Preliminary Performance Evaluation of an Agent-Based Geospatial Data Conflation System

Shahram Rahimi
Dept of Computer Science
Southern Illinois University
Carbondale, IL, 62901
rahimi@cs.siu.edu

Johan Bjursell, Dia Ali, Maria Cobb
Dept of Computer Science
University of Southern Mississippi
Hattiesburg, MS 39406-5106
{firstname.lastname}@usm.edu

Marcin Paprzycki
Computer Science Dept
Oklahoma State University
Tulsa, OK 74109
marcin@main.amu.edu.pl

Abstract

A rapid growth of available geospatial data requires development of systems capable of autonomous data retrieval, integration and validation. Mobile agent technology may provide the suitable framework for developing such systems since this technology can deal, in a natural way, with the distributed heterogeneous nature of the data. In this paper, we evaluate our novel multi-agent architecture for geospatial data integration and compare its performance with a client/server and a single-agent architecture. We analyze the performance alteration for various numbers of participating nodes, amount of database accesses, processing loads, and network loads.

1. Introduction

The aim of our current long-term research project is to develop an autonomous updating system that will 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]. 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.

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 data mining 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. More recently, researchers have turned to a fuzzy logic, rough sets and other methods for handling reasoning abilities under conditions of uncertainty to help solve general conflation problems. These approaches 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.

A client/server based distributed model was introduced by the authors, which lacked a satisfactory performance. 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 declined 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 [2], we introduced an autonomous multi-agent based application that retrieves, integrates and conflates geospatial data from multiple heterogeneous sources.

In this paper, we briefly describe our multi-agent 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 conflation process in our comparison to show the constructive effect of multi-agents' cooperation. We observe crucial parameters in the model that may affect the performance of

[1]Intelligent Database Agents for Geospatial Knowledge Integration and Management; BAA #NMA202-99-BAA-02 NIMA-DoD

0-7695-1931-8/03 $17.00 © 2003 IEEE

the conflation process, and compare the different approaches for a variety of settings. In this study, we show how our multi-agent architecture improves the conflation by reducing the size of the data transfer, distributing the actual process, improving the data mining and integration.

2. Architecture Overview

Figure 1 depicts the general multi-agent architecture of our system.

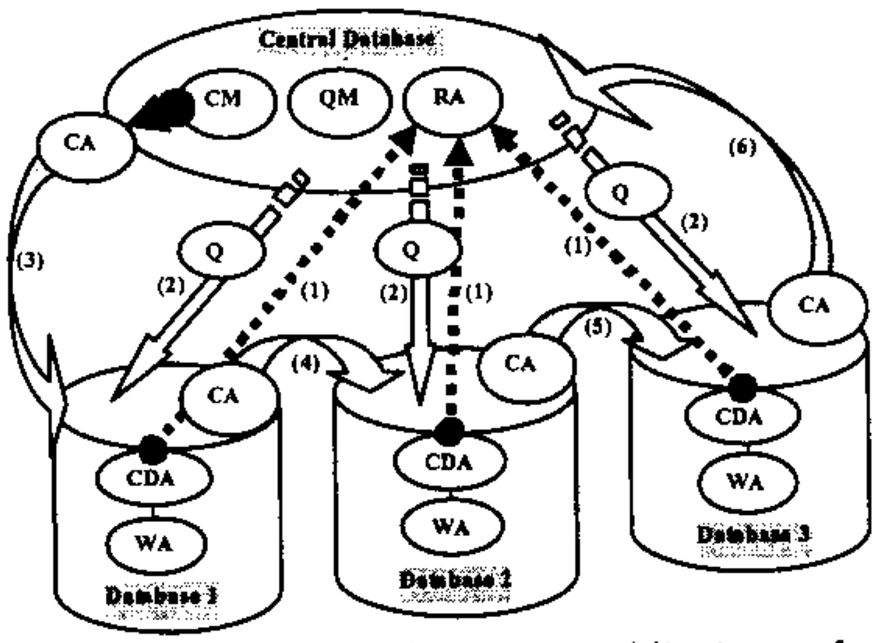

Figure 1: The general multi-agent architecture of our system.

Before proceeding to the conflation process, we list a few of the types of agents (together with a brief description of each) that currently are used in the system. Detailed information can be found in [2].

- 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, remaining on the central database during the entire process.
- Queue Manager (QM): The QM is supervising a priority queue of updates generated by the RAs.
- Conflation Manager (CM): This agent 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. CAs are intelligent mobile agents, traveling to the feeder databases to perform conflation in a round robin fashion (described below).
- Query Agents (QA): These are released from the central database by the conflation agents to gather information for conflation-related queries. Prior to the CA, the QAs arrive in all pertinent databases, perform initial queries and post the result to be used in the conflation process by the CA.

