Repairing Flocks in Peer-to-Peer Networks

Benoît Romito, Hugo Pommier, François Bourdon

Repairing Flocks in Peer-to-Peer Networks

Benoît Romito, Hugo Pommier, François Bourdon
GREYC UMR 6072,
CNRS, University of Caen, France
Email: {benoit.romito,hugo.pommier,francois.bourdon}@unicaen.fr

***Abstract*—In this paper we consider the problem of reliable and robust data storage in peer-to-peer networks. The approach we follow builds a multi-agents system in which documents are split using a (m, n)-erasure code. Each generated fragment is embedded into an autonomous and mobile agent. This mobility gives them, for example, the ability to choose the network area where they want to be hosted. But this motion may have heavy consequences on the system robustness if the initial fragmentation parameters of a document are inappropriate. As a consequence, we focus on the problem of finding the suitable m and n values to apply to a document given the underlying peer-to-peer network properties.**

***Keywords*-mobile agents, peer-to-peer networks, bio-inspired model, repair policy, reliable storage**

I. Introduction

Decentralized peer-to-peer networks are dynamic and heterogeneous systems where participants have the same responsibilities and no particular peer occupies a central function. The dynamism, caused by a connection/disconnection phenomenon called *churn*, generates some availability and dependability issues in data storage applications. Indeed, no availability guarantees can be expected from the peer that stores a data. In this kind of architecture, robustness and availability is obtained by setting up:

1) *A redundancy policy:* where documents are replicated or fragmented using erasure coding [1] to provide fault-tolerance.
2) *A monitoring policy:* it is necessary to monitor the number of fragments of a document over time in order to repair it when fragments are lost. Among existing monitoring policies [2], the local policy places the fragments of a document on neighborhoods in the overlay[1]. This policy is particularly interesting because it provides a fully decentralized monitoring.
3) *A repair policy:* it defines the time when a regeneration of fragments has to be initiated [3].

In usual decentralized approaches, like Past [4], the peers are responsible for the application of these policies and they are blindly trusted. In [5] we have proposed a mobile-agents oriented approach where this responsibility is transfered from the peers to the documents themselves. In this information model, each fragment is embedded into a mobile agent and the document is then represented by its group of agents. The network can be seen as a simple execution environment for the agents that can cooperate afterwards to apply their own repair policies. These agents have also the capacity to move inside the network. In this way, the group can apply its own policies locally and has the ability to decide the network area where it wants to be stored. To keep a high degree of locality between the agents and a decentralized decision making, flocking rules are asynchronously applied by each agent to decide its moves. The global emerging behavior is a flock of fragments. The early results in [5] indicate that the flexibility of the flocking motion as well as the network parameters have an impact on the repair policy efficiency and on the initial fragmentation scheme. As a consequence, this paper is focused on the repair policy to add on the flocking model and its implications on the fragmentation parameters to apply to a document according to the underlying network properties.

[1] i.e., the logical peer-to-peer network

II. Related Works

Setting up a redundancy policy is the first step required to achieve data availability in peer-to-peer data storage applications. It can be obtained by generating a (m, n)-erasure code that splits a data into m blocks of equal sizes. After an encoding step, n fragments of redundancy are generated. In this way, it is possible to rebuild the original data by decoding any subset of m fragments among the $m + n$. This code is able to tolerate n fragments losses. An erasure code provides the same level of availability as replication using less storage cost [6]. Among replication-based peer-to-peer data storage applications we can cite GFS [7] and Past [4]. For erasure-coding-based applications we can cite Glacier [8] and Flocking [5].

The second mechanism required to achieve availability is a monitoring policy. Some studies have been made on this particular question [2] and two distinct policies are emerging: *the global policy* and *the local policy*. The first one consists in distributing uniformly at random the fragments of a document in the whole network. In this policy a monitoring peer is randomly chosen among the set of peers to monitor the fragments of a document. This data placement provides an efficient load balancing and limits the bottleneck effect on one peer. GFS [7] uses this data placement method but relies on a ”master” entity which knows the exact state

of the network. This central entity is an evident reliability drawback. In *the local policy*, the fragments of a document are placed on the neighbors of a peer. In this way, they can monitor each other and the monitoring is not centralized on a peer anymore. It is the case in Glacier [8], Past [4] and Flocking [5].

