Decentralized Decision Making Process for Document Server Networks

Aurélie Beynier
LIP6 - University Paris 6
104 avenue du Président Kennedy
75016 Paris, France
aurelie.beynier@lip6.fr

Abdel-Illah Mouaddib
GREYC - University of Caen
Bd. Maréchal Juin
14032 Caen Cedex, France
mouaddib@info.unicaen.fr

***Abstract*— A peer-to-peer server network system consists of a large number of autonomous servers logically connected in a peer-to-peer way where each server maintains a collection of documents. When a query of storing new documents is received by the system, a distributed search process determines the most relevant servers and redirects the documents to them for processing (compressing and storing at the right document base).**

In this paper, we model this distributed search process as a distributed sequential decision making problem using a set of interactive Markov Decision Processes (MDP), a specific stochastic game approach, which represent each server's decision making problem. The relevance of a server to a document is regarded as a reward considering the capacity of the storage and the goodness score of a server. We show that using a central MDP to derive an optimal policy of how to distribute documents among servers leads to high complexity and is inappropriate to the distributed nature of the application. We present then interactive MDPs approach transforming this problem into a decentralized decision making process.

I. Introduction

The Web is real-world application of distributed environment where various entities are massively distributed and loosely connected. Documents become the basic element to manage in large document bases. Consequently their storage and search problems should be carefully considered to maintain a performant system. To alleviate these problems, distributed decision making systems formalize the problem of storing the huge web documents to a large number of autonomous local decision makers (servers) where each of the decision makers (servers) decides about how to store and how to retrieve documents [7], [8].

In this paper, we adopt a peer-to-peer distributed document storage and search server network, in which each server is autonomous and connected to only a small number of peer servers. The resource consumption for each document storage is probabilistic because of the uncertain output of the compression techniques. However, we assume that each storage technique has a probability distribution on resource consumptions (memory). To optimally store a set of documents, the system should decide how to distribute the documents among the servers. We present, first, how this decision making process can be formalized as an MDP from which a global optimal policy of servers can be derived. It, also, will be shown that using a centralized MDP is computationally high and unsuitable because of the distributed nature of the problem. Consequently, we present another approach based on a specific distributed decision making technique.

The distribution of the document decision making problem is, then, formalized as a *decentralized decision making process, a specific stochastic game approach*, among servers where each server should make a decision on which documents to store considering its capacity, the relevance of the documents and the ability of the other servers to store it. In order to model this problem, we consider a *set of interactive MDPs for a cooperative game theoretic model* where each server uses an MDP to compute its local policy which selects documents to store considering the abilities of the other agents.

The system we consider is a set of servers $\mathsf{S} = \{a_1, \dots a_n\}$ and queries **q** of searching and storing a set of documents $\{d_1, \dots d_T\}$. The problem is then how to optimally distribute documents d_t among servers a_j such that the resource consumption of the system is minimized and the relevance of documents to server is maximized.

To deal with this problem we transform it into a problem of task allocation in a multi-agent system [5], [6], [3], [1], *a specific problem of distributed network resource allocation*, where the set of servers becomes a set of agents (or players) A and the set of documents to store become a set of tasks **T**.

II. Framework for storage tasks

The problem of distributing documents to store among servers can be formalized as a decentralized cooperative system which consists of a set of agents $\mathbf{A} = \{a_1, a_2, \cdots, a_n\}$ that have to allocate and to execute a set of tasks $\mathbf{T} = \{d_1, d_2, \cdots, d_T\}$. Each agent a_k has a *bounded quantity of resources* R_{a_k} that he uses to execute his allocated tasks. Tasks will be allocated and executed in a commonly known order: without loss of generality, we assume that this ordering is $d_1, d_2, \cdots, d_T$. A task allocation is a mapping $\mathcal{P}$ from tasks to agents. It can be expressed by $\mathcal{P} = \{D^1, D^2, \cdots, D^n\}$, where D^k is the set of tasks allocated to the agent a_k. The sets D^k verify: $\forall k \neq l, D^k \cap D^l = \emptyset$ and $\bigcup_{a_k \in A} D^k \subseteq \mathbf{T}$.

Definition 1: Each agent a_k has a reward function G_{a_k}: $\mathbf{T} \rightarrow \mathtt{R}^+$ such as, $G_{a_k}(d_t)$ represents what the system gains if a_k stores the document d_t.

This function in the context of Peer-to-Peer document storage server architecture considers many criteria such as the memory capacity, the relevance, security and other criteria. In our context, we limit ourselves to memory capacity and the relevance of the server. To do that, we consider the goodness score of a server [2], [8] which assesses the relevance to store a document in comparison with the ones already stored. If the terms of the document to store appear frequently in these documents, then we believe the server is more relevant to the document. In order to avoid considering all the terms of a document, we use a relative *term frequency* according to the greatest frequency in the document. We call this frequency the local index $LI(x, d_t)$ of a term x in a document d_t. More formally:

$$LI(x, d_t) = \frac{|tf(x, d_t) - \max_{y \in d_t} tf(y, d_t)|}{\max_{y \in d_t} tf(y, d_t)}$$

where $tf(x, d_t)$ is the term frequency of term x in document d_t. This index allows us to consider only terms with LI greater than a threshold th (50% for example).