The customized CA travels to the feeder databases one by one and receives the already collected conflation data from QAs and places them 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. The features with a similarity score higher than a threshold will be considered for the conflation process. We skip the description of the actual conflation process, which takes place on the similar features. For more information please refer to [2].

3. Performance Evaluation

Before reviewing the results, we briefly describe the client/server and the single-agent approaches. Multi-threading and RMI are used for the client/server system 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 [2]). 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 itinerary where the process is repeated. After completing its itinerary, the agent returns to the central site and reports the result.

We run our tests on an Ethernet LAN with 100 Mbps bandwidth. The agent architectures are implemented with the Grasshopper Agent Platform distributed by IKV^{++} Technologies AG. In our benchmark, the number of participating database servers is four unless otherwise specified. The data is stored in Microsoft Access databases on each server.

For the performance evaluation, we measure the total conflation time for various settings. The parameters we consider in our measurements 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 number 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.

In figures 2 to 4, we evaluate the conflation process as a function of one variable while keeping the rest of the parameters constant to such values that will not dominate the performance. Figure 2 illustrates the timing results for an 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.

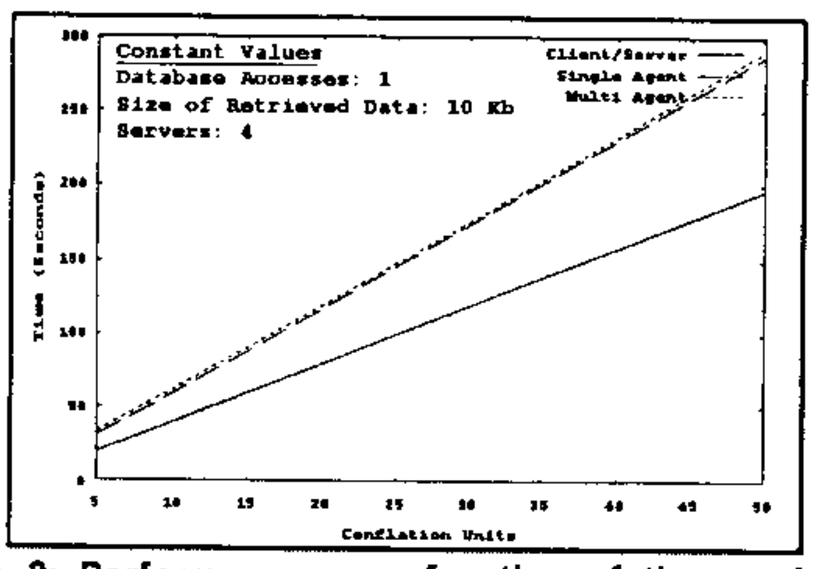

Figure 2: Performance as a function of the number of conflation units.

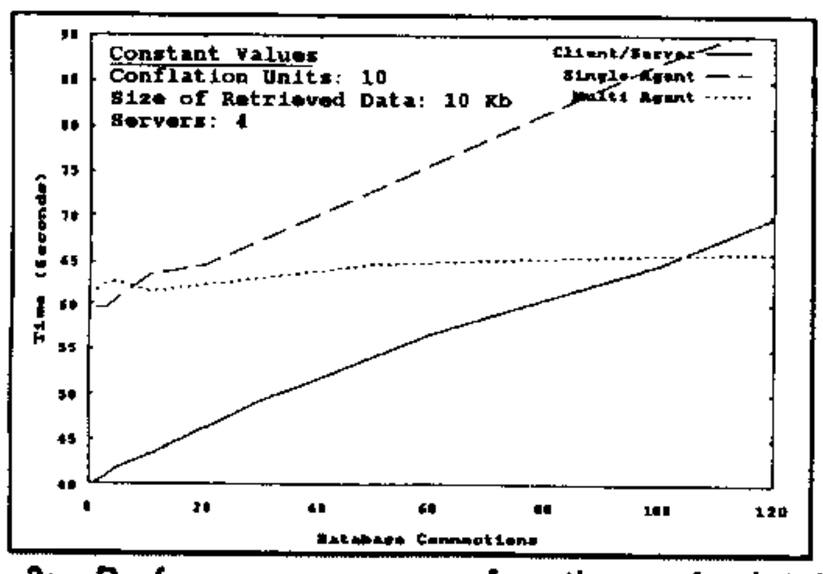

Figure 3: Performance as a function of database connections.