In the last policy, the results of the monitoring process are analyzed to repair lost fragments when it becomes a necessity. In the *eager repair* a reconstruction is triggered every time a fault is detected. This easy-to-deploy policy does not make any difference between permanent and transient failures and thus, generates an unnecessary network bandwidth consumption and a storage overhead. This policy is applied in Past [4] and Glacier [8]. To overcome these problems a more subtle policy called *lazy repair* has been proposed [9] where a tolerance threshold r is defined such that $m \leq r < m + n$. A repair is triggered only when the fragments number reaches r. Thus, the lazy repair tolerates a certain amount of temporary disconnections. Among applications relying on the lazy repair policy, we can cite GFS [7] and Total Recall [9].

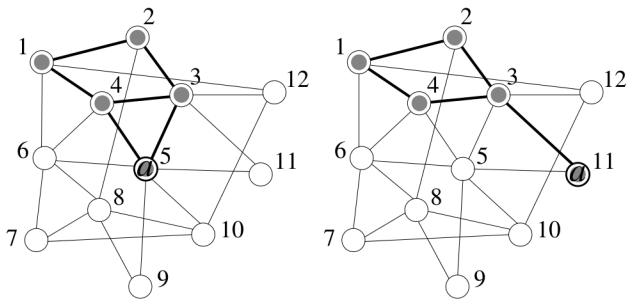
III. The Flocking Model

A. Mobile Agents and flocking rules

In the following of this article, we only consider the Flocking model detailed in [5]. This model builds a multi-agents system in which each information fragment, obtained by erasure coding, is embedded into a cognitive mobile agent able to make its own decisions. This cognition allows the agents to autonomously move from peers to peers following flocking rules similar to those proposed by Craig Reynolds [10]. Each document is then represented by the flock composed of the set of its fragment-agents. A distance measure between two agents has been introduced and is given by the round trip time (RTT) between their hosting peers. Reynold's rules are transposed in a network and applied asynchronously by each agent such that:

1) A given peer can only store one fragment per document (separation rule).
2) The agents move in order to get closer to the most distant agents (cohesion rule associated to a RTT distance).

Figure 1 displays a network of 12 peers hosting a 5 fragments flock represented by the gray discs. The flocking algorithm described in [5] works as follows. The first step for a moving agent, let say agent a hosted on peer 5, is to build $busy$, the set of neighboring peers hosting fragments from the same file and $free$ the others. We have in the example $busy_a = \{3, 4\}$ and $free_a = \{6, 9, 10, 11\}$. After that, the agent builds $anchors$, the subset of $busy$ containing the most distant peers. We consider in the example that 3 is far enough from 5 according to the RTT so $anchors_a = \{3\}$. A set of candidates, stored in $candidates$, is then build by selecting elements of $free$ that are neighbors of $anchors$. In the example, $candidates_a = \{11\}$ because 11 is a neighbor of 3 and 5. The last step of this algorithm is that the agent can now choose a peer in $candidates$ for its move.



(a) agent a before its move (b) agent a after its move

Figure 1. Flock of 5 fragments in a 12 peers network. The agent a on peer 5 moves on peer 11 and takes peer 3 as an anchor.

B. Network Structure

In [5], the agents evolve in the Scalable Membership Protocol (SCAMP) [11], a fully decentralized peer-to-peer lightweight membership protocol. Initially designed to support reliable gossip-based multicast protocols, it is used as the network layer for its good properties in terms of scalability and fault-tolerance to achieve the required reliability at network level. SCAMP constructs a random directed graph following the Erdős and Rényi model [12] and having a mean degree converging to $(c+1)\log(p)$ with p the number of peers and c a design parameter used to modulate the expected network degree. This property permits the graph to stay connected if the link failure probability is smaller than $\frac{c}{c+1}$. On top of that, the degree size grows slowly with system size so the network scales well in terms of neighborhood size.

