A Reinforcement Learning Solution for Allocating Replicated Fragments in a Distributed Database

Una solución de Aprendizaje Reforzado para ubicar fragmentos replicados en Bases de Datos Distribuidas

Abel Rodríguez Morffi[1], Darien Rosa Paz[1], Marisela Mainegra Hing[2] and Luisa Manuela González González[1]

[1]Departamento de Ciencia de la Computación, [2] Departamento de Matemática
Universidad Central "Marta Abreu" de Las Villas,
Carretera a Camajuaní km. 5.5, C.P. 54830, Santa Clara, Cuba.
Tel: +(53)(42) 281515
arm@uclv.edu.cu, drosa@uclv.edu.cu, marisela@uclv.edu.cu, luisagon@uclv.edu.cu

Article received on April 23, 2007; accepted on October 18, 2007

Abstract
Due to the complexity of the data distribution problem in Distributed Database Systems, most of the proposed solutions divide the design process into two parts: the fragmentation and the allocation of fragments to the locations in the network. Here we consider the allocation problem with the possibility to replicate fragments, minimizing the total cost, which is in general NP-complete, and propose a method based on Q-learning to solve the allocation of fragments in the design of a distributed database. As a result we obtain for several cases, logical allocation of fragments in a practical time.
Keywords: Distributed database design, allocation, replication, reinforcement learning, Q-Learning.

Resumen
Debido a la complejidad del problema de la distribución de los datos, la mayoría de las propuestas de solución presentadas hasta la fecha han coincidido en dividir el proceso de diseño de la distribución en dos fases seriadas: la fragmentación y la ubicación de los fragmentos en los sitios de la red. Este trabajo aborda el problema de ubicación de fragmentos partiendo de un modelo matemático que en su forma general es NP-Completo y propone un método metaheurístico basado en Q-Learning de Aprendizaje Reforzado que minimiza el costo total en un tiempo aceptable. Esta propuesta integra la replicación de fragmentos.
Palabras claves: Diseño de bases de datos distribuidas, ubicación, replicación, aprendizaje reforzado, Q-Learning.

1 Introduction

In the last years there has been an enormous outgrown of distributed information systems. These systems have better adaptability to the necessity of the decentralized organizations due to their capacity to simulate the physical structure of such organizations. Though more advantageous than centralized systems, the distributed ones have some more complexity and their design and implementation represent a challenge (Özsu and Valduriez, 1999).

The design should be so that for the user the distribution is transparent (Ceri and Pelagatti, 1984). The designer must define the distribution of the information in the network maximizing the locality distributing the workload. This problem is known as distribution design and involves two main tasks: fragmenting and allocating (Özsu and Valduriez, 1999).

Various approaches have already been described for the fragment allocation in distributed database systems (Hababeh et al., 2004; Ma et al., 2006). The allocation of the data influences the performance of the distributed systems given by the processing time and overall costs required for applications running in the network. Some allocation methods are limited in their theoretical and implementation parts. Other strategies are ignoring the optimization of the transaction response time. The other approaches present exponential time of complexity and test their performance on specific types of network connectivity. Most of these methods are quite complex, not well understood and difficult to use in a real life. Many assumptions have been done in order to simplify the problem, so

the solutions are applicable in specific conditions. For example, most of the proposed solutions consider that each fragment must be allocated in only one location simplifying the solution space but missing the advantages of replication.

In this paper, we propose a method for allocating database fragments to locations using as a reference the general model by (Özsu and Valduriez, 1999). Finding an optimal fragment allocation in this model is a NP-complete problem since given n fragments and m locations, there will be $(2^m - 1)^n$ different combinations. In our case, it is would very difficult to reach and optimal using an exact method because the problem is computationally very complex. Therefore, this approach proposes a metaheuristic algorithm to aid allocation decision based on Q-Learning from Reinforcement Learning techniques (Watkins, 1999).

2 The allocation problem

According to (Özsu and Valduriez, 1999), the allocation problem considers a set of fragments $F = \{f_1, f_2, ..., f_n\}$, a set of locations $L = \{l_1, l_2, ..., l_m\}$ in a network, and a set of applications $A = \{a_1, a_2, ..., a_q\}$ placed at L. These applications need to access the fragments which should be allocated in the locations of a network. The allocation problem consists on finding an optimal distribution of F over L.