Next we vary the number of queries per database (figure 3); 10 conflation units is executed per server, and 10 kilobytes of data is returned per query. Since in the multi-agent approach, 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 when the CA arrives to the first node where the agent may stay idle waiting for the QA to complete its task. However, as the CA arrives to subsequent nodes, the querying is 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 figure 4, we increase the size of the retrieved data per each query from 0 to 200 kilobyte. 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 it on the central server. As discussed previously, compared to the single-agent solution, the multi-agent 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.

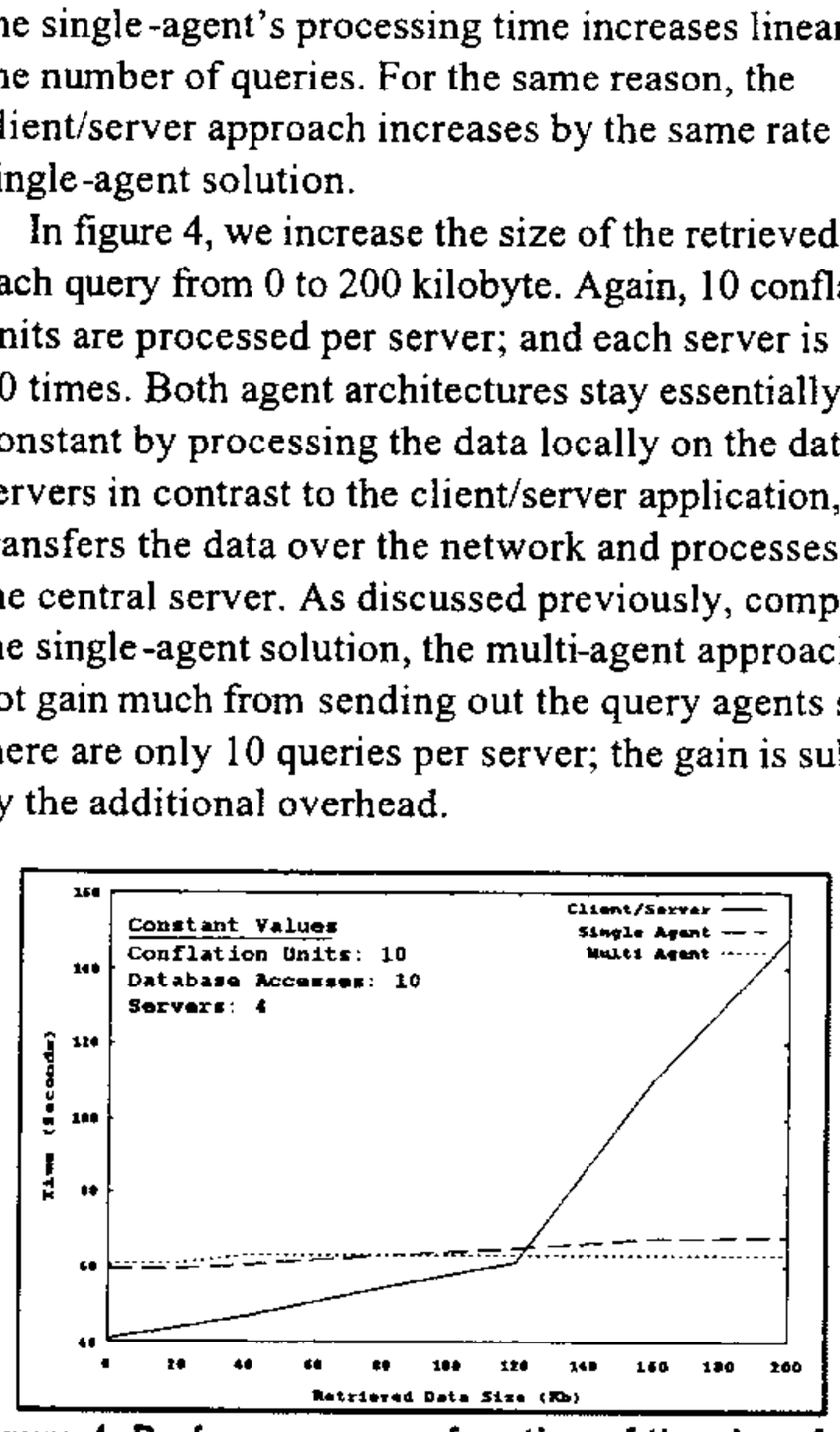

Figure 4: Performance as a function of the size of retrieved data per query.

So far, we have seen how the number of conflation units, the number of database connections and the size of the retrieved data independently impact the performance. Figures 5 to 7 combine these properties by plotting the total processing times for the three architectures with the number of queries on the x-axis and the size of the retrieved data on the y-axis.

