Task Allocation Strategies: A Study with a Multi-Agents System in Fully Distributed Information Systems

FÉLIX F. RAMOS CORCHADO1
LIMING CHEN1
MARC BUI1
DIDIER DONSEZ1
PASCAL FAUDEMAY2

1Université de Technologie de Compiègne
Heudiasyc, URA CNRS 817 Genie Informatique
Centre de Recherches de Royallieu.
BP 529 60205 Compiègne CEDEX
{ramos, chen, bui, donsez}@hds.utc.fr

2Université de Paris VI
Laboratoire MASI / IPB
4 Place Jussieu
75 252 PARIS cedex 05
faudemay@masi.ibp.fr

Abstract

In this paper we present some task allocation techniques useful on Internet or other Wide Area Network (WAN). These techniques are based on the cooperation of multi-agents organized in one community of Reactive agents and another of Allocator agents. Reactive agents represent services available on the WAN such as Internet search engines. Allocator agents' task is allocate the queries that they receive from users. To take such a decision, Allocator agents cooperate sharing their informations about the state of Reactive agents. An Allocator agent is able to characterize the behaviour of Reactive agents by means of a knowledge model we developed. In order to facilitate the composition of complex services such as retrieval of multiple language documents, we also define another agent type we call Sub-contractor agent.

KEY WORDS: Load balancing, Agents, cooperation, document query

1 Introduction

Recently, software applications such as the World Wide Web have allowed people from all walks of life to have access to the internet. This has caused massive development in information server technology offering all sorts of multimedia data. Problems such as the reliability of transactions, the searching of specific documents, selection of information servers and improvement of the access time, etc. have to be studied in the context of this new technology.

Improving response times implies improving such parameters as bandwidth mainly on backbones, the use of mirrors, the optimization of routing between the user and the information server etc. Also, the use of software techniques such as document filtering, cooperative retrieval, optimized document placement and task allocation become relevant. Within the U-Doc project, [15] a French project currently under way whose
objective is the implementation of a collection of assistance tools for hyper-document retrieval on the Internet, we are studying optimized query placement in order to reduce the response times.

Task allocation and load balancing have been widely studied in the literature [3, 19, 17, 4, 21] in the context of distributed systems. The purpose is to optimize the use of resources and improve the performance of application processing. There exist static and dynamic techniques to implement load sharing in distributed systems. In the Internet context, static solutions [3, 16] based mainly on results from operational research, are not applicable, as they rely on previous knowledge of both the system and the application. Dynamic solutions [19, 9, 4, 12] try to remove this constraint. An estimate of the availability of several parameters such as the number of processors, the task processing time etc. is made in order to determine the system state. When accessing a document or submitting a query on the Internet, the geographical location of the target site may result in poor precision in the values of these parameters. The only information available are the response time and transfer throughput, so that other approaches are needed.

In this paper we propose several strategies for dynamic query placement. The objective of these strategies is to optimize the use of information servers in order to reduce the response time of services on the Internet. They are based on the use of multi-agents [13, 11, 14], organized into a community of Reactive agents and a community of Allocator agents. Reactive agents are merely the final servers (available services in the system such as indexing engines, bibliographic databases, movie databases, etc.). Allocator agents are able to cooperate and to learn about the state of Reactive agents. Their objective is to place the queries which they receive while trying to optimize the use of information servers. A knowledge-based model is also developed which enables our Allocator agents to characterize the quality of service of the Reactive agents on the Internet, and to gather information about the system state through learning and experience. When a query is submitted to an Agent, it uses its knowledge of the system state to place the query. If it’s knowledge is insufficient to take this decision, it interrogates a particular group of Allocator agents to try and complete its knowledge. If the knowledge obtained does not enable it to decide on the allocation, it initiates a negotiation process [19]. Finally, in order to enable composition of complex services with large added value to be achieved, a new type of sub-contractor agent is defined.

The paper is organized as follows: in Section 2 the context of this work is discussed. In Section 3, we introduce the agent model and the way in which the Allocator agents represent their knowledge and learn. Strategies for dynamic query placement are described in Section 4. Sub-contractor agents are introduced in Section 5 and in the conclusions, we compare our approach with other agent-based approaches.

