Information Replication Strategy in Unstructured Peer-to-Peer Networks Using Thematic Agents

Nicolas Bonnel, Gildas Ménier, Pierre-François Marteau

Information Replication Strategy in Unstructured Peer-to-Peer Networks Using Thematic Agents

Nicolas Bonnel, Gildas Ménier and Pierre-Francois Marteau
VALORIA, University of Bretagne Sud, France
{nicolas.bonnel, gildas.menier, pierre-francois.marteau}@univ-ubs.fr

Abstract

We present in this article a method to wisely replicate information in an unstructured peer-to-peer network. We make no assumption on the network topology. Thematic agents move randomly on the network and estimate the level of redundancy of the specific information they are dealing with. They can delete or create replicas if this estimated redundancy is too high or too low. Experiments show that we can achieve an homogeneous distribution of information in a distributed environment while achieving a high level of fault tolerance.

1. Introduction

The growing need for very large databases management systems raises indexing and querying difficulties. Building an index allows to speed up the querying process [13] and in general this index is larger than the database itself. Distributing the database index allows to store all these information and to manage more volume of information. Among other solutions, Peer-to-Peer (P2P) networks are decentralized systems that can contain up to millions of nodes. In these systems, the nodes continuously enter or leave the networks, imposing a constant moving environment for the hosted database that needs to adapt to these varying conditions.

Living organisms have adapted to their environment since millions of years. Recently, some algorithms have been proposed to reproduce some mechanisms of living and evolving organisms, this is the field of Artificial Life (AL). These algorithms, such as genetic algorithms (GA), are generaly used in optimization problems. They can provide sub optimal solutions for NP-hard problems in a short time. They are suited to changing environments too. For instance, the population simulated in GA can evolve and adapt to a changing environment.

Ant Colony Optimization (ACO) algorithms are inspired by the behaviour of real ants. For instance when an ant finds a dead body alone, it picks it up and carry it until it finds other dead corpses. Then it drops it, increasing the number of dead bodies in its neighborhood. This local behaviour of the ant leads to a global clustering process, that is used to sort ant larvaes too. Ant clustering algorithms have been proposed to cluster similar data [5, 11, 18].

In a Peer-to-Peer network, the replication of information makes the system more robust and allows to improve the query response rate of random-walk-like routing algorithm [12]. We adress in this article the problem of the replication of data in an unstructured peer-to-peer network using an ACO-like algorithm. Agents moves randomly on the network carrying indexed information. They can create replicas of the information they carry on nodes hosting information with similar summary. On the other hand replicas of this information are deleted if its summary does not correspond to the one of other information hosted.

The objective is to maintain an homogoneous distribution of information replicas, so that a fixed amount of information remains reachable even if a significant amount of node is offline. The number of replicas is bounded, the lower and upper bounds vary with the size of the network. Section 2 coments on previous work, section 3 presents our approach, section 4 shows experiments and section 5 highlights some perspectives.

2. Related Work

A Multi Agent Sytem (MAS) [7, 19] is composed with autonomous entities called agents that interact with each others. Artificial ants are reactive agents inspired by the real ants behaviour. They communicate using pheromones, an indirect mean of communication. ACO algorithms are fully distributed and can solves various problems such as clustering, routing, assignment or scheduling [6].

2.1. Ant Clustering

The ant clustering problem has been studied in the last few years. The problem is to aggregate similar objects, ini-

tially disposed at random on a 2 dimensionnal grid. Basicaly, ants move randomly on the grid, pick up objects surrounded by heterogeneous ones, and drop them near similar ones. This problem has been introduced in [5]. The agents have a finite memory of size n that records the absence or presence of objects at the ant's n last locations. Moreover, they have a capability to manipulate objects, the manipulation (taking or dropping) of these objects being dependant on the ant's knowledge. In [5]'s model, manipulated objects could be either identicals or differents. [11] extends this model to cope with a continuous similarity measure. [18] proposed an algorithm called ACLUSTER used in an image database. ACLUSTER avoids random moves of ants using pheromones for communication. This algorithm can perform a continuous clustering of a data stream, contrary to recent similar approaches using Self-Organizing Maps (SOM) [8] that need to perform a new training if new categories of data appear.

