The data warehouse (DWH) is usually presented as a centralized database. In this paper, we propose a new approach to manage data storage and distribution in a data warehouse environment. This approach deals with the dynamic data distribution of the DWH on a set of servers. The data distribution that we consider is different from the “classical” one which depends on the data use. The distribution in our approach consists in distributing data when the server reaches its storage capacity limit. This distribution assures the scalability and exploits the storage and processing resources available in the organization using the data warehouse. It is worth noting that our approach is based on a multi-agent model mixed with the scalability distribution proposed by the Scalable Distributed Data Structures.

The proposed multi-agent model is composed of stationary agent classes: Client, Dispatcher, Domain and Server, and a mobile agent class called Messenger. These agents collaborate and interact to achieve automatically the storage, the splitting (distribution), the redirection and the access operations on the distributed data warehouse.

In this paper, we demonstrate the improvements obtained when we have used the multi-agent system and the Messenger agents in the data storage operation.

Keywords: Data warehouse; storage; dynamic distribution; multi-agent system; mobile agent.

1. Introduction

A data warehouse is a principal component of the information system in an organization. It is defined by its inventor, Inmon [1], as a collection of data which
are subject-oriented, integrated, stamped, non-volatile, and used to support decision making. The DWH is considered as a deposit of data that have been collected from heterogeneous and autonomous distributed sources. It is used for analytical tasks in business; such as analyzing sales/profits statistics, cost/benefits statistics or customers’ preferences statistics to make the adequate decisions. These analyses are provided by OLAP (OnLine Analytical Processing) applications.

The DWH usually contains a very large amount of data. This is because of the scope of the period that the DWH must cover (historical data) and the diversity of data sources from which data is extracted. In fact, the DWH is the subject of many research works. The research issues are classified into five groups as shown in [2]: (1) data warehouse modeling and design, (2) data warehouse architectures, (3) data warehouse maintenance, (4) operational issues, and (5) optimization.

Our research work focuses mainly on the operational issues and optimization topics, but also on data warehouse architectures and design. Our work aims at solving the problems of storage space and performance through: (1) developing a dynamic system that can manage the DWH automatically (data storage, data distribution on a set of servers, and data access), (2) taking advantage of the storage and processing resources available in the organization (processors, memory, hard disks, etc.), (3) getting better data storage time, and (4) improving the query response time.

This paper is organized as follows: Section 2 gives an overview of related works and discusses the problems related to optimization topics. Section 3 presents the example that we use to illustrate the concepts used in the proposed approach. In Sec. 4, we describe the proposed multi-agent model. Section 5 details the global dynamic of the data storage operation. Section 6 analyzes the experimental results. Finally, a conclusion and an outlook of future works are made in Sec. 7.

2. Related Works

Most researches in the literature are on optimization topics, and proposed solutions based on a centralized DWH. The latter is considered as a single database containing huge amount of data, which is very expensive from the computation perspective. The centralized data warehouse is very expensive because of the large setup costs, and it is not very flexible due to its centralized nature [10].

These researches propose several queries for optimization techniques. These techniques can be classified into two categories [5]: (1) redundant structures, and (2) non-redundant structures. Techniques in the first category compete for the same resource representing the storage cost and incur overhead maintenance in the presence of updates [6]. Among these techniques, we mention materialized views and indexes [7, 8, 14]. Compared to those in the first category, techniques in the second category do not require an extra space. Among these techniques, we mention the vertical partitioning and the horizontal one [9].
All these techniques are supported by the current database management systems (DBMS). The improvements which are provided to these systems concern the management of large data amount, are not sufficient to satisfy the needs due to the data amount growth of the DWH. In addition, the static data fragmentation schema, actually used in these systems, constitutes a major handicap.

It is worth noting that, in our approach, we use the two techniques mentioned above (non-redundant and redundant structures). The horizontal partitioning technique will be used to distribute the data warehouse on a set of machines. The materialized views and indexes will be used on each individual machine that must be tuned and optimized for performance.

So far, distribution of data warehouses has not attracted much attention in research. The use of DWH with distributed structure has appeared only with the data marts (small data warehouses containing only data on specific subjects) [21, 22]. However, the data mart does not solve the problems of storage space and performance. It is basically stand-alone and has data integration problems in a global data warehouse context [10]. In addition, the performance of many distributed queries is normally poor, mainly due to the load balance problems. Furthermore, each individual data mart is primarily designed and tuned to answer the queries related to its own subject area, whereas the response to global queries depends on the global system tuning and the network speed.

