds to the next node along its route where the process is repeated. After completing its itinerary, the agent returns to the central site and reports the result.

All three approaches are implemented in Java. The two agent architectures are implemented using the Grasshopper Agent Platform distributed by the IKV^{++} Technologies AG. In our benchmark, the number of participating database servers is four unless otherwise specified (and the fifth computer is the client). The number of queries per database server fluctuates from 1 to 100, and the total size of the retrieved data ranges between 0 to 400 kilobytes. The data (on each server) is stored in Microsoft Access databases. We run our tests on an Ethernet LAN with 100 Mbps bandwidth.

The parameters we consider in our experiments are the number of queries per server, the size of the retrieved data, and the complexity of a conflation task. Different conflation tasks require varying numbers of queries per database; thus, studying the performance as a function of queries is crucial since the client/server approach accesses the databases remotely, while the agents interact locally with the databases. Similar reasoning motivates us to consider the size of

the retrieved data from a query. Furthermore, the impact on the performance based on the complexity of the actual conflation process is analyzed. Since the intricacy of a conflation task relies on the category of features as well as type of update involved, the length of the conflation process varies significantly. To model such settings, we introduce conflation units, where a unit has a fixed processing time, and run scenarios with a varying number of conflation units. Therefore for the purpose of performance evaluation, we measure the total *conflation time* for various settings. Let us stress that in the experiments performed for the purpose of this paper no actual conflation took place. Instead, we have selected a number of parameters that were used to model variety of conflation scenarios.

Properties listed above are first observed exclusively, i.e., we set the remaining parameters as constants. However, since the parameters are likely to impact each other differently depending on particular conflation tasks, comparisons where only one parameter is variable do not offer as much information as necessary to thoroughly evaluate the three approaches. Therefore, to obtain a better overview of how the performance is affected when altering more than one variable, we include three-dimensional plots. Finally, speedup charts depict the performance of the agent architectures compared to the client/server approach.

In Figs. 3–5 we evaluate the conflation process as a function of one variable while keeping the remaining parameters constant and set to such values that will not dominate the performance. Fig. 3 presents the timing results for an

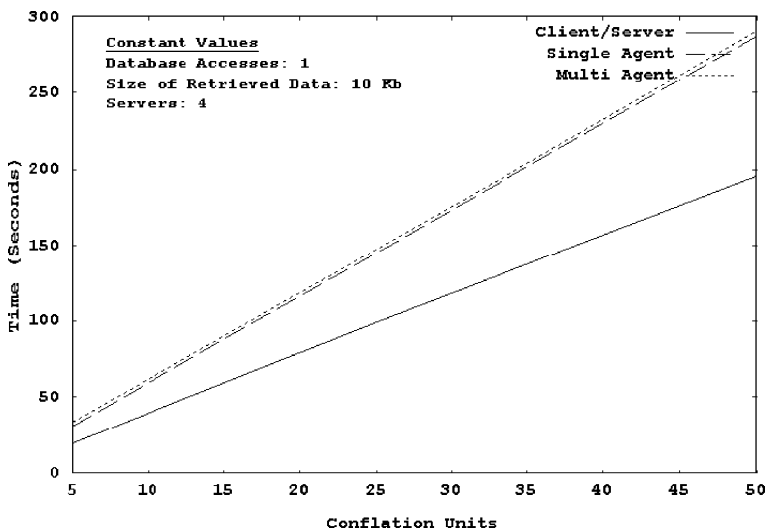

Fig. 3. Performance as a function of the number of conflation units.

