International Journal of the Physical Sciences Vol. 7(34), pp. 5297-5303, 6 September, 2012
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS12.246
ISSN 1992 - 1950 © 2012 Academic Journals

Full Length Research Paper

A multi-agent-based model for distributed system processing

N. V. Blamah and A. O. Adewumi*

School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, Durban, South Africa.

Accepted 16 July, 2012

Distributed systems have several challenges related to large datasets and numerous communication channels, and it is desirable to design systems that optimize the quality of services within the system and improve throughput to come up with the optimal result. In this paper, we present a model for multiagent system for distributed processing of tasks, where the processes continue to operate despite network disconnection and incomplete data sources' availability. The model provides a more reliable and cost-effective means of distributed processing.

Key words: Multiagent system, mobile agent, distributed system, databases, agent-based modeling.

INTRODUCTION

The spread of computers and handheld devices have brought the need to collaborate and cooperate among users in different spheres of life. As we continue to witness this trend, future networks will be dense and heterogeneous such that centralized systems design will be ineffective. This has made distributed and concurrent systems the norm of computing, and has given birth to different designs of distributed technologies where theoretical models (Wooldridge, 2009) are designed to portray computing as primarily a process of interaction.

Many businesses have come to rely totally upon the essential information asset stored in their corporate databases. If for any reason the information is not available in a timely fashion, the operations may come to a halt, and if it stays unavailable for a protracted period, serious financial consequences may follow. They therefore query their systems frequently as their businesses progress. Query processing is much more difficult in distributed environment than in centralized environments because a large number of parameters affect the performance of distributed queries. The distributed systems of enterprises are normally physically distributed across many sites by fragmenting and replicating the data, and may interact with hundreds of other systems outside the organizations. The systems select data from multiple sites in a network and perform computations on multiple CPUs to achieve a single result. It is therefore not uncommon for the communication requirements of the components to become highly intensive, with the consequences of overall poor systems performance.

It is quite evident that the performance of a distributed system is critically dependent on the ability to optimally device strategies that utilize available resources and minimize the cost of communication, data movement, and response time.

There have been several efforts (Blamah et al., 2008; Reza et al., 2008; Alaa et al., 2005; Pan et al., 2002) toward improving the quality of services provided within distributed system. This work is one of such efforts, where we present a model for multiagent system for distributed processing of tasks.

MULTIAGENT SYSTEMS

Multiagent systems are systems composed of multiple interacting computing elements, known as agents. Agents are computer systems that are at least to some extent capable of autonomous action – of deciding for themselves what they need to do in order to satisfy their

*Corresponding author. E-mail: adewumia@ukzn.ac.za.

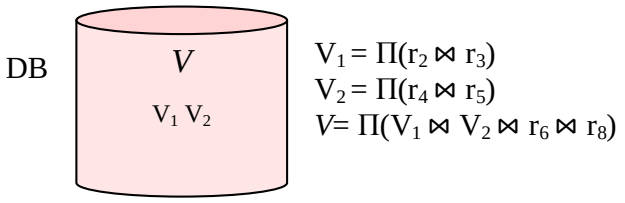

Figure 1. Database showing view definitions in an agent-based not self-maintainable re-computation materialized view.

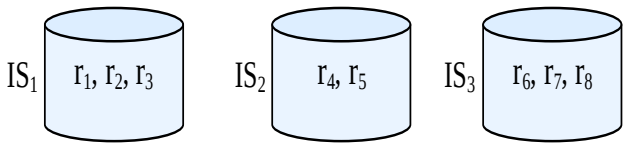

Figure 2. Relations from ISs in an agent-based not self-maintainable re-computation materialized view.

design objectives. They are also capable of interacting with other agents – not simply by exchanging data, but by engaging in analogues of the kind of social activity that we all engage in every day of our lives (cooperation, coordination, negotiation, and the like) in an intelligible manner. As an emerging sub-field of artificial intelligence, it is concerned with interaction of agents to solve a common problem (Wooldridge, 2002).