Obviously, most researches in the literature that work on the data warehouse distribution propose solutions based on the studies made on the production databases under the name of Very Large Data Bases (VLDB). These solutions are based on the “classic” data distribution which depends on the data use and has a static distribution plan. Furthermore, this type of distribution is defined at the design phase. Other researches [11, 12] use the abstract state machines [13] as a flexible and quality-oriented formal method to design and optimize a distributed DWH and OLAP applications.

We have to point out that, in our approach, the data distribution that we consider is different from the usual ones used [18]. In fact, it is not defined at the design phase. However, it is imposed by the storage capacity. As a matter of fact, when a machine reaches its storage capacity limit, we add another. Then, we distribute the data on the two machines to have a balanced load.

There are several ways to divide the relation horizontally. Typically, we can (1) assign tuples to the processors in a round-robin fashion (round-robin partitioning), (2) use hashing (hash partitioning), or (3) assign tuples to the processors by ranges of values (range partitioning). In [10, 15–17], the authors use the Data Warehouse Striping (DWS) technique. The latter is a round-robin data partitioning approach especially designed for distributed data warehouse environments. By using the DWS, the fact table will be distributed into an arbitrary number of machines which is fixed at the beginning. Consequently, the queries will be executed in parallel by all of the machines [15]. The round-robin distribution is simple to use and guarantees the load balancing, although its major disadvantage is that we
must have machines with the same treatment and storage capacities. Otherwise, some machines will be too busy and the others will be under used.

We have to note that, in our approach, we use the range partitioning. So, the queries are executed in parallel not by all the machines but only by those that contain the necessary partitions. Furthermore, the data distribution is dynamic and automatic. In fact, each time when one machine reaches its limit capacity, it starts up the data distribution operation without needing an external intervention (administrator). Moreover, the number of used machines, in our approach, is not fixed. Therefore, the storage capacity of the DWH tends theatrically to the infinite because we can, at any moment, dynamically add other machines. This infinite storage capacity and dynamic data distribution are guaranteed by the principles of the Scalable and Distributed Data Structures (SDDS) [23].

The SDDSs deal with the storage of a large data amount on a set of interconnected machines. The SDDS principle consists of distributing the file contents in a way that allows us to benefit from the available memory on a set of interconnected machines [24, 25]. This distribution is based on the identifiers (keys). In fact, the data identifiers residing in one machine must be included between a lower bound mark and a higher one (see Sec. 4.1). The increasing content of the file involves its splitting. This principle has been extended from files to operational databases [19, 20]. In the rest of this paper, we consider that the two terms “splitting” and “distributing” have the same significance.

According to the proposed approach, the DWH will be distributed on a set of machines. In this case, the data management needs the collaboration and the interaction between those machines in order to reply to the user’s queries while assuring the parallel processing of these queries. Thus, we have chosen to use the Multi-Agent System (MAS) with the mobile agents as essential actors. This type of agent is an autonomous and adaptable software entity that is able to move dynamically (code, data and execution state) in order to reach data or remote resources. The mobile agents have proven to have a high performance when we access the data distributed on a set of interconnected machines [26].

In the next section, we introduce the example used to illustrate the concepts of the proposed approach. The experimentation is also based on it.

3. Illustrative Example

In this paper, we assume that a multidimensional database is based on a relational data warehouse in which the information is organized in a star schema [3], as illustrated in Fig. 1.

A star schema is a set of dimensional tables and fact table. For most cases, the latter contains a larger data amount than the former; that is why the splitting (distribution) operation is mostly executed on the fact table. On the other hand, the dimensional tables are generally duplicated on each server, except when their PK (Primary Key) constitutes a distribution criteria (see Sec. 4.1). The star schema
is also optimized particularly for the execution of complex queries that aggregate a large amount of data from the fact table.

In this paper, we will consider the star schema presented in Fig. 2 as an example to illustrate some concepts. This star is relative to sales activity. The fact table SALES will be split. On the other hand, the dimensional tables will be either split or duplicated.

In the following, we describe the proposed multi-agent model, the data distribution principle, and the waiting database.

4. Proposed Model

In this paper, we propose a model for solving the problems in the DWH context using the available resources in the organization. These problems are related to data storage, splitting and access.