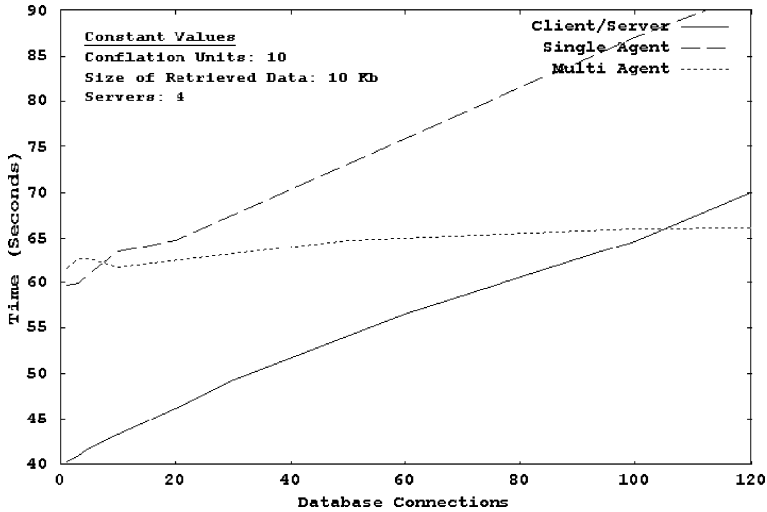

Fig. 4. Performance as a function of the number of database connections.

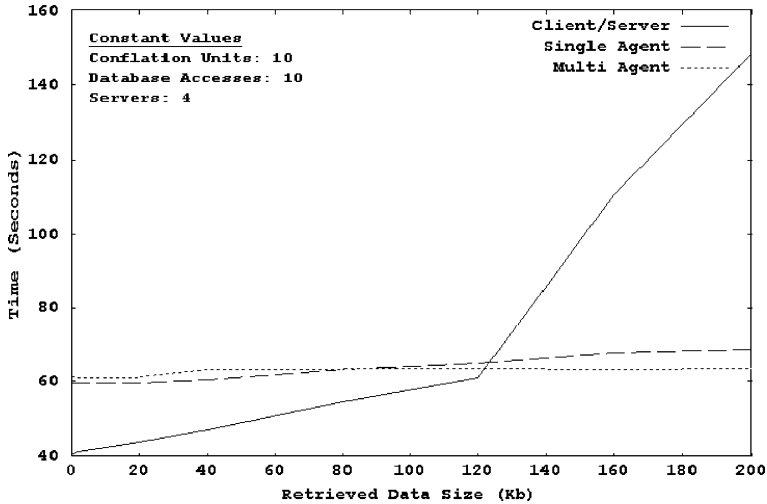

Fig. 5. Performance as a function of the size of retrieved data per query.

increasing number of conflation units. A single query is executed on each of the four databases, and the size of the retrieved data is fixed to 10 kilobytes. Consequently, only a small amount of data is conveyed over the network. As a result, the gain of local communication in the agent architectures is

minimal, and the overhead of code mobility slows down their overall performance. Furthermore, since only one query is executed per database, the overhead of utilizing query agents causes the multi-agent approach to be slightly less efficient than the single-agent approach.

Next we vary the number of queries per database (Fig. 4); 10 conflation units are executed on each server, and 10 kilobytes of data is returned per query. Since in SDIAGENT, query agents (QAs) are sent out prior to the conflation agent (CA), the conflation agent is able to start processing nearly instantly upon its arrival; thus, the processing time is almost constant. The slight increase in time originates from the situation when the CA arrives to the first node where it may have to stay idle waiting for the QA to complete its task. However, as the CA arrives to subsequent nodes, the querying is concluded (or just about to be concluded), allowing the CA to immediately process the information. Conversely, in the single-agent solution, the same agent is responsible for both querying and conflating data. Hence, the single-agent's processing time increases linearly with the number of queries. For the same reason, the client/server approach increases by the same rate as the single-agent solution.

In Fig. 5, we depict the situation when we incrementally increase the size of the retrieved data per each query from 0 to 200 kilobytes. Again, 10 conflation units are processed per server; and each server is queried 10 times. Both agent architectures stay essentially constant by processing the data locally on the data servers, in contrast to the client/server application, which transfers the data over the network and processes the data on the client. As discussed previously, compared to the single-agent solution, SDIAGENT's approach does not gain much from sending out the query agents, since there are only 10 queries per server; the gain is subdued by the additional overhead.

So far, we have seen how the number of database connections and the size of the retrieved data, independently of each other, impact the performance. Figs. 6–8 combine these properties by plotting the total processing times for the three architectures with the number of queries executed on each server on the x-axis and the size of the retrieved data retrieved for each query on the y-axis.