We consider the allocation problem that minimizes the overall cost subject to some constraints as in (Özsu and Valduriez, 1999). Furthermore, here we consider the replication option which makes this problem more complex and it is known to be NP-complete, thus there is not a polynomial time algorithm to solve it.

Let us define the problem in some more details. The decision variable $x_{jk} = 1$ if f_j is stored at location l_k; else it is 0.

Before we derive the cost formulas, some information must be analyzed in advance. That is, the quantitative data about the database, the applications behavior, the locations and network information.

Database information:

- $sel_i(f_j)$ is the number of tuples in f_j that need to be accessed by application a_i.
- $size(f_j) = card(f_j) * length(f_j)$ corresponds to the size of fragment f_j.

Application information:

- RR_{ij} is the number of read accesses of application a_i to fragment f_j.
- UR_{ij} is the number of update accesses of application a_i to fragment f_j.

Two access matrices, UM and RM, that describe the retrieval and update behaviors of all the applications are also needed. The elements u_{ij} and r_{ij} are specified as follows:

- $u_{ij} = 1$ if a_i updates f_j; else it is 0.
- $r_{ij} = 1$ if a_i reads f_j; else it is 0.
- o(i) is the location where application a_i originates.

Location information:

- USC_k is the unitary cost of storing data at location l_k.
- SPC_k is the unitary cost of processing at location l_k.

Network information:

- g_{ij} is the communication cost per frame between locations l_i and l_j.
- fsize is the frame size measured in bytes.

The total cost function has two components: the applications processing cost and the storage cost. It is expressed as follows:

$$TOC = \sum_{\forall a_i \in A} QPC_i + \sum_{\forall l_k \in L} \sum_{\forall f_j \in F} STC_{jk} ,$$

where QPC_i is the processing cost of application a_i, and STC_{jk} is the storage cost of fragment f_j at location l_k. The storage cost is given by:

$$STC_{jk} = USC_k \cdot size(f_j) \cdot x_{jk}$$

For each application a_i, the processing cost is calculated as the cost of processing (PC) plus the transmission cost (TC). Processing costs contains three factors: access costs, integrity enforcement (IE) costs, and concurrency control (CC) costs:

$$PC_i = AC_i + IE_i + CC_i$$

The specification of each cost depends on the used algorithm to make these tasks. The detailed specification of AC would be:

$$AC_i = \sum_{\forall l_k \in L} \sum_{\forall f_j \in F} (u_{ij} \cdot UR_{ij} + r_{ij} \cdot RR_{ij}) \cdot x_{jk} \cdot SPC_k$$

The operational costs of data transmission for update and read-only applications are different. For update applications it is necessary to update all existing replicas and read-only applications just need to access to one of the copies. In addition, at the end of an update operation there is only an update confirmation message, and read-only applications may cause a large amount of data transmission.

The update component of the transmission function is:

$$TCU_i = \sum_{\forall l_k \in L} \sum_{\forall F_j \in F} u_{ij} \cdot x_{jk} \cdot g_{o(i),k} + \sum_{\forall l_k \in L} \sum_{\forall F_j \in F} u_{ij} \cdot x_{jk} \cdot g_{k,o(i)}$$

The cost of the read-only applications is:

$$TCR_i = \sum_{\forall f_j \in F} \min_{l_k \in L} (r_{ij} \cdot x_{jk} \cdot g_{o(i),k} + r_{ij} \cdot x_{jk} \cdot \frac{sel_i(f_j)}{fsize} \cdot length(f_j) \cdot g_{k,o(i)})$$

The constraints are:

- Response time constraint: The execution time of an application must be less or equal than the maximum allowable response time for that application.
- Storage capacity constraint: For each location, the total storage cost of fragments assigned to this location must be less or equal than the capacity of the location.
- Processing capacity constraints: For each location, the total processing cost for all the applications executed at this location must be less or equal than the processing capacity of the location.

3 Solution approaches

There are diverse ways to attack combinatorial optimization problems (Huang and Chen, 2001; Lin and Orlowska, 1995; Ma et al., 2006; March and Rho, 1995; Pérez et al., 2005; Pérez et al., 2004; Pérez et al., 2003; Pérez et al., 2003; Pérez et al., 2002; Wolfson and Jajodia, 1995). These methods include exact and heuristics approaches which have been very useful in solving real life problems. In this section we mention some of them and analyze their applicability.