A clustering algorithm based on the chemical recognition mechanism of ants, called ANTCLUST, has been proposed in [10]. Each data is associated with an ant and defines its odor. Ant meeting is then simulated and ants with similar odors are grouped together.

2.2. Peer-to-peer (P2P) networks

Peer-to-peer (P2P) systems can have different architectures. Napster [15] for instance is a centralized P2P network that was very popular in the early 00'. The use of a central repository to answer queries makes this system poorly scalable and vulnerable to failure. Others P2P systems don't rely on a central server : they are decentralized, thus they are very scalable, and fault tolerant. They are divided in two categories.

Structured P2P networks associate network topology and location of data. Most of them implement a Distributed Hashtable (DHT) [14, 16, 17, 20, 2] and provide one basic operation : given a key, they map the key to a node. This is performed by using a distributed hash function. They use content routing to forward the key to the corresponding node. They are very well suited to retrieve rare information (i.e. with a low number of replicas). Their main limitation is that it is very costly to perform ranged and approximative queries, because hashing destroys the order on keys.

Unstructured P2P networks have no constraint between location of data and network topology. Gnutella [3] is an example of such working system. Query forwarding can be achieved either by flooding [12] - consuming a lot of bandwidth - or by random walk : a path is selected randomly according to a uniform distribution. They are suited to retrieve highly replicated data, but have limitations for rare information retrieval. Nowaday, most P2P systems have a decentralized unstructured topology.

3. Contribution

We address the design of a distributed database having self healing capacities through wise replication. To achieve this, we perform information replication using the multi agent system paradigm.

3.1. Architecture

Our current work focuses on indexing documents and distributing the database index. As we need to be able to perform approximative queries, we choose an unstructured P2P architecture. The primary purpose of the nodes on the network is not necessarily to manage this index database. The nodes may have to set aside the management of the database for a while if a user would need the resources on this node for his own purpose. For this reason our P2P network does not have super peers. The use of an unstructured P2P architecture allows us to have a network topology as close as possible to the physical topology and this can lead to use less network resources.

We consider an unstructured Peer-to-Peer network of nodes with similar computing capacities. Thus, there is no super-peers, and the network have a random graph like topology. Each node on the network hosts a part of the index database. Its knowledge is stored in a hashtable where each keywords is an entry pointing toward indexed information. This can be for instance the documents and the positions of the keyword in these documents. As they do not index the same documents, different nodes can have the same keyword with different indexed information.

Moreover, nodes have a summary of the keywords they host. This summary is implemented as a Bloom filter [1], as in Gnutella. A Bloom filter is a data structure that answers approximatively to set membership queries. It is made of an array of m bits and k hash functions $h_1, ..., h_k$. When no element has been inserted into the filter, all bits are set to 0. When an element x is inserted into the filter, all bits given by the k hash functions $h_i(x)$ are set to 1 with a classical bitwise-or operation. A membership test of an element to a set is answered by checking that all the bits given by the k hash functions are set to 1. False positives are possible but false negatives are not. The probability of having a positive answer for an element not belonging to the filter, with n being the number of elements inserted in the filter, is $(1 - (1 - \frac{1}{m})^{nk})^k$.

This summary allows to speed up query forwarding, because it is faster to query the filter than querying the whole local database. Off course, a false positive answer may trigger an unnecessary local full query. As described in [12], replicating information greatly improves random walk efficiency, and increases the system robustness. We focus in this article in a strategy of replication on nodes with simi-

lar summary. We try to keep false positive answer rate in Bloom filters as low as possible.