C. Repair Policy

We have enhanced the flocking model with a lazy repair policy. This policy takes advantage of the flock's locality to ease the information dissemination between the agents. Indeed, as displayed in Figure 1, a flock can be seen as a connected subgraph of the logical network. As a consequence, it is possible to elect a leader periodically.

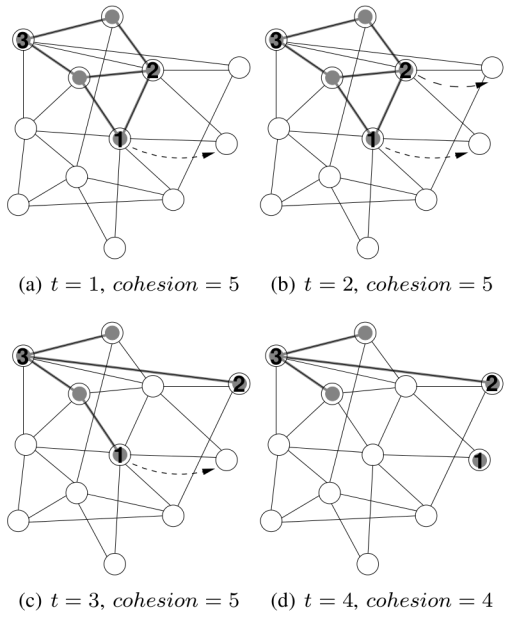
Leader election mechanism: let Δ_t be the time period between two monitoring. This value represents the system's reactivity against faults. Each agent arms a timer $t = \Delta_t + \epsilon$ with ϵ a random choice in a time interval coherent with Δ_t. When this timer expires, the associated agent launches a distributed spanning tree algorithm limited to the flock subgraph. We use an adapted version of the MST [13] algorithm in our implementation. This algorithm is designed to tolerate multiple initiators such that conflicts between multiple leaders does not happen. This is done by merging

partial subtrees until the whole network is covered. When the MST construction is finished, the root of the tree is the leader of the current session. This leader launches a flood of the spanning tree to get informations on the execution site of each agent and on the total number of agents N_a.

Repair mechanism: if $m \leq N_a \leq r$ then the current leader l chooses m agents and asks them to recreate $(m + n) - N_a$ fragments on a free peer adjacent to l. Once this regeneration is done, the flock has its $m+n$ fragments back. As a final procedure, the leader asks the whole flock to rearm its election timers and the current spanning tree is cleaned.

D. Problems generated by the flocking motion

Like in physical flocking, the application of asynchronous and local rules combined with the transit times of the agents can generate ruptures in the flocks cohesion. It is possible, for example, that a flock may be separated into multiple subflocks that can eventually merge together again later. We measure this cohesion value using the flock's graph. We define the cohesion value of a flock as the size of the biggest connected component of its associated graph. For example, in Figure 1, the cohesion value is 5. It happens that the perception an agent had when it decided to move is not true anymore when the motion is over. Such changes in the environment cause cohesion breaks. We saw in the flocking algorithm that an agent a who wants to move relies on anchor peers. These anchors permit a to stay close to the flock after its move. But the anchors are mobile agents too and their move can break the cohesion, as we can see in Figure 2. In Figure 2(a), agent 1 chooses agent 2 as an anchor for its move. But 2 decides to move as well and chooses agent 3 as an anchor while 1 is still moving (Figure 2(b)). Agent 2 finishes its motion before agent 1 due to a quicker network link (Figure 2(c)). Finally, agent 1 finishes its move too and becomes isolated from the flock (Figure 2(d)). The size of the biggest connected component (i.e., the cohesion) is 4 instead of 5. The cohesion variations consequences are immediate on the monitoring and on the repair policies. Given that the monitoring process is based on the flock's connectivity, a supervisor has only a partial vision of the whole flock which is limited to the connected component it belongs to. If we look at the example of Figure 2(d), the elected supervisor in the connected component of agent 2 will obtain a cohesion value of 4 and the one elected in the connected component of agent 1 will obtain a cohesion value of 1. However, the document has still its 5 initial fragments into the network. A first problem occurs when this cohesion rupture crosses the threshold fixed in the lazy repair policy because it has the effect of triggering one or more useless repairs. But the real problem happens when the flock is separated in subflocks such that the cohesion value is less than m. In this particular case, the data is temporarily unavailable because no subflock has enough fragments to rebuild the original data. Preliminary results in [5] suggest