For small values of the parameters, i.e., for 10 queries or fewer and where the retrieved data size is less than 50 kilobytes per query, all architectures perform similarly. However, apart from this special (and for all practical purposes not very interesting) case, SDIAGENT performs significantly better, as the processing times for the two other architectures increase considerably faster. The single-agent application loses in efficiency to SDIAGENT when the number of queries increases since, in the single-agent approach, only one agent performs the querying and conflation sequentially. In the case of the client/server approach, the remote communication becomes the liability factor as the size of the data increases. This is even further intensified as the number of database accesses grows. While the client/server quickly reaches timings over 100 s,

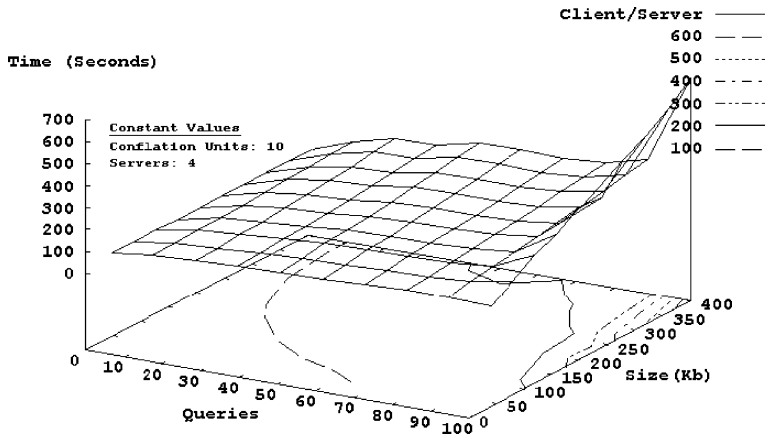

Fig. 6. Performance of the client/server architecture as a function of the number of queries and the size of retrieved data.

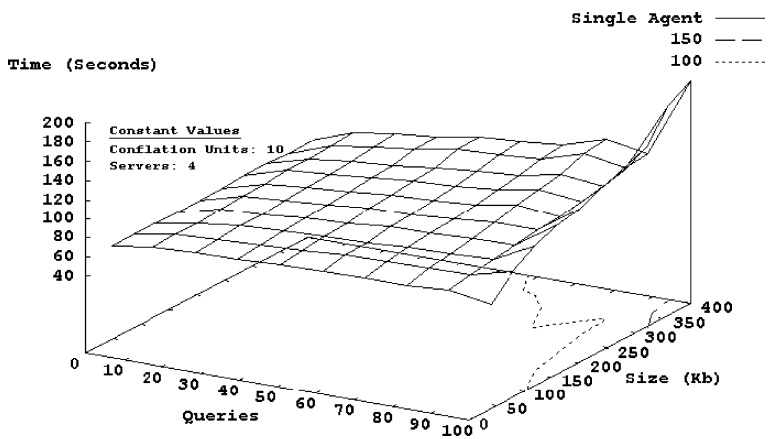

Fig. 7. Performance of the single-agent architecture as a function of the number of queries and the size of retrieved data.

SDIAGENT stays below 90 s even for 100 database hits and a retrieved data of size 400 kilobytes. Moreover, the single-agent architecture is significantly faster than the client/server, yet substantially slower than SDIAGENT.

Figs. 9–11 show the speedup of the agent architectures compared to the client/server approach as a function of the size of retrieved data. We ran the tests for 5, 50, and 100 for database accesses and processed 10 conflation units on

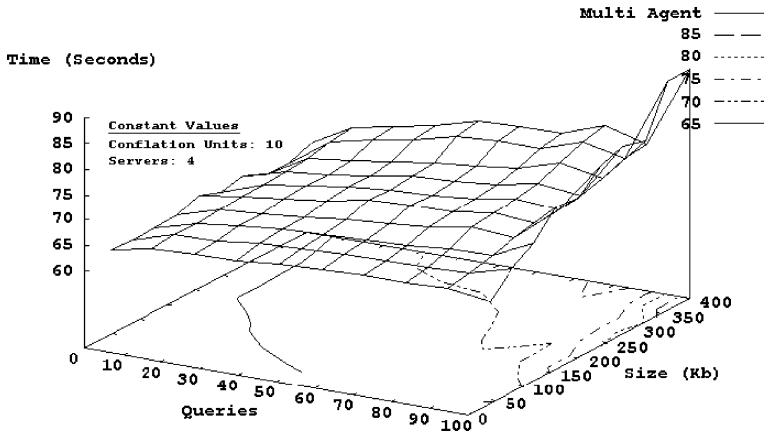

Fig. 8. Performance of SDIAGENT as a function of the number of queries and the size of retrieved data.