Agents are designed to be autonomous problem-solvers, possibly communicating with other agents and users, and are therefore equipped with sufficient cognitive abilities to reason about a domain, make certain types of decisions themselves, and perform the associated actions. This paradigm has become more and more important in many aspects of computer science by introducing the issues of distributed intelligence and interaction. They represent a new way of analyzing, designing, and implementing complex software systems. In multiagent systems, communication is the basis for interactions and social organizations which enables the agents to cooperate and coordinate their actions (Reza et al., 2008; Taghezout et al., 2009).

We have a number of factors which point to the appropriateness of an agent-based approach, e.g., environments that are open, or at least highly dynamic, uncertain, or complex; environments where the distribution of either data, control, or expertise means that a centralized solution is extremely difficult or even impossible. A typical example is distributed database systems in which each database is under separate control; they do not generally lend themselves to centralized solutions (Wooldridge, 2002) – so they may be conveniently modeled as multiagent systems, in which each database is a semi-autonomous component.

Literature review

Liu (2002) proposed two different optimization strategies which can greatly improve maintenance performance for a set of source updates in dynamic databases. The first strategy, the parallel maintainer, schedules multiple maintenance processes concurrently. The second strategy, the batch maintainer, groups multiple source updates and then maintains them within one maintenance process. This work is based on the TxnWrap model (Chen and Rundensteiner, 2000); the materialized views at the database must be maintained in response to actual relation updates in the remote sources. Many approaches to view maintenances have been proposed in literature (Liu et al., 2002; Liu, 2002; Connely and Begg, 2005; Chen, 2005; Idika, 2005), which are based on the client-server approach.

The real work of processing data and taking output from databases depends largely on how it is managed. Park et al. (2008) proposed a mobile agent platform for supporting mobile ad-hoc networks. The proposed algorithms provide agent service among mobile devices and route packets between agents. The implementation of the prototype of the algorithm was based on the Bluetooth protocol. Taghezout et al. (2009) proposed the integration of agents in a Cooperative Intelligent Decision Support System. The resulting system is designed to support operators during contingencies, where the operators will be able to gather information about the incident location, access databases related to the incident, activate predictive modeling programs, support analyses of the operator, and monitor the progress of the situation and action execution. The communication support enhanced communication and coordination capabilities of participants and a simple scenario was given to illustrate the feasibility of the design.

This paper is an effort made towards designing a more robust multiagent system for distributed processing of tasks.

FORMULATION OF MODEL

Let us consider a case of remote database (DB) being maintained by data from external information sources (ISs). Let the auxiliary view *V*, and primary materialized views V_1 and V_2 at the DB be defined by the relational algebra as presented in Figure 1. We then consider the remote relations on three different ISs as shown in Figure 2, which are the data sources for the views on DB.

Details of the relations within the remote ISs are omitted here for simplicity. The information sources exist as independent entities with no communication channels provided amongst them, but each of them can