For small values of the parameters, all architectures perform similarly. However, apart from there, the multi-agent system performs significantly better as the processing times increase considerably faster for the two other versions. The single-agent approach looses in efficiency to the multi-agent system when the number of

queries increases. For the client/server approach, the remote communication becomes the liable factor as the size of the data increases, which is even further intensified by a growing number of database accesses. While the client/server quickly reaches timings over 100 seconds, the multi-agent system stays below 90 seconds even for 100 database hits and retrieved data of size 400 kilobytes. Moreover, the single-agent architecture is significantly faster than the client/server, yet substantially slower than the multi-agent approach.

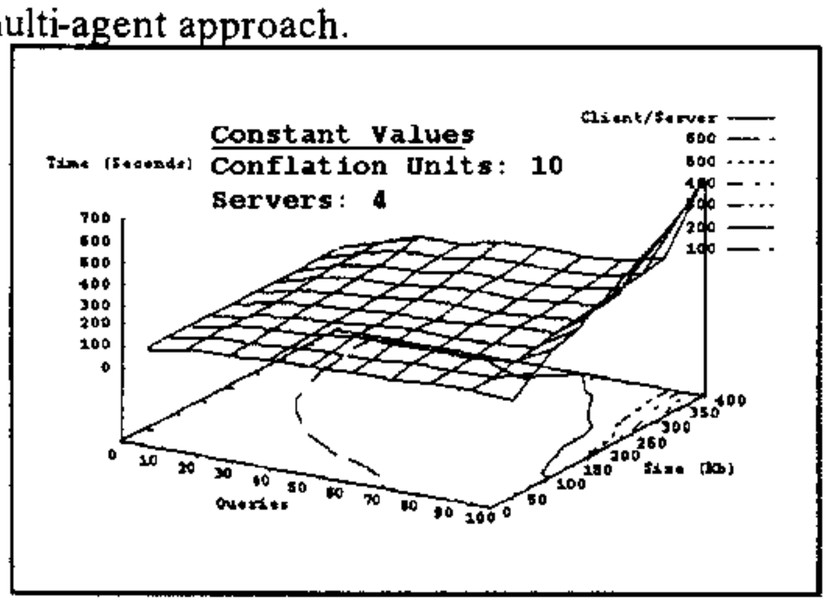

Figure 5: Performance of the client/server architecture as a function of the number of queries and the size of retrieved data.

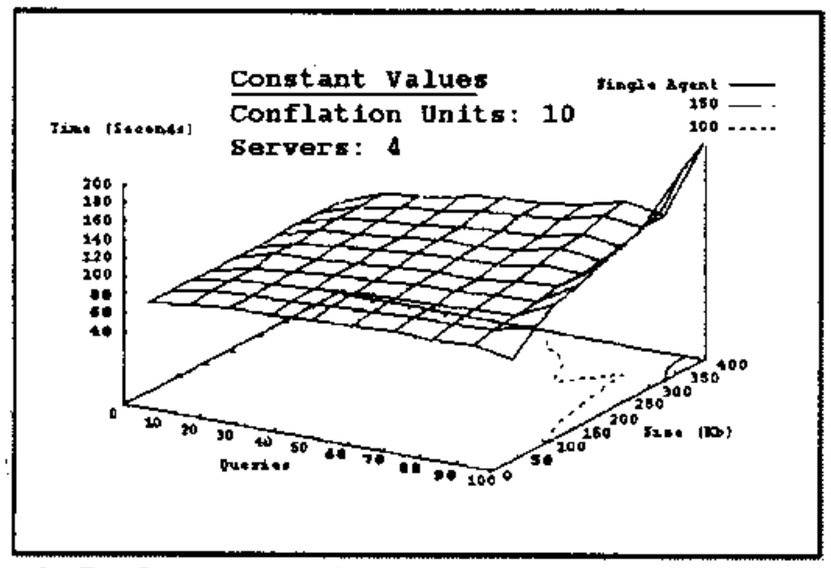

Figure 6: Performance of the single-agent architecture as a function of the number of queries and the size of retrieved data.