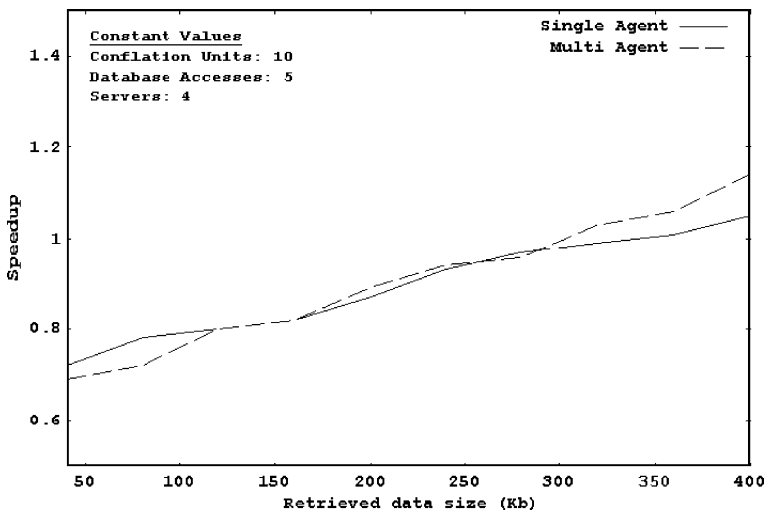

Fig. 9. The speedup of the agent approaches compared to the client/server approach as a function of the size of retrieved data with 5 database accesses per server.

each server. For five database accesses, the client/server outperformed the agent architectures for query results of size less than about 300 kilobytes. However, a larger amount of retrieved data and more database accesses favored the agent approaches, and especially the SDIAGENT system.

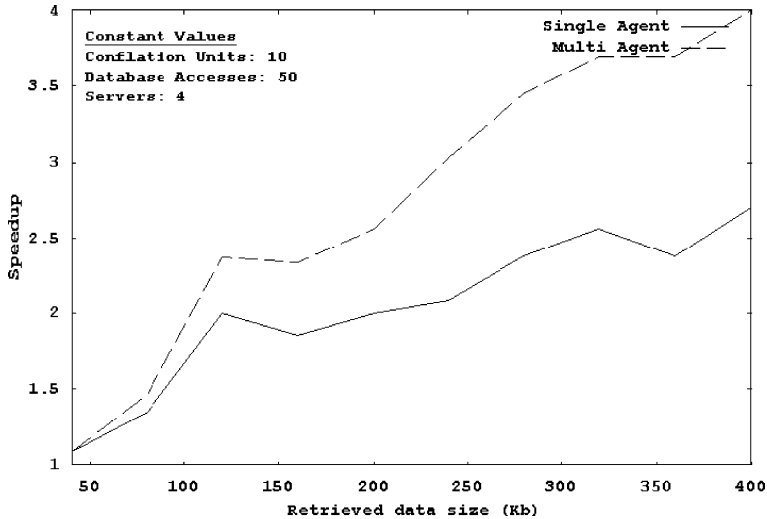

Fig. 10. The speedup of the agent approaches compared to the client/server approach as a function of the size of retrieved data with 50 database accesses per server.