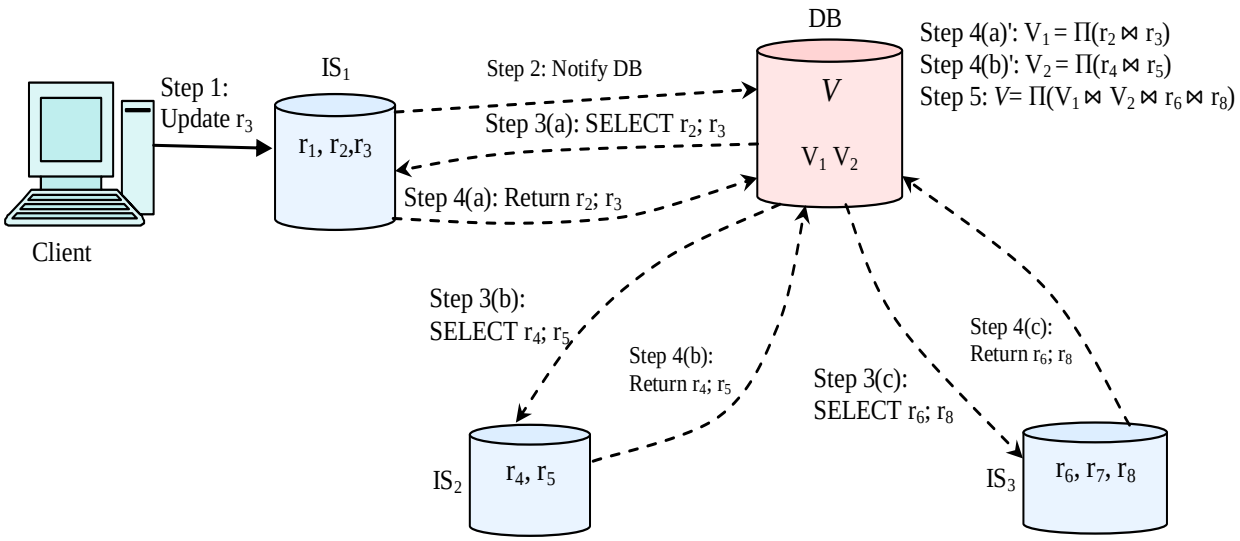

Figure 3. Database showing view updates in an agent-based not self-maintainable re-computation materialized view.

communicate with the DB.

If the views defined at the DB (V_1, V_2) and the remote relations are not self-maintainable, then *V* is also not self-maintainable. It means the DB has to send for queries to the ISs during update. We use Figure 3 to describe the update process; the solid line between the *Client* computer and IS_1 is an indication of network connection, which may be based on the client-server system, while the dashed lines are indications of network links, which are connected only when communication is about to take place.

If a *Client* application through an Update (static) Agent UA, modifies relation r_3 on IS_1, for example, which is equivalent to step 1, we have an Update (mobile) Agent called UAI, which initially resides on the ISs (IS_1 in this case), which informs the DB of such an update in step 2, and leaves the responsibility of doing the update to the DB.

The remainder of the update process continues as follows: The DB has the definition of all the views stored at it, and it also knows the locations of the remote relations. Since the process is re-computation, it means the views have to be computed from scratch. Also, the process is not self-maintainable, so the DB prepares the second Update (mobile) Agent called UAII, which initially resides on DB, connects to all the remote information sources and dispatches UAII in steps 3 (a) – (c) to get the update data, after which network is disengaged. These steps are executed independently but concurrently. If, for example, the DB is only able to connect to some ISs at a point, it goes ahead to dispatch UAII to the connected ISs so that UAII can start transactions on those systems. Whenever the DB connects to the other ISs, UAII are deployed independent of the previous ones and data is processed separately. This means data from different ISs can be processed at different periods, depending on the loads on each IS, and also on the availability of network connection between each IS and the DB. This is quite different from the implementations that require all information sources to be available at the same time and network connection to be stable throughout the update period.

Each IS that is done with data processing connects back to the DB in steps 4(a) to (c) and returns the result using UAII. These steps are executed in complete isolation from one another. When any result reaches the DB from an IS, the network between that IS and the DB maybe disconnected.

Step 4(a)' at the DB will be waiting for only step 4(a) to be executed, after which it executes to compute V_1. Step 4(b)' will also wait for only step 4(b), after which it executes to compute V_2. Step 5 is computed only after the execution of step 4(a)' (to produce V_1), step 4(b)' (to produce V_2,), and step 4(c) (to produce r_6 and r_8); this is because the definition of the primary view *V* in step 5 includes the results of steps 4(a)', 4(b)' and 4(c). The whole update process is thus completed when the primary materialized view *V* in step 5 is finally updated.

We define the environment (database) states where the agents operate as:

$E = \{e, e', \ldots\}$.