2 Context and Problems

The problem was studied in the context of the U-Doc project, whose objective is the implementation of a collection of assistance tools to facilitate document access on the Internet, as well as the implementation of their administration. We first briefly describe the U-Doc architecture [8] followed by a description of our framework.

2.1 The U-Doc Architecture

The U-Doc architecture is depicted in Figure 1. The client access request arrives at the external interface of U-Doc (mailer or DQBE). After formatting, the request is delivered sequentially to:

1. the Concepts Manager and thesaurus module which divides the request in more precise and domain-dependent ones (e.g requests about colours, sounds, geography, etc.)

2. the Indexer module which searches through documents in the local documents database

3. the profiler which extracts the long term profile from the immediate request
4. the *Examiner* which delivers the clients' immediate and permanent requests produced by the *Profiler* to external *Searchers* (lycos, Y100, etc.) and evaluates the abstracts and titles obtained.

5. the *Gatherer* to search the selected documents. These documents are then delivered to the *Storage system* and finally to the client.

The *Storage system* keeps the documents, abstracts and annotations in a cache memory and in the tertiary memory. The *Thesaurus* manages a corpus of the reference documents (for instance, articles of a review previously chosen to describe the area concerned), and learns the correlation between the concepts in that corpus.

![Figure 1: Architecture of U-Doc.](image)

### 2.2 The Framework

A document retrieval query relies on localization, format modification, translation and document selection operators, which can be based on indices, and on information extraction on normalized documents such as SGML documents. In our architecture, a retrieval query is decomposed by the DBQE module into a set of document retrieval operators, represented by a data flow graph such as the one in Figure 2. We assume query decomposition techniques are known, as they are not the subject of this paper. Based on a placement strategy aiming at response time optimization, these operators are placed on specialized servers such as those which propose services for document searching.

On the Internet, there are many such search services (indexing engines), such as AltaVista, Lycos, Yahoo, etc. A document can also be replicated on several sites, such as proxies or mirrors. Presently, dynamic query placement in U-Doc takes place in two places: firstly at the Examiner level choosing a search server on a previously defined list, when the query comes from a user of the profiler. Secondly, at the Gatherer module level, for the choice of a document when this document exists in several servers.

In both cases, measuring the classical parameters may not help the allocation problem. Therefore, we do not have a classical task allocation problem in the usual sense, as we cannot determine process allocation in the remote sites.

One solution in our Internet framework of added value services, is to submit the same query to all servers and choose the one which delivers the answer in the shortest time. This is a simple but an expensive solution. The problem is, therefore, to define dynamic placement strategies on the various subtasks of a document retrieval query, in order to reduce the response time. We propose an heuristic solution, based on past experience in terms of response times, and possibly on the experience of other sites.
2.3 The Problem

Our main objective in the U-Doc project is the minimization of response time. To meet this objective, we must solve the problem of how to choose the searcher and the document server (if the choice exists) to get the shortest response time. In the previous explanation of the U-Doc architecture's behaviour, the allocation techniques are necessitated by the Examiner to select a searcher from a set previously established. The Gatherer selects one document server from the list obtained by the selected Examiner.

The parameters our system have are the actual ones, that is, Internet available searchers (Lycos, Yahoo, etc.) do not give information about its states useful to calculate the response time of a request (e.g. number of tasks waiting for processing, processing power of the server, size of the waiting tasks, etc.). To make this problem transparent to the user, two solutions are possible. The first is to choose the servers (to search and to retrieve the document) randomly. The second is to select the servers, taking into account the experience obtained about previous relationships. Because we are using Allocator agents having a learning capability, we choose the second solution. One of the strategies proposed is a combination of both, however.

3 Model

Though our approach was developed for the U-Doc distributed information system, the proposed mechanisms are general ones and can also be applied in other contexts. Figure 3 depicts the distributed system that we take as the framework used to simulate the performance of the algorithms. The system consists of three sets distributed throughout a set of sites (computers) connected by a network: a set of allocator agents denoted by $C_A = \{C_{A1}, C_{A2}, C_{A3}, \ldots, C_{Ao}\}$. Another of Reactive agents denoted by the set $R_A = \{R_{A1}, R_{A2}, R_{A3}, \ldots, R_{Ao}\}$, and a set of users, denoted by $U = \{U_1, U_2, \ldots\}$. A site lodge zero or one user, zero or one Allocator agent and zero or more Reactive agent.