The relevance of a server to a term t is computed as a sum of LI values of all the documents d_t in the server a:

$$Relevance(t, a) = \sum_{d_t \in a} LI(t, d_t)$$

The relevance of the server to a document d_t is then:

$$Relevance(d_t, a) = \sum_{t \in d_t, LI(t,d_t) > th} Relevance(t, a)$$

This relevance is then used as the reward function gained by an agent (server) a_k when it stores a document d_t.

$$G_{a_k}(d_t) = Relevance(d_t, a_k) = \sum_{t \in d_t, LI(t,d_t) > th} \sum_{d_t \in a_k} LI(t, d_t)$$

In this paper, we will be concerned not only with the relevance but also with the *utility* to store a document d_t in a server. This function will consider in addition to the relevance of the server, its capacity of storing new documents which is its available memory resource. The utility function W_{a_k} is, then, given by:

$$W_{a_k}(d_t) = G_{a_k}(d_t) + Available\_resource(a_k)$$

the function *Available_resource* gives the available resource of the server which is an observable information.

The uncertainty on task execution (resource consumption) can be expressed by the fact that an agent a_k cannot exactly determine the quantity of resources which will be consumed when he executes a task d_t. So, he does not know whether he could execute all tasks allocated to him or he will be obligated to ignore ones. To deal with this uncertainty, we use a discrete representation of the resource consumptions for each agent:

Definition 2: The execution of a task d_t by an agent a_k consumes one of p quantities of resources: $r_k^1, r_k^2, \cdots, r_k^p$. The value of p can vary with the agent and the task.

Definition 3: Each agent a_k has a probability distribution $Pr_{a_k} : \{r_k^j : d_t \in \mathbf{T}, j = 1, ..., p\} \longrightarrow]0, 1]$, where $Pr_{a_k}(d_t, r_k^j)$ expresses the probability that the execution of the task d_t by a_k consumes the quantity of resources r_k^j. With a higher value of p, the agent handles the uncertainty in a better way. The different quantities $r_k^j, k = 1, ..., n, j = 1, ..., p$ and the probability distribution Pr_{a_k} are the knowledge of a_k about his environment. The agents' aim is then to allocate tasks (respond to query) in a way which maximizes the system expected reward. We formalize this specific allocation of tasks as a MDP [3] from which a task allocation policy is derived.

III. Document storage Task as an MDP

The problem of distributing documents among servers is formalized as a sequential decision making process where at each step, we decide to which server the document should be allocated. We define the state of the system by the set of documents stored in each server and by the available resources (available memory of servers). In centralized systems, the controller constructs and solves one MDP in order to obtain an optimal distribution of documents among servers. In the following, we describe the MDP[1] via: (1) the states, (2) the actions, (3) the transition model and (4) the expected reward.

A. State Representation

A state represents a situation of the *task allocation* and of the *anticipation of resource consumption* for all agents. We denote by $S_t = ((D_t^1, R_t^1), ..., (D_t^n, R_t^n))$ the state of the system at step t, where: D_t^k is the set of tasks allocated to a_k up until step t (set of stored documents at a server). The sets D_t^k satisfy the conditions:
$\bigcup_{k=1}^{n} D_t^k \subseteq \{d_1, ..., d_t\}$ and $\forall k \neq l$ then $D_t^k \cap D_t^l = \emptyset$
A task $d_{j \leq i}$ belongs to D_t^k if there were enough resources to execute it when the decision to allocate it to a_k has been made (see next section). The construction of the MDP starts from the initial state: $S_0 = ((\emptyset, R_0^1), ..., (\emptyset, R_0^n))$. The system arrives to a terminal state S_T when a decision has been made for all tasks or when agents have no more resources to store new documents. More formally, $S_T = ((D_T^1, R_T^1), ..., (D_T^n, R_T^n))$ where $\bigcup_{k=1}^{n} D_T^k \subseteq T$ and $R_T^k \geq 0, k = 1, \cdots, n$.

B. Actions and Transition Model

An action $\mathbf{E}(S_{t-1}, d_t, a_k)$ consists in allocating, from S_{t-1}, a task d_t to an agent a_k (allocating a document to a server) and in anticipating the amount of resources which will be consumed when executing this task. For simplicity, let this action be $\mathbf{E}(d_t, a_k)$. When it is applied to $S_{t-1} = ((D_{t-1}^1, R_{t-1}^1), ..., (D_{t-1}^n, R_{t-1}^n))$, the system moves to one of the p following states: $S_t^j = ((D_t^1, R_t^1), ..., (D_t^k, R_t^k = R_{t-1}^k - r_k^j), ..., (D_t^n, R_t^n))$ where $j = 1, \cdots, p$, $D_t^l = D_{t-1}^l$ and $R_t^l = R_{t-1}^l, \forall l \neq k$.

[1]The MDP described here is based on the MDP presented in [3].

- If $r_k^j \leq R_{t-1}^k$ then $D_t^k = D_{t-1}^k \cup \{d_t\}$ and $R_t^k = R_{t-1}^k - r_k^j$ (agent a_k has enough resources to store the document).
- Otherwise ($r_k^j > R_{t-1}^k$), $D_t^k = D_{t-1}^k$ and $R_t^k = 0$ (agent a_k lacks resources).

In fact, there are, at most, p possible states because the execution of d_t can consume one of p different quantities of resources (Def. 2). When $r_k^j > R_{t-1}^k$, the agent will consume all his resources (R_{t-1}^k) without achieving d_t. So, this task does not belong to D_t^k. However, arriving to state S_t^j, where $r_k^j \leq R_{t-1}^k$, means that a_k has executed d_t consuming the quantity r_k^j of resources and the system will gain $W_{a_k}(d_t)$. The transition probability is expressed by the distribution Pr_{a_k} because a_k reaches state S_t^j if the execution of d_t consumes the quantity r_k^j. In the other words, the probability to reach a state S_t^j is $Pr_{a_k}(d_t, r_k^j)$.