(a) $t = 1$, $cohesion = 5$ (b) $t = 2$, $cohesion = 5$

(c) $t = 3$, $cohesion = 5$ (d) $t = 4$, $cohesion = 4$

Figure 2. Example of a cohesion break caused by a moving anchor in 4 steps.

that it is possible to find a number of fragments fitted for a network instance such that cohesion breaks never goes beyond a certain value. Our contribution is twofold. For a given fragmentation scheme and a given repair policy threshold, we confirm the hypothesis that it is possible to find the suitable number of fragments to prevent the cohesion variations from interfering with the repair policy. We provide several evaluations of the system on different network sizes and we will see that a flock is able to find its steady state after multiple repairs. From this steady state we can deduce the ideal initial fragmentation parameters for each network instance.

IV. Experimental Results

A. Experimental Protocol

We present in this section a set of simulations done on several SCAMP peer-to-peer networks instances. The considered number of peers p is taken in $\{100, 300, 600, 1000, 4000, 10000\}$ and the connectivity constants c for each network instance are taken in $\{4, 10, 20\}$. The aim of these experiments is to evaluate the impact of the network size and its mean degree on a flock's behavior. Remember a SCAMP network of size p has a mean degree converging to an expected theoretical value of $(c + 1)\log(p)$. Note that in practice, this bound is rarely reached. In each experiment, we let a flock evolve in the different considered (p,c)-SCAMP networks. This flock is fragmented using a (10,5)-erasure code and is using a lazy repair threshold $r = 13$. It means that each flock is constituted of 15 fragment at the beginning of each simulation run. The repair mechanism explained in section III-C is

Network Properties		Cohesion x			Fragments Number y		
p	$deg(p)$	$\bar{x}$	σ_x	Var_x	$\bar{y}$	σ_y	Var_y
100	16	18.7	0.7	0.5	18.8	0.7	0.5
300	21	25.6	1.1	1.1	26.1	1.8	3.4
600	25	31	1.3	1.7	33.5	1.15	1.3
1000	27	36.6	1.7	3.07	41.9	1.5	2.3
4000	36	46.4	17.6	**310**	70.3	21.4	**457**
10000	39	37.1	29.6	**879**	84.8	55.7	**3113**

Table I
MEAN COHESION MEASURE AND FRAGMENTS NUMBER OF A FLOCK HAVING 15 INITIAL FRAGMENTS WITH $r = 13$ AND $c = 4$.

Network Properties		Cohesion x			Fragments Number y		
p	$deg(p)$	$\bar{x}$	σ_x	Var_x	$\bar{y}$	σ_y	Var_y
100	25	16.1	0.7	0.5	16.1	0.7	0.5
300	35	22.6	1.3	1.6	23	1.2	1.5
600	45	26.8	1.2	1.4	27.7	1.1	1.3
1000	54	31.6	1.3	1.9	33.8	1.3	1.6
4000	70	44.9	5.9	34.1	55.8	7	49
10000	72	49.6	5.84	76.2	76.7	13.5	183.9

Table II
MEAN COHESION MEASURE AND FRAGMENTS NUMBER OF A FLOCK HAVING 15 INITIAL FRAGMENTS WITH $r = 13$ AND $c = 10$.

active. As a result, the flock triggers a repair process when its cohesion is contained between 10 and 13. With these simulations, we want to observe the repairs triggered by a flock during its evolution to be able to deduce the good initial fragmentation scheme to apply. That's why we didn't add any faults model since it would have biased these results. The simulations have been made with *oRis* [14], a discrete-events and multi-agents simulator. Each simulation has been reproduced 70 times with a run duration of 30000 cycles which represents a simulated period of approximatively 21 days. The network upload bandwidths are contained between 64kB/s and 640kB/s, the size of a fragment is fixed to 64MB and $\Delta_t = 1$.