3.1 Exact methods

The total enumeration is the method that chooses the best solution out of all the possible solutions. A more sophisticated way is partial enumeration, leaving out certain areas of the solution space that for certain do not include any optimal solution, here we can mention Branch and Bound, Cutting planes and Dynamic programming methods.

The main problem with these methods is their applicability to large problems, specifically for the type of NP-complete problems for which there is no guarantee to find an optimal solution in a polynomial time (Lin et al., 1993).

3.2 Heuristics methods

A good alternative for NP-complete combinatorial optimization problems of large size is to find a reasonable solution in a reasonable time (Papadimitriou, 1997). This is the idea of the heuristic methods which are in general quite simple and based on intuitive and common sense ideas. The general problem with many heuristics is that they may get stuck in local optimal solutions. More recently a number of metaheuristics have evolved that define ways to escape local optima. Metaheuristics are higher level heuristics designed to guide other processes towards achieving reasonable solutions. The most widely used metaheuristics are Simulated Annealing, Genetic Algorithms, Tabu Search and GRASP. These methods do not guarantee in general that one will finish with an optimal solution, though some of them present convergence theories. However, they have been successfully applied to many problems.

Here we explore a much more recently approach, Reinforcement Learning. This approach may be interpreted as a conjunction between machine learning and decision making problems.

4 Reinforcement Learning

Reinforcement Learning (RL) is an approach to solve sequential decision making problems that can be modeled by Markov Decisions Processes. The main idea of RL is the continuous interaction between an agent and its environment. Through this interaction the agent tries control actions for the current state of the environment, influencing the next state; depending on the chosen action and the new state, the agent receives a reward/penalization signal. This way, the agent should learn to behave in order to achieve its goal. This approach have been successfully applied to several decision and optimization problems (Abe et al., 2003; Choi et al., 2004; Iida et al., 2004; Morales and Sammut, 2004).

4.1 Basic elements of Reinforcement Learning

An RL-system has mainly two components: the Environment where the process to be studied takes place, and an agent (RL-agent) who should be able to learn to control the process. The Environment in general, is assumed to be a Markov decision process (MDP) which is characterized by states, rewards and transitions (Puterman, 1994). An RL-agent is characterized by the goal, a knowledge structure, a learning method to update its knowledge and a specific behavior (policy).

Figure 1 summarizes the communication between the agent and its environment. At each decision moment, the agent observes the current state (1) of the environment and performs an action (2) selected according to its decision policy. As a result of the received action in the environment (3), a transition to a new state takes place and a reinforcement or reward signal is generated (4). The reward signal and the new state are received by the agent (5) and can be used through the agent's learning method in order to update its knowledge about the environment, and consequently it can update its policy (6). Rewards and state transition functions may be in general stochastic, and the underlying probability distributions are assumed not to be known to the agent.

The problem consists of finding a prescriptive rule to choose an action for each state such that a certain criterion is optimized (the goal); here we consider the minimization of the total expected discounted reward. A discount factor $0<\gamma<1$ corresponds to the idea that rewards are less attractive in the far future.

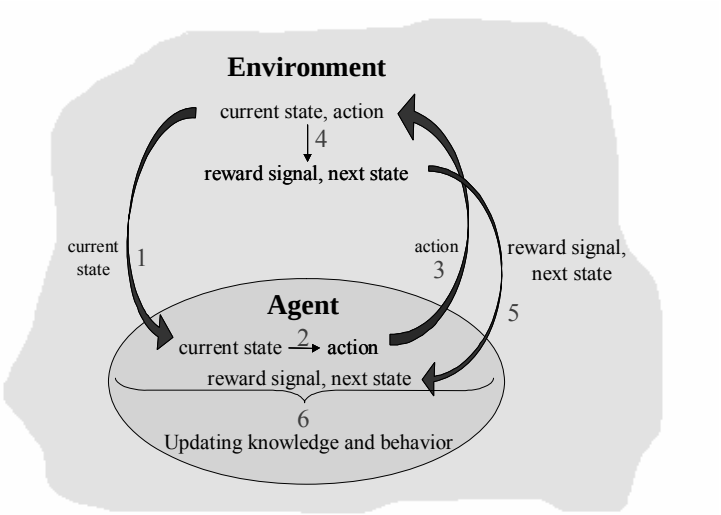

Fig. 1. Agent-Environment interaction

4.2 Q-Learning

Our focus is on RL control using Q-Learning (QL) methods that try to learn the optimal action value function Q as a way to obtain an optimal policy (Watkins, 1999). This function is defined for each state-action pair (s, a) as the total expected discounted reward starting in state s, taking action a and thereafter following the optimal policy.