C. Expected Reward and Global Optimal Policy

The decision to apply an action depends on the reward that the system expects to obtain by applying this action. An allocation policy π is a mapping from states into actions: for each state S_{t-1}, $\pi(S_{t-1}) = \mathbf{E}(d_t, a_k)$ means that being in state S_{t-1} the action $\mathbf{E}(d_t, a_k)$ will be applied.

We denote by $V(\mathbf{E}(d_t, a_k))$ the expected reward associated to the action $\mathbf{E}(d_t, a_k)$. Being in a state $S_{t-1}, t = 1, \cdots, T$, the expected reward of a policy $\pi(S_{t-1}) = \mathbf{E}(d_t, a_k)$ is the expected reward obtained by executing π: $V(\mathbf{E}(d_t, a_k))$. We define the expected reward $EV[S_{t-1}]$ associated with the state S_{t-1} as an immediate gain α_{t-1} that the system obtains being in this state ($\alpha_{t-1} = W_{a_k}(d_{t-1})$, if $d_{t-1} \in D_{t-1}^k$, 0 otherwise), accumulated by the expected reward of the policy to come (reward-to-go). An optimal policy $\pi^*(S_{t-1})$ is the one which maximizes the expected reward at each state. This policy is obtained by solving Bellman's equation adapted to our former using value iteration [4]:

- for each nonterminal state S_{t-1}:

$$EV[S_{t-1}] = \alpha_{t-1} + \max_{a_k \in A}\{V(\mathbf{E}(d_t, a_k))\} \quad (1)$$

$$V(\mathbf{E}(d_t, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_t, r_t^k) \cdot EV[S_t^j] \quad (2)$$

- for every terminal state S_T:

$$EV[S_T] = \alpha_T \quad (3)$$

Since the obtained MDP is a finite horizon with no loops[2], the policy guaranteed by Equation (1) is then optimal [4]. Formally,

$$\pi^*(S_{t-1}) = arg(\max_{a_k \in A}\{V(\mathbf{E}(d_t, a_k))\}) \quad (4)$$

[2]Being in a state S_{t-1}, the applicable actions are $\mathbf{E}(d_t, a_k), a_k \in A$ and no one can drive to a state $S_{j \leq t-1}$.

The operator arg returns the action whose expected reward is maximal.

An optimal task allocation $\mathcal{P}^*$ can be obtained from a terminal state such as: $S_T = ((D_T^1, R_T^1), ..., (D_T^n, R_T^n))$ reached by applying the optimal policy at each state starting up from S_0, thus: $\mathcal{P}^* = \{D_T^1, D_T^2, \cdots, D_T^n\}$.

Due to the MDP state space size, allocating tasks can only be performed when the number of tasks and agents is "small" (see Section VI). Since a state $S_{t-1} = ((D_{t-1}^1, R_{t-1}^1), ..., (D_{t-1}^n, R_{t-1}^n))$, in the global MDP, globalizes information (allocated tasks and available resources task) of all agents. Using such a representation while dealing with large sets of servers and documents is not realistic since it quickly limits the size of the problems that can be considered. Furthermore, because of the distributed nature of the problem (peer-to-peer server network), the distribution of the decision making process is more suitable than a centralized process. Indeed, the number of states is exponential in the number of tasks and agents. Besides, the agent who solves the MDP should have precise knowledge about the uncertain behavior of the others. The high complexity of the problem and the distributed nature of the application thus reduce the applicative range of a global MDP for real-life peer-to-peer server architecture. Consequently, we propose to consider a system, more suitable, where each server is an autonomous agent able to make a decision locally about storing or not the document through a local MDP.

In the following, we introduce a new method allowing agents to act in a decentralized way to obtain an optimal task allocation similar to the one obtained by the MDP. We re-formalize the problem of task allocation as decentralized MDPs (decentralized Local-MDP) where each agent solves his local MDP which requires less computation than the global MDP.

IV. Document storage Task as Local-MDPs

A Local-MDP consists of a set $\bar{\mathcal{S}}$ of states, a set of actions $\bar{\mathcal{AC}}$ and a transition model. In a_k's Local-MDP, each state is associated with a reward which represents what the system gains when a_k is in this state. We associate each action with an expected reward which represents what the system expects to gain if a_k applies this action. In the following, we describe the MDP locally developed by an agent a_k.

A. State Representation

A state of $\bar{\mathcal{S}}$ represents a local situation of task allocation and of the anticipation of resource consumption for the agent a_k. We denote by: $\bar{S}_t = (\bar{D}_t^k, \bar{R}_t^k)$ the state of the agent a_k at step t, where $\bar{D}_t^k$ is the set of tasks allocated to a_k up until step t, and $\bar{R}_t^k$ is a_k's available resources. The development of the Local-MDP starts from the initial state: $\bar{S}_0 = (\emptyset, R_0^k)$. The agent arrives to a terminal state $\bar{S}_T$ when all tasks have been allocated or when he has no more sufficient resources to execute other tasks. Note that a_k's Local-MDPs state space does not contain any information about the states of the other agents. This loss of information will be covered (see

next Section) using a specific action $e_\emptyset$ which leads to some interactions between agents.