Those filters can be used to improve random-walk efficiency. For instance, Exponentially Decaying Bloom Filters (EDBF) improve query routing as described in [9]. Filters act as routing indices [4], and drive queries toward regions of the network hosting the searched information. In those filters, introducing an element is performed as in a classical Bloom filter. Querying for an element x gives the number $\theta(x)$ of bits equal to 1. Thoses filters are then used to encode probabilistic routing tables, in which $\frac{\theta(x)}{k}$ is the probability to find element x among a given link in the network (Each node maintains a filter for each neighbor he has). This probability decays exponentially with the number of *hops* (or node transitions) from the node where the element x is stored. Nodes update their filters periodically. The filter for one neighbor is updated with the information attenuated from all other neighbors and the information without attenuation from the local EDBF of the node. The attenuation is performed by reseting each bit to 0 with a probability $1/d$, where d is the decay of the filter. The aggregation of information corresponds to a bitwise-or operation. To forward a query, the Scalable Query Routing algorithm (SQR) [9] is used. This algorithm act as random-walk until it finds enought information and converge to the information searched.

We believe keeping filters occupation as low as possible could be beneficial to improve such query forwarding algorithms. Thus, an homogoneous distribution of information might limit each local use of Bloom filter.

3.2. Agent behaviour

We argued previously that replicas have to be created on nodes having similar summary of the information they host. We use agents to control the number of replicas for each data in the network. According to its local knowledge, an agent can create or delete replicas for the information it is carrying. Agents have a theme, which is implemented in our system as a keyword. They have a very low probability p to be reinitialized. This process change their theme and allows to statistically eqally process all data.

The behaviour of the agents is depicted in algorithm 1. Each agent carries a keyword k and the indexed information related to this keyword. When the agent is (re)initialized on a node N_l, it records the oldest accessed or updated keyword and local index information that is associated. It stores the node where it has been initialized too. The agent records the n last nodes visited and the nodes hosting its keyword k. It then moves randomly on the network. If the local node has different knowledge related to the keyword k the agent is carrying, the agent and the visited node exchange their knowledge. Moreover, the agent forgets information about last visited nodes and considers it has been reinitialized in this current node (i.e. this node is recorded as N_l).

Algorithm 1: Agent behaviour

```
// agent initialization
Node source ← currentNode();
String k ← oldestKeyword(source);
Set info ← indexedInfo(source,k);
int score ← S(k, source);
Node current ← source;
Queue visited ← ∅;
// main loop
while true do
    if visited.size() ≥ n then
        visited.poll();
    end
    visited.enQueue(current);
    current ← randomNeighbor(current);
    if current.hasKeyword(k) then
        if indexedInfo(current,k) ≠ info then
            info ← indexedInfo(current,k) ∪ info;
            setIndexedInfo(current,k,info);
            source ← current;
            score ← S(k, current);
            visited ← ∅;
        else if φ(k, current) ≥ τ_sup then
            removeIndexedInfo(current,k);
        end
    else if φ(k, current) ≤ τ_inf then
        setIndexedInfo(current,k,info);
    end
    // new theme for the agent
    if random() ≤ p then
        reinitialize this agent;
    end
end
```

Each time it visits a node N_c, the agent computes a score $\phi(k, N_c)$, given by the following equation :

$$\phi(k, N_c) = \frac{S(k, N_l)}{S(k, N_c)} \times \frac{f(k)}{\alpha} \quad (1)$$

$S(k, N)$ is a scoring function for a node N for the keyword k. This function measures a trade off between the space available on the node and the degree of matching of the keyword to the node filters. This degree of matching is given by the function described previously that gives the number $\theta(k)$ of bits equal to 1 for the keyword k. α is a constant that tunes the replication amount to achieve. $f(k)$ is the frequence of last nodes visited hosting the information carried by the agent.

The former part of the scoring function deals with the constraint of having similar information summaries on the same node while the other part leads to have replicas of information as far as possible from each other.