In our approach, the use of MAS is very interesting because it allows assuring the progress of the dynamic data distribution, the collaboration, the interaction, the independency of the different machines, and the parallel execution of the user queries. In addition, the use of mobile agents in the proposed solution seems to be very helpful because it allows: (1) decreasing the network loads, (2) liberating client machines during the results preparation that needs generally a very important runtime, and, essentially, (3) securing the data that are transported in the network (see Sec. 6).

In our solution, we use the SDDS principle based on data distribution through intervals (range partitioning) in order to distribute the data of the DWH on a set of machines. This type of distribution allows the decomposition of the DWH into a set of domains. Each domain can be stored on one or more machines according to its data size.
4.1. Principle of data distribution

In our approach, the DWH is horizontally distributed on a set of machines that have the same DBMS and the same star schema (see Fig. 3). Furthermore, on each machine, we can use the materialized views and indexes to tune and to optimize the performance.

The principle is to start with a single machine for which we define: (1) the storage capacity limit of this machine for which the used DBMS gives its highest performance (for data access and storage), and (2) both the inferior bound mark and the superior one for each fact table key. When this machine reaches its limit, we add another one and we distribute the data on the two machines to obtain a balanced load. In most cases, the fact table undergoes a splitting operation because of its important volume. The dimensional tables are distributed when their key constitutes a distribution criterion. Otherwise, they are duplicated.

In Table 1, we present a scenario of data splitting.

<table>
<thead>
<tr>
<th>Start</th>
<th>Machine 1</th>
<th>First Splitting</th>
<th>Second Splitting</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer_Id</td>
<td>[A, Z]</td>
<td>[A, Z] [A, Z]</td>
<td>[A, Z] [A, Z] [A, Z]</td>
<td>...</td>
</tr>
<tr>
<td>Product_Id</td>
<td>[0, 99]</td>
<td>[0, 50] [51, 99]</td>
<td>[0, 50] [51, 99] [51, 99]</td>
<td>...</td>
</tr>
<tr>
<td>Region_Id</td>
<td>[AA, ZZ]</td>
<td>[AA, ZZ] [AA, ZZ]</td>
<td>[AA, ZZ] [AA, ZZ] [AA, ZZ]</td>
<td>...</td>
</tr>
<tr>
<td>Date_Id</td>
<td>[Jan, Dec]</td>
<td>[Jan, Dec] [Jan, Dec]</td>
<td>[Jan, Dec] [Jan, Jun] [Jul, Dec]</td>
<td>...</td>
</tr>
</tbody>
</table>

Machine 1 starts up the first splitting operation when it reaches its capacity storage limit. First, we search for the key value that gives two balanced partitions (e.g. Product_Id that is an integer of two numbers). Then, we move the data related to the new interval to machine 2. Finally, we update the intervals. The second splitting operation is launched by machine 2 (e.g. Date_Id that is a date). The same process is started each time when one machine reaches its limit capacity. In fact, the data distribution can be continued according to the same criteria or to others (Customer_Id, Region_Id).

We notice that each SALES table record belongs to only one DWH partition. If we consider that each of these DWH partitions is stored in separate databases,
we must, on the one hand, split the Date table and Product table according to the same criteria used for the SALES table. On the other hand, we duplicate the other tables in order to (1) facilitate the checking of the integrity constraints, (2) ensure the databases autonomy, and (3) improve the joining time when we access data.

In the following, we present the proposed multi-agent model architectures and the waiting database notion that we use in our approach.

### 4.2. The proposed multi-agent model

The proposed model consists of five static agent classes (Client, Dispatcher, Splitting, Domain, and Server) and a mobile agent class (Messenger). Each agent class is defined by its knowledge (static or dynamic), its acquaintances (agents that it knows and with which it can communicate), and its behavior [27].

Figure 4 illustrates the interaction between the different agents.

![The proposed multi-agent model architecture.](image)

Fig. 4. The proposed multi-agent model architecture.

The Client agents act as an interface between the user and the DWH management system (Dispatcher agent). In fact, the user utilizes the Client agent to send the data storage and the data access operations (queries) to the Dispatcher agent. Each Client agent has the Dispatcher agent as an acquaintance. Its static knowledge is made up of its name and its address. This agent class does not have a dynamic knowledge.