B. Actions and Transition Model

The action $\mathbf{e}(\bar{S}_{t-1}, d_t, a_k)$ consists in the allocation of a task d_t to a_k and in the anticipation of the resource quantity which will be consumed when executing d_t. The action $\mathbf{e}_\emptyset(\bar{S}_{t-1}, d_t, a_k)$ consists in ignoring the task d_t by a_k. It represents, in fact, the allocation of d_t to another agent $a_l \neq a_k$. We introduce this action in order to represent what happens in the Local-MDPs of the other agents when a_k ignores the task d_t. For simplicity, we denote $\mathbf{e}(\bar{S}_{t-1}, d_t, a_k)$ and $\mathbf{e}_\emptyset(\bar{S}_{t-1}, d_t, a_k)$ by $\mathbf{e}(d_t, a_k)$ and $\mathbf{e}_\emptyset(d_t, a_k)$. Let a_k's current state be $\bar{S}_{t-1} = (\bar{D}^k_{t-1}, \bar{R}^k_{t-1})$, then the application of the action $\mathbf{e}(d_t, \mathbf{a_k})$ drives the agent a_k to one of the p following states: $S^j_t = (\bar{D}^k_t, \bar{R}^k_t)$, where $j = 1, ..., p$.

- If $r^j_k \leq \bar{R}^k_{t-1}$, $\bar{D}^k_t = \bar{D}^k_{t-1} \cup \{d_t\}$ and $\bar{R}^k_t = \bar{R}^k_{t-1} - r^j_k$.
- Otherwise ($r^j_k > \bar{R}^k_{t-1}$), $\bar{D}^k_t = \bar{D}^k_{t-1}$ and $\bar{R}^k_t = 0$.

When $r^j_k > \bar{R}^k_{t-1}$, the agent will consume all his resources ($\bar{R}^k_{t-1}$) without achieving d_t. So, this task does not belong to $\bar{D}^k_t$. However, arriving to a state $\bar{S}^j_t$, where $r^j_k \leq \bar{R}_{t-1}$, means that d_t has been achieved and the reward $W_{a_k}(d_t)$ is obtained. Furthermore, the transition probability of the action $\mathbf{e}(d_t, a_k)$ is expressed by the distribution Pr_{a_k} because a_k reaches a state $\bar{S}^j_t$ if the execution of d_t consumes the quantity r^j_k. On the other hand, the application of the action $\mathbf{e}_\emptyset(d_t, \mathbf{a_k})$ drives to state $\bar{S}_t = (\bar{D}^k_t = \bar{D}^k_{t-1}, \bar{R}^k_t = \bar{R}^k_{t-1})$ with probability 1 (no resource consumption). This state summarizes, in a_k's Local-MDP, all states generated, in the MDP, by actions $\mathbf{e}(d_t, a_{l\neq k}), a_l \in A$.

C. Distributed expected reward and joint optimal policy

We define the reward $EV[\bar{S}_{t-1}]$ and the expected rewards associated to each action by a similar way as in the global MDP. We denote by $V(\mathbf{e}(d_t, a_k))$ and $V(\mathbf{e}_\emptyset(d_t, a_k))$ the expected rewards associated to the action $\mathbf{e}(d_t, a_k)$ and to the action $\mathbf{e}_\emptyset(d_t, a_k)$, respectively at a state $\bar{S}_{t-1}$. Being in a state $\bar{S}_{t-1}$, a policy to follow is an action $\mathbf{e}(d_t, a_k)$ or $\mathbf{e}_\emptyset(d_t, a_k)$ to apply. The expected reward of a policy is the expected reward of the corresponding action. The reward $EV[\bar{S}_{t-1}]$ is defined as an immediate gain $\bar{\alpha}_{t-1}$ that the system obtains if a_k is in the state $\bar{S}_{t-1}$ accumulated by the expected reward of the used policy. The immediate gain $\bar{\alpha}_{t-1}$ to be in state $\bar{S}_{t-1}$ is defined as follows:

$\bar{\alpha}_{t-1} = W_{a_k}(d_{t-1})$ if $d_{t-1} \in \bar{D}^k_{t-1}$, 0 otherwise.

In the following, we formulate the reward $EV[\bar{S}_{t-1}]$ and the expected rewards $V(\mathbf{e}(d_t, a_k))$ and $V(\mathbf{e}_\emptyset(d_t, a_k))$ using Bellman's equations and value iteration:

- for each nonterminal state $\bar{S}_{t-1}$:

$$EV[\bar{S}_{t-1}] = \bar{\alpha}_{t-1} + \max\{V(\mathbf{e}(d_t, a_k)), V(\mathbf{e}_\emptyset(d_t a_k))\} \quad (5)$$

$$V(\mathbf{e}(d_t, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_t, r^j_k) \cdot EV[\bar{S}_t] \quad (6)$$

$$V(\mathbf{e}_\emptyset(d_t, a_k)) = \max_{a_l \neq a_k} \{V(\mathbf{e}(d_t, a_l))\} \quad (7)$$