Here the RL-agent's knowledge is an estimation of the optimal action-value function Q. The classical representation for the estimation of the action-value function is a lookup table. In this case, for each state-action pair (s, a) there is an entry in the table which is the corresponding approximated action value Q(s, a). This estimation is updated during the learning process. The agent starts with some estimation and at each decision moment in which the environment is in state s the agent chooses an action a according to its behavior. The environment reacts to the taken action by giving a reward r to the agent and changing to a new state s' in the next decision moment. With this new information the agent updates the Q-values. The update rule for this method, given the experience tuple $< s, a, r, s' >$, is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

This method converges to the optimal action values with probability 1 as long as all pairs (s, a) are visited infinitely often and the learning rate is reduced over time according to the usual stochastic conditions for the learning (Sutton and Barto, 1998). However, it is surprising how many successful applications usually do not use step-sizes satisfying these conditions.

The behavior defines how the agent chooses the actions. The ideal action in a given state is the one which maximizes the action-value function, the greedy action. However, if we always choose the greedy action based on the actual knowledge, many relevant state-action pairs may never be visited due to the inaccurate estimation of the action-value function. Efficient exploration is fundamental for learning. Too much exploration can cause nearly random behavior and too little can lead to non-optimal solutions. This is known as the exploitation-exploration trade-off.

Exploitation deals with the use of the available knowledge for example by choosing the greedy actions. Exploration increases experience for example by choosing actions at random. Here we use the ε-greedy exploration rule with a parameter ε decreasing over time. With probability 1-ε one chooses a greedy action with respect to the current estimate of the Q-values and with probability ε a random action.

5 Implementation of the Q-Learning method for allocating fragments

Here we describe the elements of the RL-system to solve the allocation problem presented in Section 2. The environment should include all the necessary information needed by the MDP model to define the dynamics of the system: states, actions, immediate rewards, and transitions. It takes into account the restrictions of the problem.

Making an analogy between the allocation problem and an MDP problem, we consider the solution of the allocation problem as the optimal policy for the MDP problem. This solution/policy indicates for each possible pair of fragment and location whether the fragment should be located in the location or not. That is why the states should be all the pairs (fragment, location). The action space has only two actions: allocate the fragment in the location or do not allocate. The immediate reward evaluates the cost of the taken action for the current state. The problem here is that, as explained in Section 2, the allocation cost of a fragment depends on several costs related to the other locations where this fragment is also allocated (remember we are considering replication) and the applications accessing those locations. Thus, all this information is related to previous states and previous actions. This breaks the Markov property; i.e., "the reward function only depends on the current state and the action taken in that state". In our approach, we save the necessary information in the current solution, which is updated every time an action is taken, and in the data from the applications, including this information in the state representation might create a very complicated structure. We choose instead a naive strategy which is to ignore the information for the state, thus the states are like actual observations. This strategy have been used in dealing with Partially Observed MDPs, "small violations of the Markovian properties are well handled by Q-learning algorithms" (Kaelbling et al., 1996).

Summarizing, the environment has the following elements.

State Space: Is the set $\{(f_j, l_k) \mid j = 1...n, k = 1...m\}$. Each state represents a pair (fragment, location).

Actions: At each state (fragment, location) the agent could choose between two actions: allocate the fragment in the location (action=1) or not (action=0).

Immediate reward: Given the current state $s = (f_j, l_k)$ and action a, the current solution is updated making $x_{jk} = a$, and the immediate reward r_{jka} evaluates the cost of allocating the fragment f_j taking into account the updated current solution. According to the problem description in Section 2, this cost is given by:

$$TOC_j = \sum_{\forall a_i \in A} AC_{ij} + IE_{ij} + TC_{ij} + \sum_{\forall s_k \in S} STC_{jk}$$

where:

$$AC_{ij} = \sum_{\forall s_k \in S} \left(u_{ij} \cdot UR_{ij} + r_{ij} \cdot RR_{ij}\right) \cdot x_{jk} \cdot SPC_k$$

$$TC_{ij} = TCU_{ij} + TCR_{ij}$$

$$TCU_{ij} = \sum_{\forall s_k \in S} u_{ij} \cdot x_{jk} \cdot g_{o(i),k} + \sum_{\forall s_k \in S} u_{ij} \cdot x_{jk} \cdot g_{k,o(i)}$$

$$TCR_{ij} = \min_{s_k \in S} (r_{ij} \cdot x_{jk} \cdot g_{o(i),k} + r_{ij} \cdot x_{jk} \cdot \frac{sel_i(f_j)}{fsize} \cdot length(f_j) \cdot g_{k,o(i)})$$

Thus, we consider an episodic task. At each episode there are n times m transitions, going through the whole state space. In our experiments, we use 200 episodes.