B. Results

Tables I, II and III present the average results of the simulations done for different sizes SCAMP networks having constants $c = 4$, $c = 10$ and $c = 20$ respectively. These tables show the mean cohesion $\bar{x}$ and the mean number of fragments $\bar{y}$ obtained for each network configuration. They also show the average degree of the network $deg(p)$ in addition to the standard deviation σ and variance Var measures. For example, in Table I, the first line means that in a $(100, 4)$-SCAMP network, the average cohesion value of the flock is 18.7 fragments and its average fragments number during the simulations was 18.8. In this particular case, we can observe that the repair policy has generated 4 more fragments. Another visualization of these results is presented in Figures. 3(a), 3(b) and 3(c). The number of peers is fixed for each plot and each curve displays the evolution of the mean cohesion measure $\bar{x}$ and the observed fragments number $\bar{y}$ for a value of c. We observe that after

Network Properties		Cohesion x			Fragments Number y		
p	$deg(p)$	$\bar{x}$	σ_x	Var_x	$\bar{y}$	σ_y	Var_y
100	29	15.2	0.5	0.2	15.2	0.5	0.2
300	45	18.8	0.7	0.5	18.8	0.8	0.5
600	58	23.6	1.1	1.2	24	1.1	1.2
1000	67	26.5	1.1	1.2	27.5	1	1.1
4000	87	39	2.5	6.7	46.3	2.7	7.5
10000	101	43.8	5.2	27.5	59.3	6.7	45

Table III
MEAN COHESION MEASURE AND FRAGMENTS NUMBER OF A FLOCK HAVING 15 INITIAL FRAGMENTS WITH $r = 13$ ET $c = 20$.

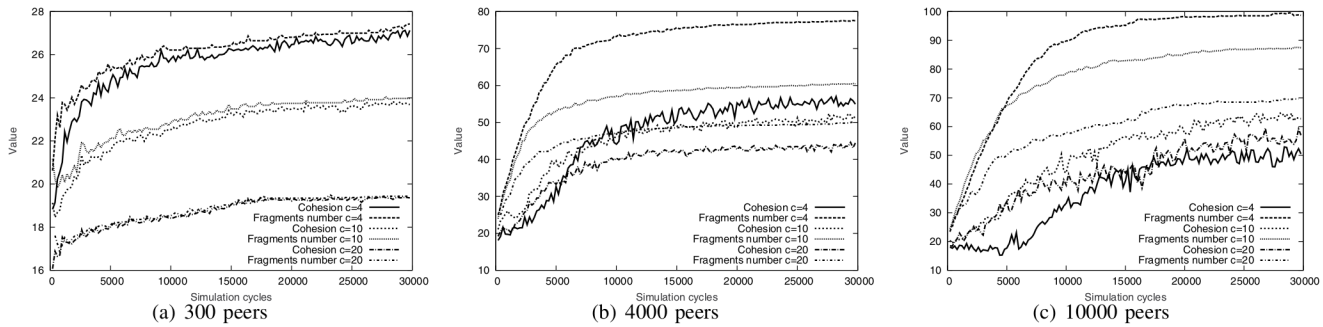
an initial repair period, the values of a flock are stabilizing around the ones given in the previous corresponding tables. It is important to see that for a fixed network size, the increase of c generates less fragments required to find a steady state. In Figure 3(b) when $p = 4000$ and in Figure 3(c) when $p = 10000$, we have a convincing overview of the fact that a large network having a small degree generates subflocks. This behavior is particularly emphasized in the (10000,4)-SCAMP network where the fragments number is twice the value of the cohesion measure.