The Dispatcher agent arranges the received operations according to their arrival order. These operations will be treated by the Messenger agent. When the Dispatcher agent receives the operation results from the Messenger agents, it sends them to the Client agent, if the latter is connected. Otherwise, it saves them until
the Client agent will be connected again. The acquaintances of the Dispatcher agent are: (i) Client agents which sent queries, (ii) Messenger agents which take charge of executing these operations, and (iii) Splitting agent. Its static knowledge consists of its name and its address. Its dynamic knowledge is made up of a list containing all the Domain agents existing in the system and two waiting queues. The first queue is used to store operations received from the Client agents. The second one is used to store the results provided by the Messenger agents. Then, the Dispatcher agent sends these results to the sending Client agent (as described above).

The Messenger agents take charge of executing each operation found in the operations waiting queue of the Dispatcher agent. Each Messenger agent makes the execution plan of this operation. Then, it visits all the Domain agents concerned with this operation. Finally, it gives the ultimate results to the Dispatcher agent. Each Messenger agent has as acquaintances the Dispatcher agent and the Domain agents necessary to execute the operation. Its static knowledge is made up of its name and its maximum size of data that it can transport. This maximum depends on the network characteristics. The Messenger agent dynamic knowledge consists of:

(i) the list of Domain agents to visit for executing the operation,
(ii) the operation to execute,
(iii) the lists of data to store (if the operation is data storage), or the list of data that are collected from visited Domain agents (if the operation is data access), and
(iv) the size of transported data.

It has a very important role in our architecture because it allows:

1. reducing the message traffic on the network,
2. accelerating the data storage and access operations, and, essentially,
3. securing the data circulation on the network (see Sec. 6).

The Domain agents are responsible for sending the operations to the Server agents which they control. Then, they collect the replies sent by the Server agents and transmit the final result to the Messenger agent. The Domain agent has as acquaintances: (i) the Server agents that are under its control, (ii) the Messenger agents with which it has operations to execute and (iii) the Splitting agent. Its static knowledge is composed of its name, its address, the disk space limit of each Server agent, the maximum number of Server agent it can manage and the maximum size of data it can receive from the Messenger agents. This maximum depends on the machine characteristics (memory, processor, etc.). Its dynamic knowledge consists of the descendant list, the size of memorized data, and two waiting queues. The first queue is used to store the operations brought by the Messenger agents. The second one is used to store the replies sent by the Server agents. Later on, the Domain agent sends them to the appropriate Messenger agent.
The Server agents undertake the received operation and send the replies to the Domain agent. Each Server agent has the Domain agent to which it belongs as acquaintances. Its static knowledge is made up of its name and its address. Its dynamic knowledge is a waiting queue used to store the operations received from the Domain agent.

The Splitting agent is responsible for the splitting operations and maintaining the data road card that allows finding the data location. The splitting operation is started up when the machine reaches its storage capacity limit. The role of this agent consists of the following steps. First, it creates a new Domain agent when it receives a splitting request. Then, it informs the Domain agent, asking for splitting of the location and the characteristics of the new one. Finally, it sends to the Dispatcher agent the new information concerning the two Domain agents in order to update the Domain agents list. The Splitting agent has as acquaintances the Dispatcher agent and the Domain agents that ask for splitting. Its static knowledge consists of its name and its address. Its dynamic knowledge is the list of splitting requests sent by the Domain agents.

The Dispatcher agent manages a metabase which allows it to follow the evolution of the data distribution on the Domain agents, the network status and the Messenger agents load rate (see Fig. 5). This metabase is also used by the Messenger agents to make the execution plans of the received operations and determine the Domain agents to visit. The Splitting agent also uses this metabase for the splitting operations and updates it at the end of each splitting operation (see Sec. 5.2).

Furthermore, each Domain agent has an appropriate metabase in order to follow the evolution of the data distribution on its descendants (Server agents) (see Fig. 5, the framed tables).
4.3. Waiting database

The DWH contains data extracted from different operational databases. Generally, these data come in a file form. This file can contain either the data to insert (records set) or the SQL commands (INSERT commands). The classic way of storing these data consists in diffusing this file to all of the used machines. Each machine will analyze the data in the file and insert those which belong to its interval. This diffusion leads to an increase in the network traffic and the overload of the different machines.

To solve these problems, we suggest the use of a waiting structure that we call waiting database (WDB). In the latter, we store temporarily the data coming in a file form. Then, we distribute them on the different machines according to the used distribution criteria (see Fig. 6). This WDB has the same star schema as the distributed DWH and it is managed by the Dispatcher agent.