The set of Reactive agent represent the available services in the system. The same service can be deliv-
erected by different Reactive agents. Allocator agents receive the tasks delivered by users, are able to communicate with each other by sending messages over the communication network, learn about the system state and have the skill to allocate a task based on their knowledge about the system (servers). Finally, users or clients deliver theirs tasks $T_u = \{t_1, t_2, \ldots, t_n\}$ to the Allocator agents via an interface.

In the U-Doc architecture, the set of Reactive agents represent the set of searchers available (Lycos, Yahoo, etc.). A delivered task is either a search request that should be addressed to one of the searchers or a request of access to get an specific document that must be delivered to one of the document servers.

Figure 3: The system is organized in two communities: one of Reactive Agents delivering the services available over the system and the other consists of Allocator agents storing the global state of the system. The major objective of Allocator agents' community is collaborate to distribute in a fair way the load among the first community.

As depicted in Figure 3 the structure of a Allocator agent contains a knowledge and a control elements. Control element implement the location policy. The transfer policy is every time a task is received, decide about its transfer, and the selection policy is allocation of the tasks arriving). To decide about the allocation of a task, the Allocator agent requires information about the state of the servers offering the required service. As established in Section 2.2 of our framework, even if Reactive agents are able to deliver useful information to decide about an allocation, communication delays render them useless. We are using the learning capability of our Allocator agents to alleviate this problem. Figure 4 shows the information learned by Allocator agents.

3.1 Learning and Knowledge

The Allocator agents are able to learn about the evolution of Reactive agents’ states. For this purpose, they memorize for each Agent, the next piece of information that will be helpful to “predict” their service quality. We denote $q_{zi}$ as the the quality of service $x$ on the Reactive agent $i$. 
Figure 4: Knowledge an Agent has about the system's state.

- $q_{z1}$: quality of service.
- $Th$: throughput of a query on the network.
- $T$: response time of a reactive agent to a query.
- $x$: the service proposed by the reactive agent.
- $D$: an array containing the days of the week.
- $H$: an array containing Allocator agent local time.
- $V$: validity duration of the knowledge.
- $\mu$: a coefficient between 0 and 1, which represents the Agent's ability to remember the past.

In the Internet context, the quality of a service of a Reactive agent is a function of the response time, the throughput, the Reactive agent local time (a server is more heavily loaded during the working hours than at night) and day of the week (weekends are less heavily loaded than other days) [5]. Thus a Allocator agent learns the behaviour of a Reactive agent for each hour of the day and for each day of the week. To do this, it uses the $q_{z1} = Th/T$ formula to measure this quality when it sends it a task. The response time is an indicator of the Reactive agent's load, while the throughput is an indicator of the network's load.

It is necessary to take into account the changes of behaviour of a Reactive agent, however. In our case, every time an answer from a Reactive agent is received, its quality service for that day and time is modified as follows:

$$q_{z} = \mu \cdot q_{z} + (1 - \mu)q$$

where $\mu$ represents the Agent's ability to remember the past and lies between 0 and 1. The choice of the best value of $\mu$ is determined by the simulation results.

However, if the information has not been updated before some delay $V$, it is considered to be out of date. In this case, the Agent must start a new learning phase on all the relevant knowledge.

4 Dynamic Request Placement Strategies

Two criteria appear to be essential in the dynamic request placement. Firstly, work distribution implies that an application must be widely distributed in order to use the available services in the best possible way. The locality criterion aims at reducing the overheads due to communication tasks by dispatching the application only over a neighborhood. These two criteria are easily expressed by an economic equation [7, 21], but
one can see in a straightforward way that these requirements conflict. The strategies presented are dynamic and non pre-emptive. They are based on the behaviour of Allocator agents which collaborate to achieve a common goal; Reactive agent which execute a task; a bidding protocol [19] used by Allocator agent to get information about the system state and finally on the learning capacity of the Allocator agent. The use of a multi-agent approach allows us to deal with the trade-off problem in a dynamic way.