- for every terminal state $\bar{S}_T = (D^k_T, R^k_T)$:

$$EV[\bar{S_T}] \ = \bar{\alpha}_T \quad (8)$$

Agent a_k uses Equation (5) to determine the reward of state $\bar{S}_{t-1}$. For that, he locally calculates the immediate gain $\bar{\alpha}_{t-1}$ using the definition described above, and the expected value $V(\mathbf{e}(d_t, a_k))$ given in Equation (6). However, expected value $V(\mathbf{e}_\emptyset(d_t, a_k))$ given in Equation (7) is calculated by receiving $V(\mathbf{e}(d_t, a_l)), a_l \neq a_k$ from the other agents via communication. On the other side, each agent $a_l \in A$ calculates $V(\mathbf{e}(d_t, a_l))$ from his Local-MDP and sends it to the others. Since the obtained Local-MDP is a finite horizon with no loops, the policy guaranteed by Equation (5) is then locally optimal and it leads to a global optimality as it will be shown in Section V. Formally,

$$\pi^*(\bar{S}_{t-1}) = arg(\max\{V(\mathbf{e}(d_t, a_k)), V(\mathbf{e}_\emptyset(d_t, a_k))\}) \quad (9)$$

D. Coordination of distributed value calculation

As we have shown above, Equation (7) generates communication between agents. Indeed, an agent a_l calculates $V(\mathbf{e}(d_t, a_l))$ using Equation (6) and then sends it to the other agents. In order to obtain an optimal policy $\pi^*(\bar{S}_{t-1})$, agents exchange the values $V(\mathbf{e}(d_t, a_l)), a_l \in A$ that needs an exchange of the values $V(\mathbf{e}(d_{t+1}, a_l)), a_l \in A$, and so on. At the end, agent a_k can directly calculate $V(\mathbf{e}(d_T, a_k))$ using Equation (8). After exchanging these values, the reward $EV[\bar{S}_{T-1}]$ can be calculated for any state $\bar{S}_{T-1}$. By backward chaining, $EV[\bar{S}_{T-1}]$ allows to calculate $EV[\bar{S}_{T-2}]$, and so on up to state $\bar{S}_{t-1}$. Algorithm 1 describes how an agent a_k derives his local policy from his local MDP.

$\mathbf{COM}(V(\mathbf{e}(d_{t+1}, a_k)), \{V(\mathbf{e}_\emptyset(d_{t+1}, a_k)\})$ is a communication primitive allowing a_k to communicate the value $V(\mathbf{e}(d_{t+1}, a_k))$ to agents $a_{j\neq k}$ which uses it for computing $V(\mathbf{e}_\emptyset(d_{t+1}, a_j))$ and to receive from them values $V(\mathbf{e}(d_{t+1}, a_j))$ allowing him to compute $V(\mathbf{e}_\emptyset(d_{t+1}, a_k))$.The function $reachable_k(t)$ returns all states $\bar{S}_t$ reached by agent a_k using action $\mathbf{e}(d_t, a_k)$ at step t. $reachable_k(t)$ also describes states reached by agent a_k at step t.

The optimal policies obtained by the agents do not lead to conflicts between them (see next lemma).

Lemma 1: For any state $\bar{S}_{t-1}, t = 1, ..., T$, the optimal policies obtained by the agents according to Equation (9) are consistent. Formally, $\forall a_k \in A, t = 1, ..., T$,

$\pi^*_k(\bar{S}_{t-1}) = \mathbf{e}(d_t, a_k) \Longrightarrow \forall a_l \neq a_k, \pi^*_l(\bar{S}_{t-1}) = \mathbf{e}_\emptyset(d_t, a_l)$.

Proof Based on Equations 7 and 9.

Let $\pi^*_k(\bar{S}_{t-1})$ be $\mathbf{e}(d_t, a_k)$:

$\pi^*_k(\bar{S}_{t-1}) = \mathbf{e}(d_t, a_k) \overset{(eq.(9))}{\Longrightarrow}$

$\mathbf{e}(d_t, a_k) = arg(\max\{V(\mathbf{e}(d_t, a_k)), V(\mathbf{e}_\emptyset(d_t, a_k))\}) \overset{(eq.(7))}{\Longrightarrow}$

$\mathbf{e}(d_t, a_k) = arg(\max\{V(\mathbf{e}(d_t, a_k)), \max_{a_{l\neq k}}\{V(\mathbf{e}(d_t, a_l))\}\})$

$\Longrightarrow V(\mathbf{e}(d_t, a_k)) > V(\mathbf{e}(d_t, a_l)), \forall a_{l\neq k} \in A$

As the values $V(\mathbf{e}(d_t, a_h)), \forall a_h \in A$ are identical in all Local-MDPs (each agent a_h calculates the value of

$V(\mathbf{e}(d_t, a_h))$ and sends it to the others), from the precedent inequality we then have: $\forall a_{l\neq k} \in A$,

$$\mathbf{e}(d_t, a_l) \neq arg(\max\{V(\mathbf{e}(d_t, a_l)), \max_{a_{h\neq l}}\{V(\mathbf{e}(d_t, a_h))\}\})$$

because at least, we have: $V(\mathbf{e}(d_t, a_k)) > V(\mathbf{e}(d_t, a_l))$. Consequently, $\forall a_{l\neq k} \in A$,

$$\mathbf{e}(d_t, a_l) \neq arg(\max\{V(\mathbf{e}(d_t, a_l)), V(\mathbf{e}_\emptyset(d_t, a_l))\})$$
$$\forall a_{l\neq k} \in A, \pi_l^*(\bar{S}_{t-1}) = \mathbf{e}_\emptyset(d_t, a_l)$$

□

Algorithm 1 Modified value iteration algorithm of $\mathcal{M}_{a_k}$:

Require: set of tasks (documents to store)
Ensure: policy π_k^* of agent a_k

for all $\bar{S}_T \in reachable_k(T)$ **do**
  $EV[\bar{S}_T] = \bar{\alpha}_T$
  COM$(EV[\bar{S}_T], \{V(e_\emptyset(\bar{S}_{T-1}, a_k))\})$
end for
for t = T-1 **down to** 1 **do**
  for all $\bar{S}_t \in reachable_k(t)$ **do**
    $EV[\bar{S}_t] = \bar{\alpha}(\bar{S}_t) + \max\{V(\mathbf{e}(d_{t+1}, a_k)), V(\mathbf{e}_\emptyset(d_{t+1}, a_k))\}$
    COM$(V(\mathbf{e}(d_{t+1}, a_k)), \{V(\mathbf{e}_\emptyset(d_{t+1}, a_k))\ \})$
  end for
end for
for t = 1 to T **do**
  for all $\bar{S}_t \in reachable_k(t)$ **do**
    $\pi(\bar{S}_t) = arg(\max(\bar{\alpha}(\bar{S}_t) + \{V(\mathbf{e}(d_{t+1}, a_k)), V(\mathbf{e}_\emptyset(d_{t+1}, a_k))\}))$
  end for
end for
return π

V. An optimal joint policy

In this section we prove that the optimality obtained in the MDP is saved in the coordinated Local-MDPs. Firstly, we show the relationship between the MDP state space and the Local-MDPs' state spaces. Secondly, we prove the equality between the optimal policy obtained by the MDP and the one obtained by the Local-MDPs.

Lemma 2: For each state $S_t((D_t^1, R_t^1), ..., (D_t^n, R_t^n)), t = 0, ..., T$ in the MDP, it exists a state $\bar{S}_t$ in the Local-MDP of each agent a_k, where $\bar{S}_t = (D_t^k, R_t^k), k = 1, ..., n$.

Proof For $t = 0$, the agents are in the initial state $S_0((\emptyset, R_0^1), ..., (\emptyset, R_0^n))$ in the MDP. The couple $(\emptyset, R_0^k), k = 1, ..., n$ is the initial state $\bar{S}_0(\emptyset, R_0^k)$ in a_k's Local-MDP.

Supposing now that the lemma is correct for a value $t-1$: $\forall S_{t-1}((D_{t-1}^1, R_{t-1}^1), ..., (D_{t-1}^n, R_{t-1}^n)) \in$ MDP $k = 1, ..., n, \exists \bar{S}_{t-1} \in a_k$'s Local-MDP, where $\bar{S}_{t-1} = (D_{t-1}^k, R_{t-1}^k)$.

We show in the following that the lemma is also correct for the step t. In fact, every state S_t, in the MDP, is produced by the application of an action $\mathbf{e}(d_t, a_k), a_k \in A$ in a step S_{t-1}. According to the transition model (see Section III-B), the action $\mathbf{e}(d_t, a_k)$ has effects only on the set D_{t-1}^k and on the resources R_{t-1}^k. So, for a state $S_t = ((D_t^1, R_t^1), ..., (D_t^n, R_t^n))$ produced by this action, it exists in the a_k's Local-MDP a state $\bar{S}_t = (D_t^k, R_t^k)$ produced by the action $\mathbf{e}(d_t, a_k)$ in the state $\bar{S}_{t-1} = (D_{t-1}^k, R_{t-1}^k)$, and it exists in the Local-MDPs of the other agents $a_{l\neq k} \in A$ a state $\bar{S}_t = (D_t^l, R_t^l)$ produced by the action $\mathbf{e}_\emptyset(d_t, a_l)$ in the state $\bar{S}_{t-1} = (D_{t-1}^l, R_{t-1}^l)$. □

Lemma 3: For every state $S_{t-1} = ((D_{t-1}^1, R_{t-1}^1), ..., (D_{t-1}^k, R_{t-1}^k), ..., (D_{t-1}^n, R_{t-1}^n))$ in the MDP, the expected reward associated to an action $\mathbf{E}(d_t, a_k), \forall a_k \in \mathcal{A}$ is equal to the one associated to the action $\mathbf{e}(d_t, a_k)$ applied in the state $\bar{S}_{t-1} = (D_{t-1}^k, R_{t-1}^k)$ in a_k's Local-MDP.

Proof We prove this lemma by induction. For the last decision step ($t = T-1$), we are in a state $S_{T-1} = ((D_{T-1}^1, R_{T-1}^1), ..., (D_{T-1}^k, R_{T-1}^k), ..., (D_{T-1}^n, R_{T-1}^n))$ in the MDP, and in the state $\bar{S}_{T-1} = (D_{T-1}^k, R_{T-1}^k)$ in a_k's Local-MDP. According to Equation (2), we have:

$$V(\mathbf{E}(d_T, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_T, r_k^j) \cdot EV[S_T^j] \overset{(eq.(3))}{\Longrightarrow}$$
$$V(\mathbf{E}(d_T, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_T, r_k^j) \cdot \alpha_T^j$$

where α_T^j is the immediate gain of state S_T^j. Since, the agent to whom the task d_T is allocated by the action $\mathbf{E}(d_T, a_k)$ is a_k, then $\alpha_T^j = \bar{\alpha}_T^j$, (definitions of α and $\bar{\alpha}$). Moreover,

$$V(\mathbf{E}(d_T, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_T, r_k^j) \cdot \bar{\alpha}_T^j \overset{(eq.(8))}{\Longrightarrow}$$
$$V(\mathbf{E}(d_T, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_T, r_k^j) \cdot EV[\bar{S}_T^j] \overset{(eq.(6))}{\Longrightarrow}$$
$$V(\mathbf{E}(d_T, a_k)) = V(\mathbf{e}(d_T, a_k)).$$

Supposing now that the lemma is correct for t, formally:
$\forall S_t((D_t^1, R_t^1), ..., (D_t^k, R_t^k), ...(D_t^n, R_t^n)),$
$V(\mathbf{e}(d_{t+1}, a_k)) = V(\mathbf{E}(d_{t+1}, a_k)), \forall a_k \in A$
where $\mathbf{e}$ is applied in the state $\bar{S}_t = (D_t^k, R_t^k)$. We show in the following that the lemma also is correct for $t-1$.