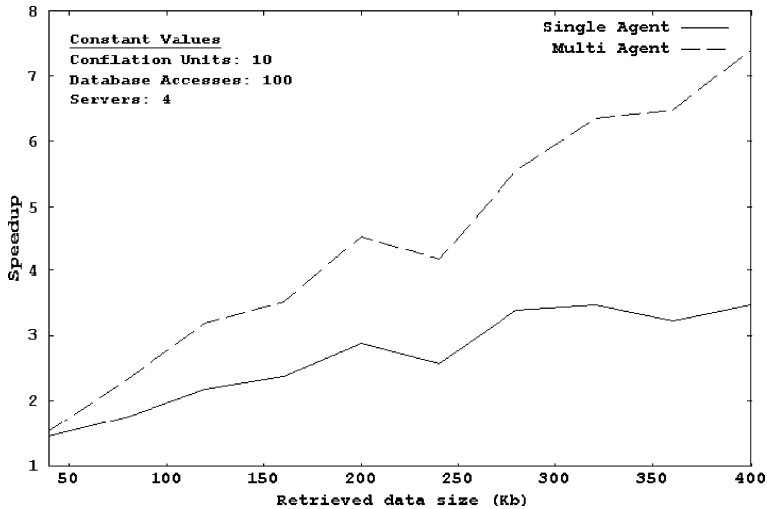

Fig. 11. The speedup of the agent approaches compared to the client/server approach as a function of the size of retrieved data with 100 database accesses per server.

4. Conclusion

In this paper, we have studied performance characteristics of the SDI-AGENT, a new multi-agent-based geospatial data conflation system, and

compared it with a client/server-based and a single-agent-based approaches. Only for a very small amount of data, the client/server performs better. This is due to the overhead of managing the agents, as well as the time of transferring the agents with their data state, knowledge base, and functionality from one node to another. In all the remaining (and practically interesting) cases the SDIAGENT demonstrates a substantial advantage over the two other system architectures, especially when the total amount of computations and data transfers increases significantly (which is what is to be expected in realistic applications). From the results presented, we have demonstrated how our multi-agent architecture can decrease the network traffic, divide the tasks efficiently and, thus, increase the performance of the system.

References

[1] K. Chmiel, D. Tomiak, M. Gawiniecki, P. Kaczmarek, M. Szymczak, M. Paprzycki, Testing the efficiency of JADE agent platform, in: Proceedings of the ISPDC 2004 Conference, IEEE Computer Society Press, Los Alamitos, CA, 2004, pp. 49–57.
[2] K. Chmiel, M. Gawiniecki, P. Kaczmarek, M. Szymczak, M. Paprzycki, Efficiency of JADE agent platform, Scientific Programming, in press.
[3] M. Cobb, F. Petry, K. Shaw, Uncertainty issues of conflation in a distributed environment, in: Proceedings of the GIS/LIS, 1998.
[4] M. Cobb, M. Chung, V. Miller, H. Foley, F. Petry, A rule-based approach for the conflation of attributed vector data, GeoInformatica 2 (1998) 7–35.
[5] M. Cobb, F. Petry, Modeling spatial relationships within a fuzzy framework, Journal of the American Society for Information Science 49 (1998) 253–266.
[6] L. Ismail, D. Hagimont, A performance evaluation of the mobile agent paradigm, ACM SIGPLAN Notices, 1999.
[7] S. Rahimi, J. Bjursell, R. Angryk, M. Paprzycki, D. Ali, M. Cobb, M. Gibutowski, An evaluation of mobile agent environments for geospatial knowledge integration and management system, in: S. Niwinski (Ed.), Proceedings of the PIONIER, 2001.
[8] S. Rahimi, J. Bjursell, D. Ali, M. Cobb, Preliminary performance evaluation of an agent-based geospatial data conflation system, in: Proceedings of the IEEE International Conference on Intelligent Agent Technology (IEEE-IAT 2003), Halifax, Canada, 2003, pp. 550–553.
[9] S. Rahimi, M. Cobb, D. Ali, M. Paprzycki, F. Petry, A knowledge-based multi-agent system for geospatial data conflation, Journal of Geographic Information and Decision Analysis, 1480-8943 6 (2002) 67–81.
[10] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, Reading, MA, 1989.
[11] G. Vitaglione, F. Quarta, E. Cortese, Scalability and performance of JADE message transport system, AAMAS Workshop on AgentCities, Bologna, 2002. Available from: <http://sharon.cselt.it/projects/jade/papers/Final-ScalPerfMessJADE.pdf>.