4.1 Strategy of Placement Based on Service Negotiation

Initially, the Allocator agents have no knowledge of the state of the system due to lack of experience. To determine which Reactive agent to choose and at the same time to enrich its knowledge, it starts a process of negotiation similar to that of bidding, found in free markets. Three phases in such a process are identified. Firstly, a request-for-bidding is launched beside all Reactive agents proposing the service. Secondly, an evaluation of the Reactive agents’ replies is executed; and thirdly, the contract attribution phase determines the reactive agent on which the request is placed. If this negotiation mechanism is general and simple, it necessitates a large number of messages. In a context of a large system such as the Internet, the cost associated with such a communication can be prohibitive. Most importantly, the servers are currently unable to reply to a request for a bid. (There is presently a great deal of effort to establish the actual minimal information needed in current distributed systems, some formalisms like KQML [18] and kif [10] have been studied). Therefore, we propose strategies based on this simple bidding negotiation protocol, but using the learning capacity of Allocator agent and different organizations of the communities of Reactive agent and Allocator agent to minimize the number of times it must be executed.

Strategy 1: when the Allocator agents have an individual organization

We present here the behaviour of the system when the Allocator agents use the simplest organization, that is they work individually. In this organization, when an Allocator agent receives a task, its knowledge of the system’s state should be used for the task allocation. If the knowledge is insufficient, the Allocator agent start a process we call also RFB consisting of broadcasting a test request to learn about the system state, and allocate the task.

```
Procedure Allocator agent’s allocation mechanism

Case event of { Task T :
    For each subtask t ∈ T do
        If local information is enough to allocate t
            allocate(t);
        Else { * start the negotiation to get the service *
            RFB(service);
            evaluate(offers);
            allocate(t);
            update knowledge;
        }

    Load : ... 
    ...
}
```
Example 1.

To demonstrate the behaviour of the algorithm, we consider the next example having two services $S = \{s_1, s_2\}$ offered by Reactive agents placed on sites $A$, $B$, $C$ and $D$. The service $s_1$ is offered by Reactive agents of the sites $A$ and $B$ while the service $s_2$ by sites $C$ and $D$. Submitted requests are $R_1$, $R_2$, $R_3$ and $R_4$ that make calls to services $\{s_1, s_2\}$, $\{s_1\}$, $\{s_1, s_2\}$ and $\{s_1\}$, respectively.

Table 1 illustrates the execution of these sequence of requests using this strategy. Initially, sites do not work. Similarly, Allocator agents have no knowledge of the state of the system. When an Allocator agent has to place the request $R_1$ that makes a call to services $s_1$ and $s_2$, the former makes a RFB. All four sites $A$, $B$, $C$ and $D$ propose the same quality of service, the Allocator agent places without preference the request $R_1$ on $A$ and $C$. The Allocator agent also correspondingly modifies its knowledge of the quality of service of the two sites. During the arrival of request $R_2$, the Allocator agent, consulting its knowledge base, allocates this request to the site $B$ that has become the site proposing the best quality of service for $s_1$. The execution of all requests necessitates six broadcast and 24 point-to-point communications.

Table 1: Execution of the sequence $R_1$, $R_2$, $R_3$, $R_4$, $R_1$ and $R_5$

<table>
<thead>
<tr>
<th>Temp</th>
<th>Site/Req.</th>
<th>$s_1/A$</th>
<th>$s_1/B$</th>
<th>$s_2/C$</th>
<th>$s_2/D$</th>
<th>$X'$ Knowledge</th>
<th>$Y'$ Knowledge</th>
<th>$Z'$ Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$X/R_1$</td>
<td>$1_{R_1}$</td>
<td></td>
<td></td>
<td></td>
<td>$s_1/A = s_1/B = 0$</td>
<td>$s_1/C = s_2/D = 0$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$Y/R_2$</td>
<td>$1_{R_1}$</td>
<td>$1_{R_2}$</td>
<td>$1_{R_1}$</td>
<td></td>
<td>Valid inf.</td>
<td>$s_1/A = 1$</td>
<td>$s_1/B = 0$</td>
</tr>
<tr>
<td>3</td>
<td>$X/R_3$</td>
<td>$2_{R_1, R_3}$</td>
<td>$1_{R_2}$</td>
<td>$1_{R_3}$</td>
<td>$1_{R_1}$</td>
<td></td>
<td>Valid inf.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$X/R_4$</td>
<td>$1_{R_3}$</td>
<td></td>
<td></td>
<td></td>
<td>$s_1/A = s_1/B = 1$</td>
<td></td>
<td>$s_1/C = 0; s_2/D = 1$</td>
</tr>
<tr>
<td>5</td>
<td>$Z/R_5$</td>
<td>$1_{R_2}$</td>
<td></td>
<td></td>
<td></td>
<td>not valid inf.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$X/R_6$</td>
<td>$2_{R_1, R_3}$</td>
<td>$2_{R_4, R_6}$</td>
<td>$2_{R_1, R_6}$</td>
<td>$s_1/A = s_1/B = 0$</td>
<td></td>
<td></td>
<td>$s_1/A = 1$</td>
</tr>
</tbody>
</table>