For every state $S_{t-1}((D_{t-1}^1, R_{t-1}^1), ..., (D_{t-1}^k, R_{t-1}^k), ...(D_{t-1}^n, R_{t-1}^n))$, we have:

$$V(\mathbf{E}(d_t, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_t, r_k^j) \cdot EV[S_t^j] \overset{(eq.(1))}{\Longrightarrow}$$
$$V(\mathbf{E}(d_t, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_t, r_k^j) \cdot [\alpha_{\mathbf{t}}^{\mathbf{j}} + \max_{a_l \in A}\{V(\mathbf{E}(d_{t+1}, a_l))\}]$$

As the lemma is supposed correct for t, then:

$$V(\mathbf{E}(d_t, a_k)) = \sum_{j=1}^{p} Pr_{a_k}(d_t, r_k^j) \cdot [\bar{\alpha}_{\mathbf{t}}^{\mathbf{j}} + \max_{a_l \in A}\{V(\mathbf{e}(d_{t+1}, a_l))\}]$$
$$V(\mathbf{E}(d_t, a_k)) = V(\mathbf{e}(d_t, a_k)). \square$$

Lemma 4: The optimal policy obtained in the MDP is equivalent to the optimal policies obtained by the coordinated Local-MDPs, formally: $\forall a_k \in A, t = 1, ..., T,$
$\pi_k^*(\bar{S}_{t-1}) = \mathbf{e}(d_t, a_k) \Longleftrightarrow \pi^*(S_{t-1}) = \mathbf{E}(d_t, a_k)$

Proof Let a_k be the agent verifying that $\pi_k^*(\bar{S}_{t-1}) = \mathbf{e}(d_t, a_k)$ at step $t-1$. According to Equation (9), we have:

$\mathbf{e}(d_t, a_k) = arg(\max\{V(\mathbf{e}(d_t, a_k)), V(\mathbf{e}_\emptyset(d_t, a_k))\}) \overset{(eq.(7))}{\iff}$
$\mathbf{e}(d_t, a_k) = arg(max\{V(\mathbf{e}(d_t, a_k)), \max_{a_{l\neq k}\in\mathcal{A}}\{V(\mathbf{e}(d_t, a_l))\}\})$
$\iff V(\mathbf{e}(d_t, a_k)) > V(\mathbf{e}(d_t, a_{l\neq k})), \forall a_l \in A$
$\overset{(lem.(3))}{\iff} V(\mathbf{E}(d_t, a_k)) > V(\mathbf{e}(d_t, a_{l\neq k})), \forall a_l \in A$
$\iff \mathbf{E}(d_t, a_k) = arg(\max_{a_l \in A}\{V(\mathbf{e}(d_t, a_l))\})$
$\overset{(eq.(4))}{\iff} \pi^*(S_{t-1}) = \mathbf{E}(d_t, a_k)$. □

VI. Performance and Complexity

A. Cost of Computation

We studied the computation time of a problem with different numbers of documents to store (from 10 to 160) and four servers. We recorded the computation time of cases where only one server solves the whole problem (large MDP), two servers (two interactive MDPs), three servers and 4 servers (Figure 1). We observe a gain in computation time when using many servers. The main reason is that the Local-MDP's state space size $|\bar{\mathcal{S}}|$ is defined by: $|\bar{\mathcal{S}}| = \frac{(p+1)^{T+1}-1}{(p+1)-1}$, while the MDP's state space size $|\mathcal{S}|$ is defined by $|\mathcal{S}| = \frac{(n\cdot p)^{T+1}-1}{(n\cdot p)-1}$.

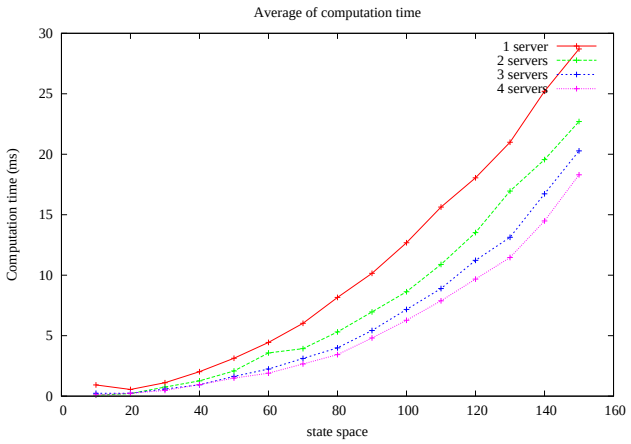

Fig. 1. Comparison of computation time

B. Cost of Communication

Each Local-MDP of each agent a_k sends, at each time t, $(n-1).|reachable_k(t)|$ messages to the others. The total number of messages exchanged between all Local-MDPs is then, $\sum_{k=1}^{n}(n-1).|reachable_k(t)|$. Note that $|reachable_k(t)|$ decreases from step T to step 0 (only one message is sent). The cost of communication is then $\sum_{k=1}^{n}(n-1).|reachable_k(t)|$.Cost(size(message)) where the function *Cost* is the cost of transmitting a message with a given size. The communication cost of the search task is, in the worst case, n since each server sends a message to one server (considered as the most appropriate).