The operations of data insertion in the WDB and data sending towards the target machines will be done in an asynchronous way. In fact, the source file content will be stored in the WDB. Then, the data will be sent to the appropriate machines. Finally, the data of the facts table will be deleted from the WDB and we keep the data of the dimensional tables.

In the following, we will provide details of the dynamics of the proposed model for data storage and data splitting operations.

5. Multi-Agent Dynamic

The proposed model is designed to support the different management operations of the data warehouse, namely the data storage, splitting, redirection and access. In this paper, we present only the data storage and splitting operations.

The sequence diagrams presented later describe both the interactions and the agent behaviors made to accomplish the data storage and splitting operations. The formalism used to represent these diagrams is the MA-UML (Mobile Agent – UML) [28], which is an extension of AUML (Agent UML allows modeling of the mobile agent behaviors).

5.1. Data storage

We will present two scenarios for this operation. In the first, we will not use the Messenger agents. In the second, we will benefit from these mobile agents and prove
their utility. For the two scenarios, we suppose that the data come in a file form (INSERT commands). In the two cases described later, we will not consider the data redirection, splitting and rejection.

5.1.1. Scenario 1

In this scenario, the used agents are: the Dispatcher agent, the Domain agents, and the Server agents. These agents exchange different messages in order to accomplish the data storage operation without using the Messenger agents (see Fig. 7).

The data storage operation is started up when the Dispatcher agent receives the data file. This agent stores the data in the WDB. Then, it finds the Domain agents to which the received data belong. The Dispatcher agent uses the available information in its metabase to determine these agents and their addresses, and make for each Domain agent its proper queries, such as:

\[
\text{Insert Into SALES ( ...)} \\
\text{(Select ... From SALES@WDB} \\
\text{Where (Product_Id >=0 and Product_Id <=50 and ...)}
\]

Each time the Dispatcher agent formulates the queries; it sends them to the appropriate Domain agent.
The following pseudo-code sums up the Dispatcher agent tasks:

```plaintext
While (not end of Data_File)
  Read line
  Insert line into WDB
End While
Make Domain_agents_list
While not end of Domain_agents_list
  Formulate the appropriate insert_query
  Send insert_query
End While
```

When the Domain agent receives these queries, it verifies, for each query, whether the clause WHERE belongs to the Server agents which are under its responsibility. If this condition is true, the Domain agent sends this query to the appropriate Server agent. Otherwise, referring to the available information in its metabase, the Domain agent forwards the query to the right Domain agent. This query redirection occurs when a splitting operation happens before the query’s arrival. The appropriate pseudo-code is as follows:

```plaintext
When receiving insert_query
  Find concerned Server agent
  Send insert_query
```

The Server agent executes the received query to load the appropriate data existing in the WDB. Then, it formulates a DELETE query with the same WHERE clause to clear the fact table in the WDB. Finally, the Server agent informs the responsible Domain agent that the data loading is successfully done. The Server agent starts up the splitting operation, if it detects, at the data loading time, that the machine has reached its storage capacity limit. The following pseudo-code describes the Server agent tasks:

```plaintext
When received insert_query
  Execute query and load data
  Inform Domain agent (operation_terminated)
```

When the Domain agent receives all the replies from the Server agents, it informs the Dispatcher agent that the data storage operation is successfully terminated.

When the Dispatcher agent receives all the replies from the Domain agents, it informs the administrator that the storage operation is achieved.

5.1.2. Scenario 2

In this second scenario, the agents used are: Dispatcher agent, Domain agents, Server agents and Messenger agents. The afore-mentioned agents exchange different messages in order to accomplish the data storage operation using the Messenger agents (see Fig. 8).
In this scenario, when the Dispatcher agent receives the data file from the administrator, it creates a Messenger agent.

The Messenger agent reads the data, file each record separately, and composes the appropriate data list for each Domain agent. It uses the available information in the metabase to perform this process. Meanwhile, the Messenger agent makes up the address list of the visited Domain agents, according to the data lists that it can bring. Generally, the Messenger agent cannot load the entire data file in order to avoid burdening the machine memory and/or the network. So, the loaded data lists size must not be superior to the maximum loading capacity of the Messenger agent. This capacity is relative to the machine and the network characteristics. If the Messenger agent reaches its capacity limit and cannot load all the records in the data file, it informs the Dispatcher agent. The latter creates another Messenger agent that repeats the same process. The Dispatcher agent generates the necessary number of Messenger agent to load all the data file records. The pseudo-code of the Messenger agent is the following:

\[
\text{N\_Line} = 0 \\
\text{While (not end of Data\_File)}  \\
\quad \text{And NLine < Messenger\_agent\_Capacity)}  \\
\text{Read line}
\]
Each Messenger agent visits the Domain agents for the first time to distribute data lists. After each visit, the Messenger agent deletes the distributed data list and updates its transported data size by subtracting the data list size. Before its first visit to each Domain agent, the Messenger agent requests its permission. The Domain agent allows the Messenger agent to visit it, when the sum of the data size, in its memory, and the size of the data lists transported by the Messenger agent, is inferior to the maximum data size that can be carried by the Domain agent. Otherwise, the Messenger agent asks the next Domain agent in the address list for its permission. If all the Domain agents cannot receive the Messenger agent, it must wait until one of these agents is discharged. Once distributing the data lists on the Domain agents is completed, the Messenger agent visits them again to verify whether the storage operation is successfully done. When it collects all the Domain agent replies, it returns to the Dispatcher agent and informs it that the storage operation is successfully accomplished.

When receiving the data lists from the Messenger agent, the Domain agent updates its memorized data size by adding the size of the received data lists. Then, it verifies whether the data list belongs to the Server agents which are under its responsibility. If this condition is true, the Domain agent sends these data to the appropriate Server agents. Otherwise, the Domain agent forwards this data list to the right Domain agent. The last case occurs when a splitting operation happens before the data arrival. After sending or forwarding the data list, the Domain agent updates the size of its memorized data by subtracting the data list size.

The Server agent stores the received data list and informs the responsible Domain agent that the storage of the data list is successfully accomplished. The Server agent starts up the splitting operation, if it detects, while storing the data, that the machine has reached its storage capacity limit.

The Domain agent informs the Messenger agent that the storage of the received data lists is terminated, when it receives all the replies from the Server agents. When the Dispatcher agent receives all the Messenger agents replies, it informs the administrator that the storage operation is successfully achieved.

5.2. **Data splitting**

The agents contributing in the data splitting operation are: Server agent, Domain agent, Splitting agent, Messenger agent and the Dispatcher agent.

The splitting operation will be automatically started up when the Domain agent detects that the available space at the Server agent cannot support entirely the data amount received at the storage operation. This operation does not necessitate the
modification of both the treatments (agents) and the data structures.

Each Domain agent is characterized by a maximum number of Server agents that it can control. If the Domain agent does not reach this number, the splitting operation will be managed by the split Domain agent and a new Server agent will be created. Otherwise, the splitting operation will be managed by the Splitting agent and a new Domain agent will be created.

The afore-mentioned agents exchange different messages in order to achieve the splitting operation which involves the creation of a new Domain agent. This exchange is shown in the diagram presented in Fig. 9.

When the Server agent detects that the available space cannot support the total amount of the received data, it sends a message informing the Domain agent that it reaches its capacity limit and it needs to split its data. This message contains, also, the data that are not inserted. Then, the Domain agent sends a message to the Splitting agent. This message contains the new values of both the superior bound mark and the inferior one. To determine these bounds, the Domain agent computes its records and determines the key values which allow dividing these records into two balanced parts.
When the Splitting agent receives the spilling request, it informs the Dispatcher agent that the Domain agent \( i \) will start up a splitting operation. Thus, the Dispatcher agent stops momentarily the operations sent to this agent. The Splitting agent is responsible for preparing the new Domain agent. Once the new agent is created, the Splitting agent informs the Domain agent asking for splitting of the new-created agent address as well as the bound marks of each dimensional table. According to these bound marks, the split Domain agent selects the data from the Server agents and sends them to the Messenger agent. Then, it informs the Splitting agent that the splitting operation is terminated and updates its descendants list. The Messenger agent moves to the new Domain agent and gives it the received data. This new agent achieves the storage operation as described in Sec. 5.1.2.

When the Splitting agent is informed that the splitting operation is terminated, it sends a message informing the Dispatcher agent that the new Domain agent and the split Domain agent are ready to receive operations. So, The Dispatcher agent updates its domain agents list according to the received information.

In the following, we present the results obtained for the data storage operation.