We define the replicating bound τ_{inf} and the deleting bound τ_{sup}, with $\frac{\tau_{inf}+\tau_{sup}}{2} = 1$. If $\phi(k, N_c) \leq \tau_{inf}$, a replica of the index information carried by the agent is created on the local node N_c. If $\phi(k, N_c) \geq \tau_{sup}$, all indexed information for the keyword k is removed from the local node N_c, if it exists. When the system is stable (i.e. the number of replicas created or deleted per hour is very low), a network with m nodes should have $\frac{m}{100} \times \alpha$ average number of information replicas.

4. Experiments

We have simulated an unstructured peer-to-peer network with a random graph like topology. In this first experiment, the network topology is stable : there is no connection or node failures. Experiment settings are described in table 1.

Table 1. Experiment settings

Parameter	Value
Number of nodes	400
Node degree	2 - 8
Documents per node	75
Size of filters	2^{13} bits
Number of hash functions	32
Replicating constant	2
Bounds τ_{inf} and τ_{sup}	0.8 and 1.2
Nodes recorded	100
Number of keywords	10985
Number of ants	2000

4.1. Information replication

The number of replicas follows a normal distribution, centered on 13, with a maximum of 24 replica for 2 information and a minimum of 1 replica for 1 information. The average number of replica is superior to what we expected, as having 10 replicas leads to an average redundancy of 2.5. With $\alpha = 2$, having more replicas than 10 should produce a destruction of some replicas. This could be explained by the random move of agents that could miss some replicas, therefore having only an estimation of the redundancy.

Figure 1 shows the evolution of the replication process. Initialy, the information is not replicated and the occupation of filters is 43.47%, corresponding to a false positive answer probability of 2×10^{-52}. Then the number of replicas increase to an average of 13 replicas, with a increase in filter occupation to 70.88%, corresponding to a false positive answer probability of 1.6×10^{-5}. This is still small comparing to the increase in term of query answer efficiency. There is no difference in filters occupation between 9 and 13 replicas, meaning 4 replicas (in average) are created without adding information in filters.

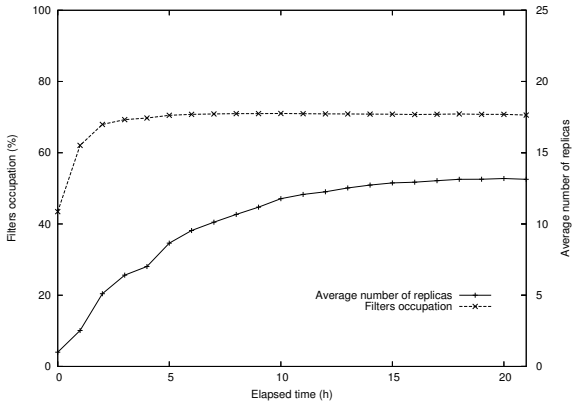

Figure 1. Evolution of the number of replicas and of the filters occupation.

Figure 2 shows that random walk performs way better in a highly replicated environment. Half answers to queries are obtened within 50 hops when there is 13 replicas (in average) when it takes about 1000 hops to have the same results with no replication. In figure 3, we measure the number of hops required to answer from 5% to 50% of queries and make the ratio between result in unreplicated and replicated environment.

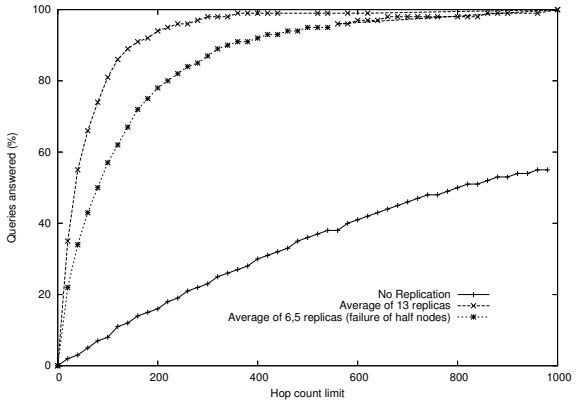

Figure 2. Comparison of random walk in unreplicated and replicated environment.