Cooperative placement strategies

Cooperative placement strategies are based on the organization of the elements of a system in groups that collaborate to get a common objective. In our case the objective is to carry out global load sharing by means of the load sharing among the groups of Reactive agent. This organization has the advantage of having the possibility of reducing:

- the number of messages exchanged between Allocator agents
- the quantity of information to manage at the level of each Allocator agents
- the overhead associated with the placement algorithm.

The idea of organizing processes in groups has been implemented on different systems such as Amoeba [20], PVM [1], etc. and has proved to be a powerful mechanism for reducing the complexity in distribution costs. Arranging the Allocator agents in groups can be done as a function of the geographical distribution of the Allocator agents, in which case it is termed geographical clustering. If they are organized as functions of the characteristics of Allocator agents, for example, in terms of the services they offer, their homogeneity, etc., then the term virtual clustering is used. Each of these methods of organizing the Agent’s communities have their advantages and disadvantages. The problem is that for each type of application, there is a different organization.
Task Allocation Strategies:...

Strategy when AR are organized in groups of collaborators

Our objective of grouping Reactive agent is carry out global load sharing by means of the load sharing among the groups of Reactive agent formed. We organize Reactive agent in virtual groups by service type. The results we look for are the three described before and the establishment of a boundary which assists the satisfaction of the locality criterion described before.

Organizing the Reactive agent community in this way, necessitates a management apparatus for each group. In this case, our approach associates an administrator to each group (type) of Reactive agents. The Allocator agents can address it to obtain useful data for the allocation of information. The administrator updates its information about the elements of the group by periodically sending them "probe" requests.

The algorithm an Allocator agent executes when it receives a task is illustrated in Procedure 2. Firstly, the Allocator agent tries to allocate the task with the information it has. If it's information is not enough, it ask the manager of the group fulfilling the service necessitated the information it requires. If ever it doesn't knows the group manager, it obtain this information by means of a bidding process. The drawback of this algorithm is that managers' information is necessitated.

Procedure 2 Allocator agent's allocation mechanism

Case event of 

\[
\text{Task T :}
\]

\[
\text{For each subtask } t \in T \text{ do}
\]

\[
\text{If local information is useful to allocate the subtask}
\]

\[
\text{allocate(t);}
\]

\[
\text{Else}
\]

\[
\text{If the manager of the required service is known}
\]

\[
\text{negotiate the service with the manager;}
\]

\[
\text{allocate(t);}
\]

\[
\text{update knowledge;}
\]

\[
\text{Else \{ * start the negotiation to get the service *}
\]

\[
\text{RFB(service);}
\]

\[
\text{evaluate(offers);}
\]

\[
\text{allocate(t);}
\]

\[
\text{update knowledge; * about the manager and services *}
\]

\[
\text{Load : ...}
\]

Example 2.

We resume the same series of requests $R_1$, $R_2$, $R_3$, $R_4$, and $R_5$ as well as services $s_1$ and $s_2$ used in Example 1. There exist 5 sites A, B, C, D, and E. The service $s_1$ is offered by Reactive agents on sites A and B while service $s_2$ is offered by Reactive agents on sites C, D and E. Reactive agents are regrouped according to their type of service and therefore there exists managers for service $s_1$ and for service $s_2$. We also add another aspect concerning the duration of validity of the information that is comprised of two units of time. Request $R_1$, $R_3$ and $A_5$ are launched by the Agent on the site $X$, the request $R_2$ by the Agent on the site $Y$ and the request $R_4$ by Agent on $Z$.