### 6. Experimental Evaluation

In order to validate our model for the data storage operation, we have implemented four prototypes. One of them permits to store data on a centralized database (DB). The others allow storing data on a set of machines (having the same configuration: P4 and 256Mo RAM): one of them contains the WDB and/or the MB; the other machines (three then five) contain the DWH partitions. We have, also, used JDeveloper10g as a development toolkit, Oracle as a DBMS, and IBM Aglets as a multi-agent platform. We have programmed an engine that generates file-form data (INSERT commands) representing only daily or weekly DWH refreshment. In the future, we will generate a larger data amount.

In the first prototype, we have used only one machine and we have programmed an insertion engine which reads the data from the file and inserts them on a centralized DWH.

In the second prototype (see Fig. 10), we have programmed an insertion engine, without MAS, that reads the data from the file and inserts them on the distant DWH partitions using the database links (@DWH1, ..., @DWHN, etc.).

![Fig. 10. Machines for the prototypes without MAS.](image-url)
In the third prototype (see Fig. 11), we have programmed the first scenario. In the last prototype (see Fig. 12), we have programmed the second scenario.

In the two last prototypes, the machines are used as follows: (1) at one of these machines we have made the Dispatcher agent, the metabase (MB) and the WDB (prototype 3) or the messenger agent (prototype 4), and (2) at each of the other \( N \) machines, we have made a Domain agent, a partition of the DWH database (DWHi), a metabase and a Server agent.

We have implemented these prototypes in order to prove the improvements given by the MAS and by the Messenger agents in the data storage operation.

We have to note that in the four prototypes, we will consider that the data storage operation goes off without needing to the data splitting operation.

In Fig. 13, we present the insertion time given by the four prototypes using three machines. We notice that the time needed to insert the same data size in a centralized DWH is less than the time needed to insert data in distributed DWH without using MAS (an average gain of 67%). In fact, the data transfer on the network is behind this difference. So, when we have used the MAS and the WDB, to distribute the data, in the third prototype, we have reduced the time of communication between the machines. Thus, the time needed in case of a distributed insertion is reduced significantly (an average gain of 61%). It becomes close to the time needed in case of a centralized DWH (plus an average of 18% to the centralized prototype). This gain is thanks to: (1) the local data storage (executed by the Dispatcher agent) in the WDB, and (2) the parallel data loading (done by the Server agents).
The use of the mobile Messenger agents in the forth prototype, not only gives the best storage time (an average gain of 4.8% the maximum gain reaches 30% compared to the insertion time given by the centralized prototype) but it also secures the data circulation on the network. In fact, we have made a function that the Messenger agent executes, at each time, when it reaches one machine. This function allows the Messenger agent to check whether the address of the reached machine belongs to its address list. If the address is not found, the Messenger agent tries to leave this machine. If it cannot leave this machine, it destroys the transported data and kills itself. The time reduction is the result of reducing the time of read/write operations (done on the disk of the machine that contains the Dispatcher agent) and storing the data directly in their final destination.

In Fig. 14, we present the same measurements for the four prototypes using the same data sizes, as in Fig. 13, and five machines. We remark that we obtain the
same improvement when using the MAS and the Messenger agents. Furthermore, the average gain between the insertion time in the first prototype (centralized DWH) and the third one (MAS without MA) becomes 6.5% (with three machines, we have obtained +18%). In addition, the average gain between the insertion time in the first prototype and the forth one (MAS with MA) becomes 6% and the maximum gain reaches 37% (with three machines, we have 4.8% and the maximum gain 30%).

So, we can conclude that when the number of used machines increases: (1) the difference between the insertion time in a centralized DWH and a distributed one decreases, and (2) the average gain in the insertion time gives by the Messenger agent increases.

7. Conclusion

In this paper, we have proposed a MAS-based approach to distribute horizontally the DWH on a set of interconnected machines available in the organization. The distribution principle consists of starting with a single machine and adding dynamically another one when the first reaches its capacity limit. Once the second machine is added, we redistribute the existing data equally on the two machines. It is worth noting that our approach is beneficial as it solves problems related to data storage, splitting and access.

The experimental results illustrated in this paper show how the use of MAS and mobile agent in the data storage operation is efficient as it reduces data storage time and secures data circulation on the network.

Our future works aims at defining the multi-agent dynamic for the data access operation. As well as, we will implement this operation and compare the given results to those obtained for the Benchmark used in the literature (TPC-H and APB-1).

References

5. L. Bellatreche and K. Boukhalfa, An evolutionary approach to schema partitioning selection in a data warehouse, in Proc. Int. Conf. on Data Warehousing and Knowledge Discovery (DAWAK’05), Copenhagen, Denmark, 2005.