The agents have series of operations and actions at their disposals, which we represent by:

$Ac = \{\alpha, \alpha', \ldots\}$, where we assign:

α_0 = Step 1: Update r_3

α_1 = Step 2: Notify DB

α_2 = Step 3(a): SELECT r_2; r_3
α_5 = Step 3(b): SELECT r_4; r_5

α_3 = Step 4(a): Return r_2; r_3
α_6 = Step 4(b): Return r_4; r_5

α_4 = Step 4(a)': $V_1 = \Pi\ (r_2 \bowtie r_3)$
α_7 = Step 4(b)': $V_2 = \Pi(r_4 \bowtie r_5)$

α_8 = Step 3(c): SELECT r_6; r_8
α_{10} = Step 5: $V = \Pi\ (V_1 \bowtie V_2 \bowtie r_6 \bowtie r_8)$

α_9 = Step 4(c): Return r_6; r_8

The set of actions is represented in the following sequence so as to make the tasks disjointed and allow the various agents to handle the respective transactions independently:

$$\left.\begin{array}{l} Ac_1 = \{\alpha_0\} \\ Ac_2 = \{\alpha_1\} \\ Ac_3 = \{\alpha_2, \alpha_3, \alpha_4\} \\ Ac_4 = \{\alpha_5, \alpha_6, \alpha_7\} \\ Ac_5 = \{\alpha_8, \alpha_9\} \\ Ac_6 = \{\alpha_{10}\} \end{array}\right\} \quad (1)$$

A run is defined as $r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \ldots\ldots \xrightarrow{\alpha_{u-1}} e_u$

Therefore using the database states and the defined actions in equation (1), the runs will result to:

$$r_1 : e_{10} \xrightarrow{\alpha_0} e_{11}$$
$$r_2 : e_{20} \xrightarrow{\alpha_1} e_{21}$$
$$r_3 : e_{30} \xrightarrow{\alpha_2} e_{31} \xrightarrow{\alpha_3} e_{32} \xrightarrow{\alpha_4} e_{33}$$
$$r_4 : e_{40} \xrightarrow{\alpha_5} e_{41} \xrightarrow{\alpha_6} e_{42} \xrightarrow{\alpha_7} e_{43}$$
$$r_5 : e_{50} \xrightarrow{\alpha_8} e_{51} \xrightarrow{\alpha_9} e_{52}$$
$$r_6 : e_{60} \xrightarrow{\alpha_{10}} e_{61}$$

Therefore the set of runs is represented by

$$\mathcal{R} = \{r_1, r_2, r_3, r_4, r_5, r_6\} \quad (2)$$

A state transformer function for an update within the above setting, which represents the behavior of the environment as a result of an agent's action, is modeled as follows:

$\tau : \mathcal{R}^{AC} \rightarrow \wp(E)$; that is:

$$\tau_1(\mathcal{R}^{Ac_1}) = e_{11}$$

$$\tau_2(\mathcal{R}^{Ac_2}) = e_{21}$$

$$\tau_3(\mathcal{R}^{Ac_3}) = \begin{cases} e_{31} \\ e_{32} \\ e_{33} \end{cases}$$

$$\tau_4(\mathcal{R}^{Ac_4}) = \begin{cases} e_{41} \\ e_{42} \\ e_{43} \end{cases}$$

$$\tau_5(\mathcal{R}^{Ac_5}) = \begin{cases} e_{51} \\ e_{52} \end{cases}$$

$$\tau_6(\mathcal{R}^{Ac_6}) = e_{61}$$

The environment, which is a triple, is therefore defined as

$$Env = \langle E, e_0, \tau \rangle \quad (3)$$

Where E is set of environment states $e_0 \in E$ is initial state, and τ is state transformer function.