An execution of the different requests is illustrated in Table 2. To simplify our example, the quality for each service is represented by a simple value. At time $t = 1$, the Allocator agent of site $X$ starts request $R_1$ that calls services $s_1$ and $s_2$. By asking service managers $M_{s_1}$ and $M_{s_2}$, it learns that sites A and B
propose the same quality of service for \( s_1 \) while \( C \) and \( D \) propose the same for \( s_2 \). Without preference, \( R_1 \) is placed on sites \( A \) and \( C \). At time \( t = 4 \), the duration of the validity of knowledge being fixed at two units, the knowledge acquired of site \( X \) by the Allocator agent is expired. At time \( t = 5 \), request \( R_4 \) that simply uses service \( s_1 \) is started by \( Z \). At this stage, the knowledge of the Allocator agent of site \( Z \) has allowed the favorable placement of the request on site \( B \). The placement of these requests on the different Reactive agents necessitates four broadcasts, three multicasts and 29 point-to-point communications.

Table 2: Execution of the sequence \( R_1, R_2, R_3, R_4, R_1 \) and \( R_5 \)

<table>
<thead>
<tr>
<th>temp</th>
<th>Site/Req.</th>
<th>( s_1/A )</th>
<th>( s_1/B )</th>
<th>( s_2/C )</th>
<th>( s_2/D )</th>
<th>( X' ) Knowledge</th>
<th>( Y' ) Knowledge</th>
<th>( Z' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>( X/R_1 )</td>
<td>( 1_{R_1} )</td>
<td>( 1_{R_1} )</td>
<td></td>
<td></td>
<td>( s_1/A = s_1/B = 0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td>( Y/R_2 )</td>
<td>( 1_{R_1} )</td>
<td>( 1_{R_2} )</td>
<td>( 1_{R_1} )</td>
<td></td>
<td>Valid Inf.</td>
<td>( s_1/A = 1 )</td>
<td>( s_1/B = 0 )</td>
</tr>
<tr>
<td>t3</td>
<td>( X/R_3 )</td>
<td>( 2_{R_1,R_3} )</td>
<td>( 1_{R_3} )</td>
<td>( 1_{R_2} )</td>
<td>( 1_{R_3} )</td>
<td>Valid Inf.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t4</td>
<td>( X/R_1 )</td>
<td>( 1_{R_3} )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 1_{R_3} )</td>
<td>not valid inf.</td>
<td>not valid inf.</td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td>( Z/R_4 )</td>
<td>( 1_{R_3} )</td>
<td>( 1_{R_4} )</td>
<td></td>
<td></td>
<td>not valid inf.</td>
<td>not valid inf.</td>
<td></td>
</tr>
<tr>
<td>t6</td>
<td>( X/R_1 )</td>
<td>( 2_{R_1,R_3} )</td>
<td>( 2_{R_4,R_5} )</td>
<td>( 2_{R_1} )</td>
<td>( 1_{R_3} )</td>
<td>( s_2/C = 0 )</td>
<td>( s_2/D = 1 )</td>
<td></td>
</tr>
</tbody>
</table>

Allocation Strategy using Virtual Groups

To allow knowledge sharing between Allocator agents, we organize them in groups. Each Allocator agent knows the elements of its group. Thus, when an Allocator agent does not know how to place a request due to lack of knowledge, it addresses a request to other Allocator agents in the same group to enrich its knowledge. If, after such knowledge enrichment, it still does not know how to place the request, it initiates a negotiation process.