The RL-Agent should learn to allocate database fragments in the locations of the network in such a way that the total costs are minimized. It is characterized by the knowledge structure, the policy and the learning method. The learning process is episodic; at each episode the RL-Agent visits all the states and updates its knowledge.

The knowledge structure: The Q action-value function represented by the Q(*s*, *a*) matrix is updated using equation (1) at each transition. The transitions are deterministic; actually the next state only depends on the previous state, see table 1. In a deterministic world this matrix may be initialized arbitrarily and we consider a zero matrix.

Table 1. State transitions

Current state	(f_j, l_k)	
action	0	1
reward	r_{jk0}	r_{jk1}
Next state	(f_j, l_{k+1}), if $k < m$ (f_{j+1}, l_1), if $k = m$	

The policy: The agent follows an ε-greedy policy based on the current action-value function given by the Q matrix.

Next, we summarize the algorithm. Besides initializing the Q matrix as a zero matrix, we need an initial solution satisfying all the constraints for the problem, excluding the storage capacity constraint that was disregarded. This solution is used to evaluate the costs of each action and it is initialized allocating each fragment randomly in only one location.

Algorithm 1. A Q-Learning method for allocating fragments

```
Init Q as a zero matrix
Init SolutionMatrix(random)
btSol←SolutionMatrix
goodSolution←false
Init α←0.5, ε←0.99, γ←0.8
Repeat for all episodes
   Repeat for j←1..n, k←1..m
      S←(f_j, l_k)
      a←SelectAction using an ε-greedy policy
      UpdateSolutionMatrix(a)
      r←Reward(s,a)
      s'←NextState(s)
      Update Q using equation (1)
      s←s'
   UpdateSolutionMatrix(Q)
   If α > 0.15 Then //Update parameters α and ε
      α ← α - 0.015
   If ε > 0.5 Then
      ε ← ε - 0.005
   If SolutionMatrix feasible Then
     If cost(SolutionMatrix)< cost(btSol) Then
       btSol←SolutionMatrix
If btSol feasible Then
  goodSolution←true
```

During the learning process, after an action is chosen, this solution is updated taking the action into account. At the end of each episode the solution matrix is updated considering a greedy policy with respect to the current Q matrix, that is, assigning to each state the action with the smallest value in the Q matrix. Thus, any fragment can be allocated to more tan one location since it is possible to find for one fragment in the state space the smallest value in the Q matrix corresponding to the same action=1. If a fragment is not allocated in any location (unfeasible solution) we force the fragment to be in the location with the smallest difference in the action-values for that fragment. In the algorithm we keep record of the best feasible found solution (btSol), which is initially the same initial solution, not necessarily feasible (goodSolution←false). A parameter tuning was completed as a result of the tests applied to the

algorithm 1. Parameters α, ε and γ were fixed with values 0.5, 0.99 and 0.8 respectively. For the implementation we used Visual C# from Visual Studio 2005 by Microsoft®.

6 Experimental cases

In order to test the intelligent RL-Agent from Section 5 in the allocation problem, we generate 15 random test cases given the rank of values for the amount of fragments (FRG.), locations (LOC.) and applications (APP.). In the first three cases we obtained the optimal solution using an exact method. In the rest of the cases (†), we report the best solution found by using heuristics like Simulated Annealing and Genetic Algorithms (Rosa, 2006).

Table 2 shows these cases and the lesser time elapsed to find the solution for the total cost function (TOC) as in section 5, namely the best solution costs (BSC) measured in milliseconds.