We define the model of an agent as:

$$Ag : \mathcal{R}^E \rightarrow Ac;$$

The agents UA, UAI and UAII are therefore defined and represented as:

UA=$Ag_0(\mathcal{R}^E) = \alpha_0$;

UAI=$Ag_1(\mathcal{R}^E) = \alpha_1$

and

UAII=$Ag_2(\mathcal{R}^E) = \begin{cases} \alpha_2 \\ \alpha_3 \\ : \\ : \\ \alpha_{10} \end{cases}$

The set of agent is represented as

$$Ag = \{Ag_0, Ag_1, Ag_2\} \quad (4)$$

By the definitions of runs, environments and agents as presented in equations (1), (2) and (3), the model of the system is represented as a pair containing an agent Ag, and environment Env, which is a set of possible runs

$$\mathcal{R}(Ag, Env) \quad (5)$$

SERVER DESIGN

We implemented the model of the system in equation (5) by designing servers, which are composed of a number of components.

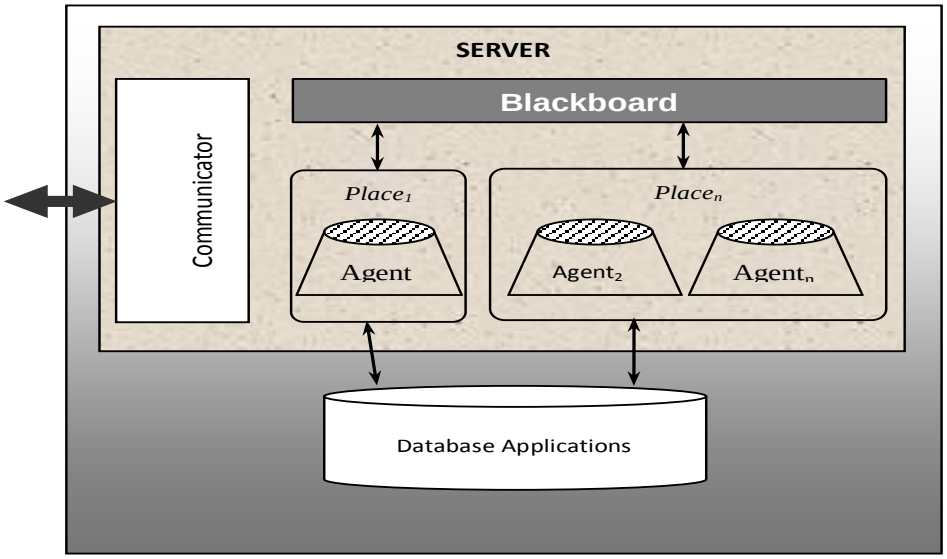

Figure 4. Server design.

The generalized server design is depicted in Figure 4. Agents execute within places on servers. The servers instantiate place(s) in order to execute agents, and they must also instantiate communicator(s) to enable them communicate with other entities. A *Server* provides blackboard where agents record information about their next destinations so that they can be traced. The servers were designed as simple class hierarchies and were installed and configured on each computer node.

The experimental environment used comprised of a local area network (LAN) with four computers; Table 1 outlines the specifications (and names) of each of the computers that was used to conduct the experiments, while the LAN comprised of a 16-port D-Link switch and UTP cat 5e cables in a star topology.

All the databases used for the purpose of the experiments were implemented using MySQL Server 5, while the mobile agents and all the relevant servers were implemented with the installed J2SE platform.

Experiment goal

The goal of this experiment is to measure the time taken to complete an update and see the effect of distorting network connections during the not self-maintainable re-computation process based on agent framework and compare the results with the none-agent based client-server system using the same setting. This is aimed at observing the reliability of the system.