---

Procedura 3 Allocator agent’s allocation mechanism

Case event of 

Task T : "task allocation request"

For each subtask \( t \in T \) do

If local information is useful to place the subtask

allocate(t);

Else { "the agent tries to obtain information from its friend"

enrich-knowledge(t, Friends);

If the enriched information is useful to allocate t

allocate(t);

Else { "start the negotiation to get the service"

RFB(service);

evaluate(offers);

allocate(t);

update knowledge; "about the manager and services"
}

Load : " a reply containing information about a reactive agent"

---
5 Sub-contractor Agents

In multi-media information systems, to be able to offer complex added value services, it is necessary to allow the composition of services, that is, allow the possibility of building a service from other less complex ones offered by Reactive agents. An instance of a complex service is the bibliographical search that is derived from a translator, a localization server, a gatherer and a format translator. We introduce a new type of agent we call Sub-contractor agents which offer complex services. Sub-contractor agents have the same behaviour as Reactive agent because they offer a service (even which, although complex, are still services). Also, the behaviour of an Allocator agent because it negotiate the services it necessitate to offer it’s service. (Figure 5 shows some of the information a Sub-contractor keeps about the system.

Sharing the knowledge among Sub-contractors is useful, because not all the Reactive agent they use have the same information validity time. Thus a communication amid Sub-contractors can be enough to obtain missing information about the behaviour of Reactive agent, when an allocation decision is necessary to be taken. The results obtained in this way have the following characteristics:

- the time of response of a Sub-contractor is lower because the communication is at the group level, not at the system level (a multicast communication replaces a broadcast communication).
- the quality computed may not be the best on the system.

![System and Own Knowledge of a Sub-contractor](image)

Figure 5: A Sub-contractor agent keeps information allowing the delivery of a service planning a complex task. In this illustration, the name of the service is bibliographical search, necessitating the services of a bibliographic database and text processing.

Interactions between the different agents are illustrated by the Figure 6. The algorithm used by Sub-contractor agents to place requests is briefly developed below. The algorithm implemented is shown in Procedure 4.

Procedure 4 Sub-contractor’s allocation mechanism

Case event of {
    RFB :
        For each subtask $t \in T$ do
            If own information is useful to allocate or subcontract the subtask
                compute bid;
                return(bid);
Else {
    ask for useful information to its collaborators
    If received information is enough
        compute(bid)
        return(bid);
    Else
        For each missing service{
            start a negotiation
            If all services are obtained
                compute(bid);
                update information
                return(bid)
            Else
                delivers a diagnostic message
        }
    }
}

Task: allocate(task)

Example 3

We consider three services $s_1$, $s_2$, $s_3$ and $s_4$ offered by the Reactive agents of five sites $A$, $B$, $C$, $D$ and $E$, $s_1$ by $A$ and $B$, $s_2$ by $C$ and $E$ and $s_3$ by $F$. We also consider three Sub-contractor agent types $SS_1$, $SS_2$ and $SS_3$, each one proposing an added value service composed from $s_1$, $s_2$ and $s_3$. The Sub-contractor type $SS_1$ using $s_1$ and $s_2$ is available on sites $A$, $B$ and $E$; The Sub-contractor type $SS_2$ using $s_2$ is available on sites $C$ and $D$, while the Sub-contractor type $SS_3$ using $s_1$ and $s_3$ is available on site $C$. The validity duration of the information is a function of the service and is one for services $s_1$ and $s_2$, and three for service $s_3$. Three complex requests $R$, $S$ and $T$ are considered, $R$ accessing Sub-contractors types $SS_1$ and $SS_2$, $S$ accessing $SS_2$ and $SS_3$ while $T$ accesses $SS_2$.

An execution of these requests is illustrated in Table 3 which shows the evolution of knowledge obtained
from the different sites. At time $t = 1$, the request $R$ making a call to Sub-contractor types $SS_1$ and $SS_2$ is launched by an Agent on site $X$. The former, having no knowledge, makes a request-for-bids (RFB) beside Sub-contractor types $SS_1$ and $SS_2$ on all sites. Sub-contractors, due to lack of knowledge, also start a (RFB) beside their Reactive agents so as to complete their knowledge. Knowledge obtained by the Allocator agent on site $X$ allows it to place the request on sites $A$ and $C$ without preference. Sites $A$ and $C$ allocate the work on services $s_1$, $s_2$. At time $t=2$, request $S$ launched by an Allocator agent on site $Y$ also executes a RFB beside Sub-contractors types $SS_1$ and $SS_2$. These Sub-contractor types, having acquired knowledge, no longer need to get information beside their service suppliers. Request $S$ is placed on sites $C$ and $D$. At time $t = 3$, request $R$ is ended while $T$ is launched by an Agent on site $Z$. Not having knowledge, the Allocator agent of site $Z$ makes a RFB beside Sub-contractors serving $SS_2$. In our example Sub-contractors serving $SS_2$ have knowledge partial about the Reactive agent necessitated. Thus firstly they share their information trying to take a decision, but finally they get information beside the Reactive agent, and the request $Z$ is placed on site $D$.