Table 2. Experimental cases

CASE	FRG.	LOC.	APP.	BSC
1	6	2	22	61104.85
2	7	4	5	13549.53
3	3	10	6	4479.58
4	17	3	8	52659.82†
5	24	2	10	102502.55†
6	16	3	22	54498.22†
7	8	6	16	71426.56†
8	17	11	5	86087.00†
9	11	8	7	40975.05†
10	25	7	5	51282.82†
11	14	9	29	252440.20†
12	24	16	16	198615.14†
13	27	18	18	369647.17†
14	30	20	20	339398.45†
15	36	24	24	549275.72†

7 Experimental results

Here we present the experimental results of our approach in order to evaluate the quality of the solution and the processing time. Times are also considered in milliseconds. Table 3 shows these results.

Each case has been solved 20 times and the average cost was found (AVE). The table also shows the times the best known solution was found (TB), the cost of the worst found solution (WORST) and the absolute value of the relative error (RE) calculated as follows:

$$\mathrm{RE} = \frac{|\mathrm{AVE} - \mathrm{BEST}|}{\mathrm{BEST}} \cdot 100$$

The best known solution obtained by the Q-Learning is guaranteed to converge to the optimal due to the use of a lookup table to store the Q-values. Every state-action pair continues to be visited, and the learning rate is decreased appropriately over time. It has been assumed our function approximator is a lookup table. This is normally the case in classical dynamic programming. However, this assumption limits the size and complexity of the problems solvable.

Table 3. Experimental results

CASE	BEST	AVE	TB	WORST	RE
1	61104.85	61106.27	19	61133.38	0.0023
2	13549.53	13559.16	17	13613.77	0.0711
3	4479.58	4496.48	19	4817.70	0.3773
4	52659.82†	52659.82	20	52659.82	0.0000
5	102502.55†	102571.99	5	102913.56	0.0677
6	54498.22†	54651.85	15	55285.40	0.2819
7	71426.56†	71962.17	3	72621.84	0.7499
8	26087.00†	26672.93	2	27983.13	2.2461
9	40975.05†	41922.69	4	43213.53	2.3127
10	51282.82†	52080.81	2	52861.03	1.5561
11	252440.20†	259728.43	2	266100.01	2.8871
12	198615.14†	210416.34	1	223640.47	5.9417
13	369647.17†	400167.30	1	426956.78	8.2566
14	339398.45†	379218.21	1	403171.71	11.7325
15	549275.72†	629684.21	1	688023.40	14.6390

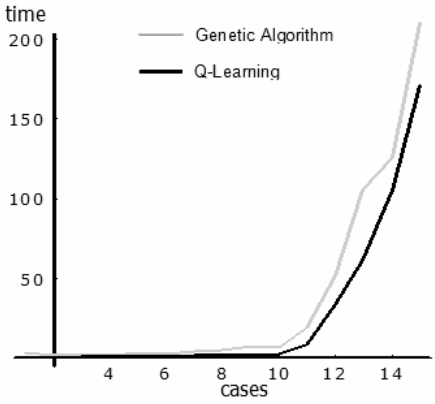

Fig. 2. Execution time for comparing methods

Figure 2 depicts the time elapsed to find the optimal solution for every case by using a Genetic Algorithm in (Rosa, 2006) and the Q-Learning method proposed here. Both methods have been developed to find a near-optimal allocation such that the total cost (TOC) is minimized as much as possible. These methods use the same quantitative data about the database, the applications behavior, the locations and network information. Both methods also use the same cost function. As a result, we find that the Q-Learning method performed better than Genetic Algorithm in (Rosa, 2006) for smaller cases. The relative error values for both methods are fairly similar in most cases.

8 Conclusions

We consider the allocation problem in a distributed database and propose an intelligent RL-Agent to solve it by using the Q-Learning algorithm. The improvements are sought by maintaining copies of the fragments. The decision regarding replication is a trade-off which depends on the ratio of the read-only applications to the update applications. The problem is then modeled using the Reinforcement Learning approach and the algorithm is adapted to the problem. The parameters are chosen through experimental experience and are by no means optimized. This method

neglects integrity enforcement and concurrency control costs as well as the storage capacity constraint, as an attempt to reduce the complexity of the problem.

The results show that this approach could obtain practical solutions, even for larger cases. We believe the RL approach is a very flexible method that can be applied to obtain good solutions in complex problems.

9 Recommendations and future work

Further research could study the tuning of the parameters involved in the algorithm. Specifically we would like to recommend the tuning of the number of episodes according to the size of the problem, i.e., fragments times locations.

Many real-world problems have extremely large or even continuous state spaces. In practice it is not possible to represent the value function for such problems using a lookup table. Hence, we recommend using a function approximator that can generalize and interpolate values of states never before seen as an extension to classical value iteration. For example, one might use a neural network for the approximation.