Procedure

The experimental environment was set up to have the following relations defined at the ISs:
$IS_1 = IS_1$ (r_1: category; r_2: order; r_3: product)
$IS_2 = IS_2$ (r_4: company; r_5: orderDetails)
$IS_3 = IS_3$ (r_6: Customer; r_7: Time; r_8: Sales)

Agent-based/Client-server based transactions

The procedures are the same, except steps 3 and 4:

1. Start the Java Remote Method Invocation Register (rmiregister) on all the machines
2. Start the DB Server on the main database component and the Remote Systems Server on the remote systems components
3. Agent: Trigger an update on IS_1using UA, which is equivalent to step 1 in Figure 3; UAI at IS_1 is sent to inform the DB of such an update in step 2, and leaves the responsibility of doing the update to the DB.
Client-Server: Trigger an update on IS_1 and make remote invocations on the DB (from the IS) to notify the DB of such an update, this also leaves the remaining activities to the DB.
4. The DB has the definition of all the views stored at it, and it also knows the locations of the remote relations. Since the DB is not self-maintainable and the process is re-computation, it means the views had to be computed from scratch.
Agent based: Sent UAII to complete the update.
Client-Server based: Make Remote Invocation to complete the remaining update.
5. Wait until update completes.

RESULTS AND DISCUSSION

Fifteen different sets of observations were made by varying the number of tuples in the relations. In the course of conducting the experiment, it was observed that each time network connections were tampered with, the client-server model aborted execution and sent network error messages, and the process had to restart from the beginning. The agent based design also behaved in the same manner if the mobile agent was on transit in the course of network distortion. But after successful migrations of mobile agents, restarts were never required in the midst of network distortions and each of the ISs was independent of the others. This is quite different from the pure client-server implementation that required all information sources involved in the view definitions to be available and network connection to be

Table 1. Computer units specifications.

Computer name	Type	Processor speed (GHz)	HDD/RAM	Operating system/databases/software/network analyzer
Comp1	PM	1.8	80GB/512MB	Win XP SP2, MySQL Server 5, Java (J2SE 1.6 update 1), Performance Logs and Alerts Network Analyzer, *OperationalSystemServer, DataAccessServer, Mobile Agents*
Comp2	PIV	3.2	150 GB/1 GB	Win XP SP2, MySQL Server 5, Java (J2SE 1.6 update 1), Performance Logs and Alerts Network Analyzer, *OperationalSystemServer, DataAccessServer, Mobile Agents*
Comp3	PM	1.8	80 GB/ 512 MB	Win XP SP2, MySQL Server 5, Java (J2SE 1.6 update 1), Performance Logs and Alerts Network Analyzer, *DatabaseServer, Mobile Agents*
Comp4	PIV	3.2	150 GB/1GB	Win XP SP2, MySQL Server 5, Java (J2SE 1.6 update 1), Performance Logs and Alerts Network Analyzer, *OperationalSystemServer, DataFactoryServer,Mobile Agents*

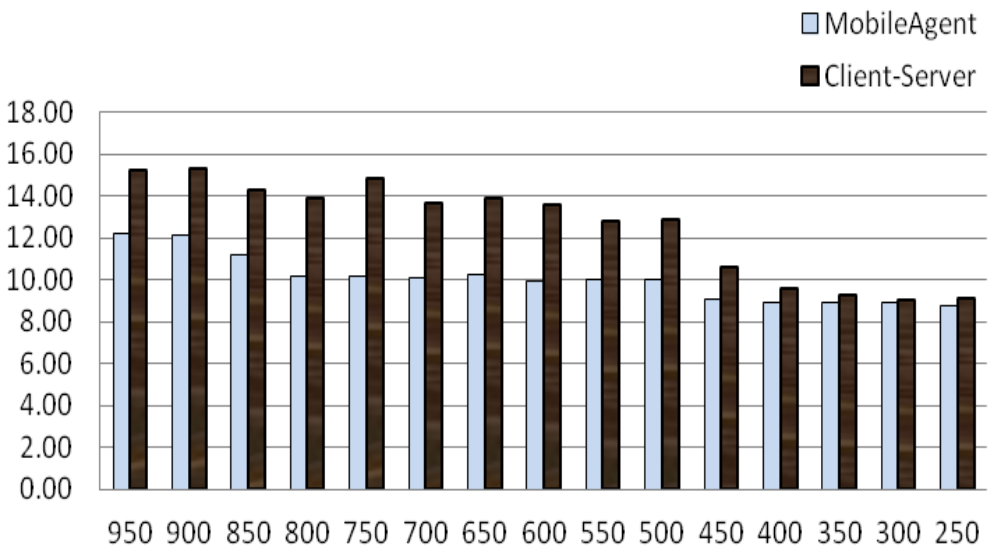

Figure 5. Performance measurement using time.

stable throughout the update period.