C. Reducing communication

Equation 7 uses all information sent by all the other Local-MDPs. This Equation can be solved partially in a distributed way where each Local-MDP, instead of sending all values of reachable states, sends only the max of them. Then, Equation 7 can be solved using only the maximum values coming from all Local-MDPs and then computes their maximum. By this way, we reduce the number of messages and the size of the messages which contain only the maximum value.

There is no effect on the results of Lemma 1-4. Indeed, instead of applying max operator locally on values coming from different Local-MDPs, we apply max operator only on maximum values coming from the other Local-MDPs: $V(\mathbf{e}_\emptyset(d_t, a_k)) = \max_{j\neq k} V_j^{max}$ where $V_j^{max} = \max_{a\in A_j} V(\mathbf{e}(d_t, a_j)$ computed in a_j's Local-MDP. This processing does not modify the result of Equation 7. Consequently, results of lemma 1-4 remain valid and the cost of communication is reduced to $n.(n-1).Cost(size(message))$.

Other techniques based on approximations can be considered such as sending the minimal values (pessimistic approach) or any combination of max and min values but these techniques have an immediate effect on the optimality since the result of Equation 7 should change and the claims could not be valid in these situations. The evaluation of these approximations will be considered in a short time future work.

Figure 2 shows the reduction of the cost of the communication using more local computation. When servers compute the max values, they reduce the communication cost while the quality of the solution remains optimal. In the same way, the computation time is reduced. Figure 1 shows the computation time results with the reduced mechanism of communication as explained in this section.

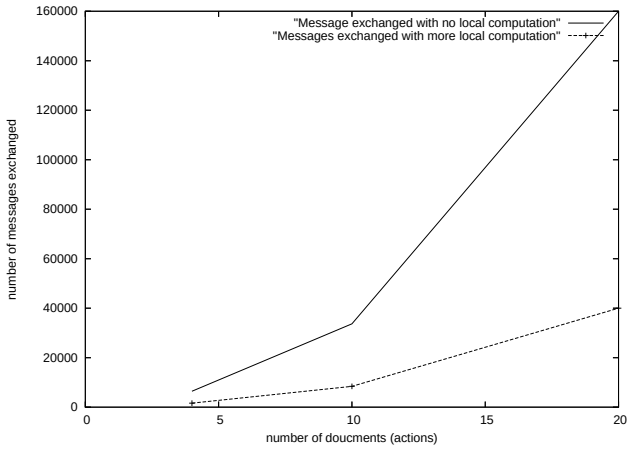

Fig. 2. Reduction of communication cost

The techniques presented above reduces the number of messages but the complexity is $O(n^2)$. This complexity can be reduced by using some specific architectures. A solution is that each server a_i communicates only with its neighbor server a_{i+1} and server a_n communicates with server a_1. This architecture reduces the complexity from polynomial to linear $O(n)$ since the number of messages is $2 \cdot n \cdot T$. This

architecture presents a drawback because servers waste time waiting for the propagation of the max value. Further work, will be concerned with a grid architecture and the effect on the performance of the approach.

VII. Conclusion

In this paper, we presented an approach of applying MDP techniques for the problem of document storage and search using a server network architecture. We discussed the benefit of using a decentralized decision making technique.

This work opens a new direction in using decision-theoretic techniques to control peer-to-peer server network architectures to improve the performance, the security and the robustness of these systems. Further work will also be concerned with the evaluation of this approach to the security and the fault tolerance. We will deal with problems where some servers are unavailable or become less and less safe and documents should be moved automatically to another server.

We also plan to develop a method allowing to reduce both the number of exchanged messages and the state space size. An agent can develop his Local-MDP considering only a subset of agents (but not all agents). Another one considers a different subset of agents. In this case, the problem of the coordination of the obtained task allocations has to be treated.

References

[1] S. Abdallah and V. Lesser. Modeling task allocation using decision theoretic model. In *Proceedings of International Joint conference on Autonomous Agents & Multi-Agent Systems, AAMAS05*, 2005.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[3] H. Hanna and A.I. Mouaddib. Task selection problem under uncertainty as decision-making. In *Proceedings of International Joint conference on Autonomous Agents & Multi-Agent Systems, AAMAS02*, volume 3, pages 1303–1308, 2002.

[4] M. L. Puterman. *Markov Decision Processes*. John Wiley & Sons, New York, 1994.

[5] M.H. Rothkopf, A. Pekec, and R.M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[6] T. Sandholm. Emediator: A next generation electronic commerce server. In *International Conference on Autonomous Agents (AGENTS), Barcelona, Spain*, june 2000.

[7] Y. Shen and D. L. Lee. An mdp-based peer-to-peer search server network. In *IEEE Proceedings of WISE*, 2002.

[8] B. Yuwono and L. Lee. Server ranking for distributed text retrieval systems on the internet. In *Proceedings of Fifth International Conference on Database Systems for advanced applications, DASFAA*, pages 41–50, 1997.