At this moment we work on the integration of several algorithms for the allocation problem, specifically Q-Learning and SARSA from RL, Genetic Algorithms, Bird Flocks and some other tools developed by our research group. The main goal is to help in the design of Distributed Databases in a more efficient way by using less effort and time.

10 Acknowledgements

This work is part of a research and development Project “Database Technologies applied to problems”, Number 01700031 from the National Program of Information Technology supported by the Cuban Ministry of Science, Technology and Environment. The authors would like to thanks to the Flemish Interuniversity Council (Vlaamse InterUniversitaire Raad) for their support through the IUC VLIR-UCLV Program.

References

1. **Abe, N.; Biermann, A. W. and Long, P. M.** (2003). "Reinforcement Learning with Immediate Rewards and Linear Hypotheses." Algorithmica 37(4): 263-293.
2. **Ceri, S. and Pelagatti, G.** (1984). Distributed Databases: Principles and Systems, McGraw-Hill Book Company.
3. **Choi, S. P. M.; Zhang, N. L. and Yeung, D.-Y.** (2004). Reinforcement Learning in Episodic Non-stationary Markovian Environments. Proceedings of the International Conference on Artificial Intelligence, IC-AI '04. Proceedings of the International Conference on Machine Learning; Models, Technologies and Applications, MLMTA '04, Las Vegas, Nevada, USA, CSREA Press.
4. **Hababeh, I. O.; Bowring, N. and Ramachandran, M.** (2004). A Method for Fragment Allocation in Distributed Object Oriented Database Systems. Proceedings of the 5th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting (PGNet), Liverpool, UK.
5. **Huang, Y.-F. and Chen, J.-H.** (2001). "Fragment Allocation in Distributed Database Design." Journal of Information Science and Engineering 17(3): 491-506.
6. **Iida, S.; Kuwayama, K.; Kanoh, M.; Kato, S. and Itoh, H.** (2004). A Dynamic Allocation Method of Basis Functions in Reinforcement Learning. Proceedings of the 17th Australian Joint Conference on Artificial Intelligence AI 2004: Advances in Artificial Intelligence, Cairns, Australia, Springer.
7. **Kaelbling, L.; Littman, M. and Moore, A.** (1996). "Reinforcement learning: A survey." Journal of Artificial Intelligence Research 4: 237-285.
8. **Lin, X. and Orlowska, M. E.** (1995). "An Integer Linear Programming Approach to Data Allocation with the Minimum Total Communication Cost in Distributed Database Systems." Information Sciences 85(1-3): 1-10.
9. **Lin, X.; Orlowska, M. E. and Zhang, Y.** (1993). On Data Allocation with the Minimum Overall Communication Costs in Distributed Database Design. Proceedings of the Fifth International Conference on Computing and Information - ICCI'93, Sudbury, Ontario, Canada, IEEE Computer Society.