Table 3: Execution of the sequence $R_1$, $R_2$, $R_3$, $R_4$, $R_1$ and $R_5$

<table>
<thead>
<tr>
<th>Time</th>
<th>Requester</th>
<th>$SS_1/A$</th>
<th>$SS_1/B$</th>
<th>$SS_1/E$</th>
<th>$SS_2/C$</th>
<th>$SS_2/D$</th>
<th>$SS_2/E$</th>
<th>Knowledge of $X$</th>
<th>Knowledge of $Y$</th>
<th>Knowledge of $Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RX</td>
<td>$s_1/A = 0$</td>
<td>$s_1/B = 0$</td>
<td>$s_1/E = 0$</td>
<td>$s_2/C = 0$</td>
<td>$s_2/D = 0$</td>
<td>$s_2/E = 0$</td>
<td>$SS_1/A = 0$</td>
<td>$SS_2/B = 0$</td>
<td>$SS_2/C = 0$</td>
</tr>
<tr>
<td>2</td>
<td>SY</td>
<td>$s_1/A = 0$</td>
<td>$s_1/B = 0$</td>
<td>$s_1/E = 0$</td>
<td>$s_2/C = 0$</td>
<td>$s_2/D = 0$</td>
<td>$s_2/E = 0$</td>
<td>$SS_1/A = 0$</td>
<td>$SS_2/B = 0$</td>
<td>$SS_2/C = 0$</td>
</tr>
<tr>
<td>3</td>
<td>RX</td>
<td>$s_1/A = 0$</td>
<td>$s_1/B = 0$</td>
<td>$s_1/E = 0$</td>
<td>$s_2/C = 0$</td>
<td>$s_2/D = 0$</td>
<td>$s_2/E = 0$</td>
<td>$SS_1/A = 0$</td>
<td>$SS_2/B = 0$</td>
<td>$SS_2/C = 0$</td>
</tr>
<tr>
<td>4</td>
<td>RX</td>
<td>$s_1/A = 2$</td>
<td>$s_1/B = 1$</td>
<td>$s_1/E = 1$</td>
<td>$s_2/C = 0$</td>
<td>$s_2/D = 0$</td>
<td>$s_2/E = 0$</td>
<td>$SS_1/A = 0$</td>
<td>$SS_2/B = 0$</td>
<td>$SS_2/C = 0$</td>
</tr>
</tbody>
</table>

6 Conclusion

The agent has recently take as a subject of research in distributed systems, distributed artificial intelligence, information retrieval, among others. This paper tries to contribute to the research of information retrieval, studying the agent approach to improve the response by means of a dynamic allocation of requests. This agent approach is quite recent. Schaerf et al [2] have studied the interaction of various parameters and their effect on the system efficiency. In Arcadia [6], dynamic placement is mainly based on the cooperation of two agent types, "system agents", and "application agents". Both approaches usually assume some control over the system, in the sense that it is possible to move a process to another site to balance the load. They also rely on several indicators such as the number of messages received on a site. This makes them inapplicable in the context of the Internet. The strategies for dynamic query placement which we have developed in this paper are part of the U-Doc project and assume a worldwide distribution of the information system at Internet scale. They rely on the approach of multiple agents organized in a community of Allocator agents and a community of Reactive agents. In order to optimize dynamic query placement, we mainly use the knowledge an Allocator agent obtains during its experiences and also allow their cooperation to share information. The response time and the throughput are the only parameters that the Allocator agents use to build their knowledge through their experiences. In order to enable complex services with large added value, Sub-contractor agents are also proposed. We have developed these strategies of allocation taking into account the actual conditions of our project. Today algorithms are being tested by means of a simulation. The results of our simulations will undergo actual validation in our U-Doc project.
References


Task Allocation Strategies: ...