Figure 3 shows that it is 22 times faster (in average) to answer between 5 % and 50 % of queries with an average of 13 replicas per information comparatively to the case with no replication. This shows that the distribution of replicas

is homogoneous, as wherever the query is forwarded at random, it still finds a replica of the searched information.

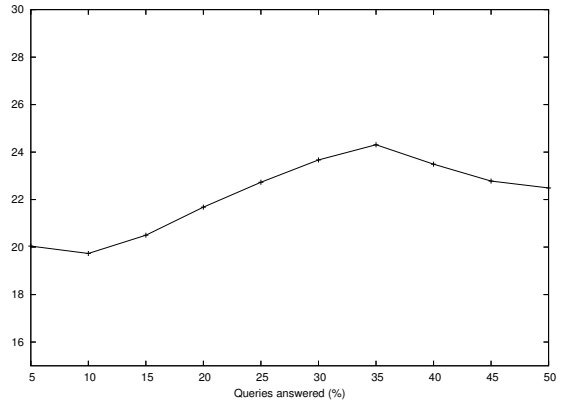

Figure 3. Ratio between random walk in unreplicated and replicated environment (average of 13 replicas per data).

4.2. Self healing capacities

In a second experiment, we have simulated a failure for half of the nodes in the network. Basically, we have reseted the memory of half of the nodes of the network chosen at random. This cuts the average number of replicas down to 6.5. We definitely lost $0,036\%$ of indexed information. The query answer efficiency just after this failure is given in figure 2. The number of replicas then grows like in figure 1 and reaches back asymptotically 13.

Nodes stores information with similar summary. However, because the agents move randomly on the network, information with similar summary will not necessarily be stored on the same nodes. For instance, if information A, B and C have similar summary, we can have a node having replicas of A and B, another having A and C and a third one having B and C. As very few information has been lost, we believe our system leads to this kind of strategy for replicas storage. However we still need to perform further experiments to prove such behaviour.

5. Conclusion

We have introduced an agent designed to perform information replication in an unstructured peer-to-peer network. This can achieve an homogeneous distribution of information replicas and features a kind-of self healing property. Given a hard failure of the network (half nodes fail), the lost of information is very low. Because of its fully decentralized nature, our algorithm may scale very well, but we need to test it in larger P2P networks simulations.

Further experiments are required to model a more changing environment, for instance with less nodes failures but with higher fault frequencies. It would also be interesting to see if our model can increase query routing efficiency of algorithm such as SQR.

6 Acknowledgements

This research was supported by Region Bretagne.

7. References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009, 2001.

[3] Clip2. The gnutella protocol specification v0.4, 2002.

[4] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems, 2002.

[5] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting robot-like ants and ant-like robots. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 356–363, Cambridge, MA, USA, 1990. MIT Press.

[6] M. Dorigo and T. Sttzle. *Ant Colony Optimization (Bradford Books)*. The MIT Press, July 2004.

[7] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[8] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[9] A. Kumar, J. Xu, and E. W. Zegura. Efficient and scalable query routing for unstructured peer-to-peer networks. In *Proc. of IEEE Infocom*, 2005.

[10] N. Labroche, N. Monmarché, and G. Venturini. AntClust: Ant Clustering and Web Usage Mining. In E. Cantu-Paz, editor, *Genetic and Evolutionary Computation Conference*, volume 2723 of *Lecture Notes in Computer Science*, pages 25–36, Chicago, july 12-16 2003. Springer-Verlag Telos.

[11] E. D. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, pages 501–508, Cambridge, MA, USA, 1994. MIT Press.

[12] C. LV, P. CAO, E. COHEN, K. LI, and S. SHENKER. Search and replication in unstructured peer-to-peer networks, 2001.

[13] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajamaran. Indexing semistructured data, 1998.

[14] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM 2001*, San Diego, CA, September 2001.

[15] Napster. Napster homepage, 2001. http://www.napster.com/.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[17] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[18] J. J. M. Vitorino Ramos. Self-organized stigmergic document maps: Environment as mechanism for context learning.

[19] G. Weiß, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[20] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.