C. Results Analysis

From these simulations results, we can conclude that each flock is able, after a period of successive repairs, to find the fragmentation parameters fitted for its network. It is materialized by the number of fragments converging toward its mean value. Secondly, it is clear that the required flock size is function of the network properties. More precisely, the size of a flock has to grow when the network size grows to prevent the cohesion breaks from disrupting the monitoring and the repair policies. This property is particularly important in large network instances where the peer number is greater than 1000. We also observe that increasing the mean degree of the network when its size is fixed is reducing the number of generated fragments required by the flock to find its steady state. Finally, from the provided set of tables, it is possible to deduce the initial fragmentation to apply to a flock for this repair threshold. For example, in a $(1000, 10)$-SCAMP network, we read in Table II that a fragmentation scheme that may not disrupt the repair policy for $r = 13$ could be $m = 10$ and $n = 25$ for a total of 35 initial fragments.

V. CONCLUSION

We consider in this paper the problem of documents mobility in peer-to-peer data storage applications. This mobility is obtained according to a flocking model in which documents are fragmented with erasure coding and where these fragments move in groups. We have seen that this group motion is particularly interesting because it relies on asynchronous local rules applied by each agent. But this flexibility has a cost on the fragmentation to apply for a given repair policy. Indeed, It happens that a flock may be



(a) 300 peers (b) 4000 peers (c) 10000 peers

Figure 3. Average cohesion and fragments number evolution across the time for fixed size SCAMP network with a varying c constant.

separated into multiple subflocks that can eventually merge together again later. This instability of the cohesion makes the decentralized monitoring mechanism having a partial vision of the entire flock restricted to its own subflock. This partial perception of the document's global state has the effect of initiating useless repairs when the fragmentation scheme is not adapted to the network properties and to the fixed repair threshold. We saw that when the network size increases, the flock must generate more fragments to obtain a cohesion value that does not interfere with the repair policy. We also saw that, for a fixed network size, increasing the degree lowers the number of generated fragments. Finally, we gave a set of experimental values that can be used to deduce the ideal fragmentation scheme to apply to a flock given a repair threshold. Future works on this topic will focus on finding the relation between the parameters that affect cohesion in ordrer to provide a function that give the mean cohesion of a particular flock in a network instance. This information will give us a better way of finding the expected repair threshold r value.

REFERENCES

[1] J. S. Plank, "A tutorial on reed-solomon coding for fault-tolerance in raid-like systems," University of Tennessee, Tech. Rep. CS-96-332, July 1996.

[2] F. Giroire, J. Monteiro, and S. Pérennes, "P2P storage systems: How much locality can they tolerate?" in *Proc. IEEE LCN*, 2009, pp. 320–323.

[3] O. Dalle, F. Giroire, J. Monteiro, and S. Perennes, "Analysis of Failure Correlation Impact on Peer-to-Peer Storage Systems," in *Proc. P2P*, 2009, pp. 184–193.

[4] P. Druschel and A. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," in *Proc. HotOS*, 2001, pp. 75–80.

[5] H. Pommier, B. Romito, and F. Bourdon, "Bio-inspired Data Placement in Peer-to-Peer Networks - Benefits of using Multi-agents Systems," in *Proc. WEBIST*, 2010, pp. 319–324.

[6] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. IPTPS*. Springer-Verlag, 2002, pp. 328–338.

[7] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. SOSP*, 2003, pp. 29–43.

[8] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: highly durable, decentralized storage despite massive correlated failures," in *Proc. NSDI*. USENIX, 2005, pp. 143–158.

[9] R. B. Kiran, K. Tati, Y. chung Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. NSDI*, 2004, pp. 337–350.

[10] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proc. SIGGRAPH*, 1987, pp. 25–34.

[11] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "SCAMP: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication," in *Proc. NGC*, 2001, pp. 44–55.

[12] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, vol. 5, pp. 17–61, 1960.

[13] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Trans on Programming Languages and Systems*, vol. 5, no. 1, pp. 66–77, 1983.

[14] F. Harrouet, "oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonomes," Ph.D. dissertation, Université de Bretagne Occidentale, 2000.