In all the observations that were made for the timing, when the update task was much, the agent design outperformed the client-server system; this is because of the high number of tuples involved, in which the effect of network delay contributed much to the time on the client-server system. The time required to complete updates as the number of tuples decreased tended towards being the same for the two designs, which was as a result of the lesser tasks, which did not place much of the effect of network slowdowns on the client-server model. This is shown in Figure 5.

CONCLUSION

Multiagent systems are designed to be composed of autonomous entities that solve problems, and they have the abilities to reason and take decisions themselves. In this paper, we present a multiagent based model for distributed systems management, which facilitates more robust systems operation. The system was designed as a hybrid of static and mobile agents, and it enables reduced requirement for network connectivity and provides an opportunity for data to be processed offline.

This improves on the systems availability; components

that would otherwise be idle because of lack of network connections would be utilized once the mobile agents successfully migrate to those systems. The model for multiagent system presented in this paper provides a more reliable and cost-effective means of distributed processing of tasks.

REFERENCES

Alaa A, Emad A, Mohammed O (2005). A Survey of Distributed Query Optimization. Int. Arab J. Inf. Technol. 2(1):48-57.

Blamah NV, Wajiga GM, Baha BY, Mu'azu HG (2008). A Mobile Agent-Based Data Warehouse Materialized View Maintenance. J. Institute Math. Comput. Sci. (Computer Science Series), India 19(2):189-206

Chen J, Rundensteiner EA (2000). Txnwrap: A transactional approach to data warehouse maintenance. Technical Report WPI-CS-TR-00-26, Worcester Polytechnic Institute, UK.

Chen S (2005). Efficient Incremental View Maintenance for Data Warehousing, PhD Dissertation Submitted to the Faculty of The Worcester Polytechnic Institute, UK.

Connely T, Begg C (2005). Database Systems, a Practical Approach to Design, Implementation and Management, 4th Edition, Addison Wesley.

Idika O (2005). Data Mining and Data Warehousing. A Publication of the Digital Bridge Institute, International Center for Advanced Communication Studies, Abuja, Nigeria.

Liu B (2002).Optimization Strategies for Data Warehouse Maintenance in Distributed Environments, MSc Thesis Submitted to the Faculty of the Worcester Polytechnic Institute, UK.

Liu B, Chen S, Rundensteiner EA (2002). A Transactional Approach to Parallel Data Warehouse Maintenance, A Publication of the Worcester Polytechnic Institute, UK.

Pan L, Bic LF, Dillencourt MB, Lai MK (2002). Mobile Agents: The Right Vehicle for Distributed Sequential Computing, HiPC '02 Proceedings of the 9th International Conference on High Performance Computing pp.575-586.

Park J, Youn H, Lee E (2008). A Mobile Agent Platform for Supporting Ad-hoc Network Environment. Int. J. Grid Distributed Computing pp.9-16

Reza G, Amin MF, Hamid T, Mahdi S (2008). Evolutionary Query Optimization for Heterogeneous Distributed Database Systems. World Acad. Sci. Eng. Technol. 19:43-49.

Taghezout N, Adla A, Zarate P (2009). A Multi-agent Framework for Group Decision Support System: Application to a Boiler Combustion Management System (GLZ). Int. J. Software Eng. Appl. 3(2):9-20.

Wooldridge M (2002). An Introduction to MultiAgent Systems. John Wiley & Sons Ltd, Chichester, England.

Wooldridge M (2009). An Introduction to MultiAgent Systems.2nd ed., John Wiley & Sons Ltd, Chichester, England.