10. **Ma, H.; Schewe, K.-D. and Wang, Q.** (2006). A Heuristic Approach to Cost-Efficient Fragmentation and Allocation of Complex Value Databases. The 17th Australasian Database Conference (ADC2006) Hobart, Australia, ACM International Conference Archive Proceeding Series vol. 170. Australian Computer Society, Inc.
11. **March, S. T. and Rho, S.** (1995). "Allocating Data and Operations to Nodes in Distributed Database Design." IEEE Trans. Knowl. Data Eng. 7(2): 305-317.
12. **Morales, E. F. and Sammut, C.** (2004). Learning to fly by combining reinforcement learning with behavioural cloning, ACM.
13. **Özsu, M. T. and Valduriez, P.** (1999). Principles of Distributed Database Systems, Second Edition, Prentice-Hall.
14. **Papadimitriou, C. H.** (1997). NP-Completeness: A Retrospective. Proceedings of the 24th International Colloquium on Automata, Languages and Programming, ICALP'97, Bologna, Italy, Springer.
15. **Pérez, J.; Pazos, R. A.; Frausto-Solis, J.; Reyes, G.; Santaolaya, R.; Fraire, H. J. and Cruz, L.** (2005). An approach for solving very large scale instances of the design distribution problem for distributed database systems. Proceedings of the 4th International School and Symposium on Advanced Distributed Systems (ISSADS2005). Lecture Notes in Computer Science, Springer.
16. **Pérez, J.; Pazos, R. A.; Mora, G.; Castilla, G.; Martínez, J. A.; Landero, V.; Fraire, H. J. and González, J. J.** (2004). Dynamic Allocation of Data-Objects in the Web, Using Self-tuning Genetic Algorithms. Proceedings of the 17th Brazilian Symposium on Artificial Intelligence- SBIA 2004, Advances in Artificial Intelligence São Luis, Maranhão, Brazil, Springer.
17. **Pérez, J.; Pazos, R. A.; Romero, D.; Santaolaya-Salgado, R.; Rodríguez, G. and Sosa-Sosa, V. J.** (2003). Adaptive and Scalable Allocation of Data-Objects in the Web. Proceedings of the International Conference on Computational Science and Its Applications - ICCSA 2003, Part I., Montreal, Canada, Springer.
18. **Pérez, J.; Pazos, R. A.; Santaolaya-Salgado, R.; Frausto-Solís, J.; Rodríguez, G.; Cruz, L. and Bravo, M.** (2003). Data-Object Replication, Distribution, and Mobility in Network Environments. Revised Papers of the 5th International Andrei Ershov Memorial Conference, Perspectives of Systems Informatics, PSI 2003, Akademgorodok, Novosibirsk, Russia, Springer.
19. **Pérez, J.; Pazos, R. A.; Velez, L. and Rodríguez, G.** (2002). Automatic Generation of Control Parameters for the Threshold Accepting Algorithm. MICAI 2002: Advances in Artificial Intelligence, Second Mexican International Conference on Artificial Intelligence, Merida, Yucatan, Mexico, Springer.
20. **Puterman, M. L.** (1994). Markov Decision Processes: Discrete Stochastic Dynamic Programming. New York, NY John Wiley & Sons.
21. **Rosa, D.** (2006). Aplicación de técnicas de Inteligencia Artificial en la solución del problema de ubicación en el diseño de bases de datos distribuidas. Departamento de Ciencia de la Computación. Santa Clara, Universidad Central "Marta Abreu" de Las Villas. BSc.Thesis.
22. **Sutton, R. and Barto, A.** (1998). Reinforcement Learning: An Introduction. Cambridge, Massachussets, USA., MIT Press.
23. **Watkins, C.** (1999). Learning from Delayed Rewards. Cambridge, UK, King's College.
24. **Wolfson, O. and Jajodia, S.** (1995). "An Algorithm for Dynamic Data Allocation in Distributed Systems." Inf. Process. Lett. 53(2): 113-119.

Abel Rodríguez Morffi *was born on January 25, 1969 in Santa Clara, Cuba. In 1993 he obtained his B.Sc. in Computer Science at the Central University of Las Villas, Santa Clara, Cuba, and in 1996 obtained his M.Sc. in Information Management at the University of Havana, Havana City, Cuba. In 2007 he obtained his Ph.D. at the Central University of Las Villas on a research project "Integrated tools to design distributed databases". He is a professor in the Department of Computer Science, Central University of Las Villas.*

Darien Rosa Paz *was born on September 18, 1982 in Villa Clara, Cuba. In 2006 he obtained his B.Sc. in Computer Science at the Central University of Las Villas, Santa Clara, Cuba. His current research interests are Operations Research and Distributed Databases Design. He is assistant professor at the Central University of Las Villas.*

Marisela Mainegra Hing *was born on April 15, 1972 in Santiago de Cuba, Cuba. In 1995 she obtained her B.Sc. in Computer Science and in 1997 her M.Sc. in Applied Mathematics at the Central University of Las Villas, Santa Clara, Cuba. In 1998 she graduated from the master class program in Operations Research organized by the Mathematical Research Institute in The Netherlands. In 2006 she obtained her Ph.D. at the University of Twente on a research project "Intelligent Computational Techniques supporting Learning Organizations". She is assistant professor at Central University of Las Villas.*

Luisa Manuela González González *was born on April 25, 1954 in Sancti Spiritus, Cuba. In 1973 she obtained her B.Sc. in Mathematics at the Central University of Las Villas, Santa Clara, Cuba, and in 1995 she obtained her Ph.D. on a research project "Integrated tool for engineering design". Since 1975, she has been a professor in the Department of Computer Science, Central University of Las Villas. For almost twenty years, she has been the manager of the Database Research Group. Her current research interests are data modeling, decision support systems, and ontologies.*