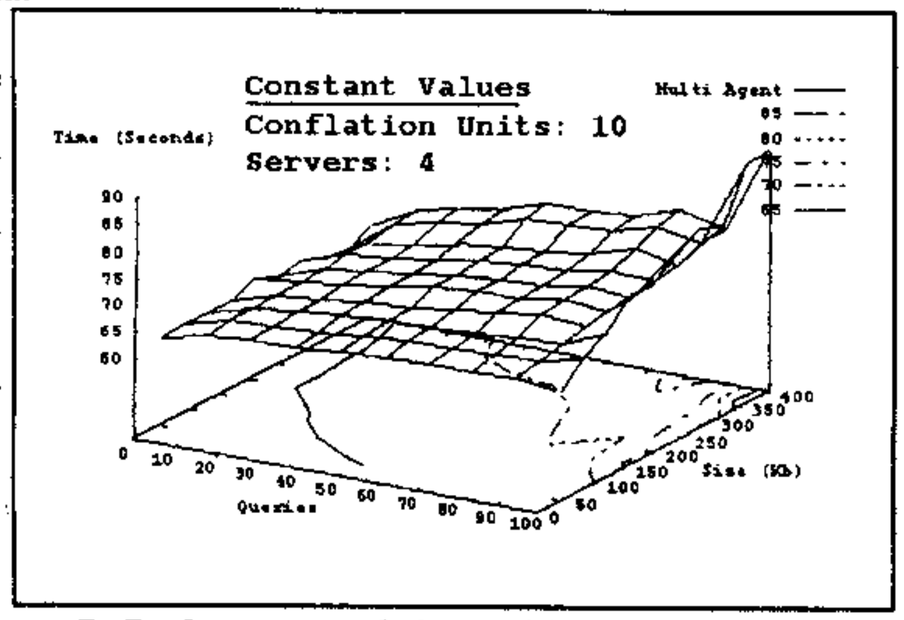

Figure 7: Performance of the multi-agent architecture as a function of the # of queries and the size of retrieved data.

Figure 8 shows the speedup of the agent architectures compared to the client/server approach as a function of the size of the retrieved data with 50 database accesses and process 10 conflation units per server. The agent approaches clearly outperform the client/server. For 5 database accesses, the client/server outdoes the agent architectures for query results of size less than 300 kilobytes; however, a larger amount of retrieved data and more database accesses favor the agent approaches, and especially the multi-agent architecture. For 100 accesses the agent approaches dominate even more.

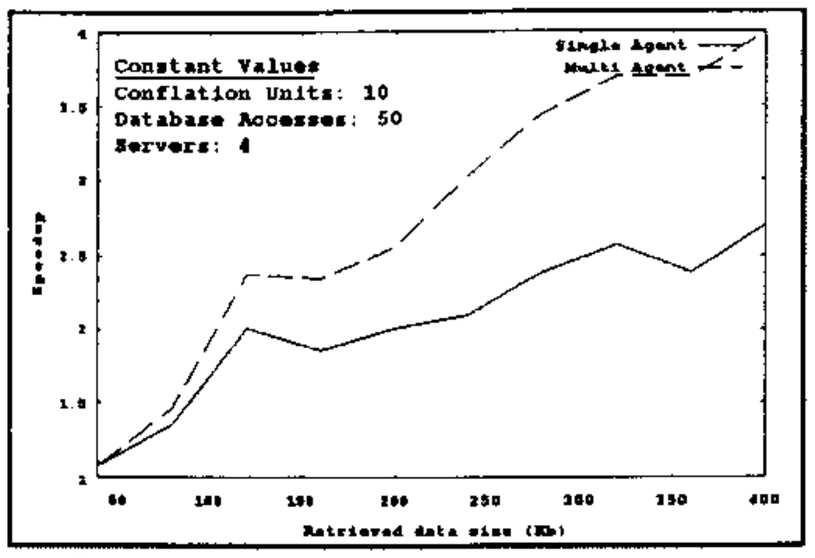

Figure 8: The speedup of the agent approaches compared to the client/server approach as a function of the size of retrieved data with 50 databases accesses per server.

4. Conclusion

In this paper, we benchmarked a new multi-agent-based geospatial data conflation system and compared it with a client/server based and a single-agent based approach. The multi-agent architecture substantiates great advantage over the two other approaches, especially when the capacity of the computation and data transfer increase. 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 such a system.

For a sufficiently small amount of geospatial 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 including their data state, knowledge base, and functionality from one node to another.

5. References

[1] M. Cobb and F. Petry, "Modeling Spatial Relationships within a Fuzzy Framework", *Journal of the American Society for Information Science*, 49(3), 253-266, 1998.

[2] S. Rahimi, M. Cobb, D. Ali, M. Paprzycki, and F. Petry, "A Knowledge-Based Multi-Agent System for Geospatial Data Conflation", *Journal of Geographic Information and Decision Analysis*, ISSN 1480-8943, pp. 67-81, Vol. 6, No. 2, 2002.