Search in P2P Triangular Mesh by Space Filling Trees

Nicolas Bonnel, Gildas Ménier and Pierre-Francois Marteau
VALORIA, UBS, Universite Europeenne de Bretagne, France
{*nicolas.bonnel, gildas.menier, pierre-francois.marteau*}@*univ-ubs.fr*

Abstract

We introduce a new P2P exploration strategy based on an extension of space filling curve principles. This stategy is exhaustive and do not generates redundant messages. Initiated at the source node of a search query, a walker is sent to explore the neighbourhood of this node. This walker is carrying an increasing list of visited nodes as an Ariadne-sequence, building at the same time an uncrossable fence. When encountering a fence, this walker splits and its offsprings continue the scanning of the close neighbourhood. The set of walkers creates a tree-like covering structure that ensures that all nodes are visited once and only once. We describe the support overlay and introduce a low cost node heterogeneity management. We conclude with a discussion on experimental validation results.

1. Introduction

In the last few years, the quality (reliability and bandwidth) of Internet connections has increased drastically. During the download of data, the transfert rate is now only limited by the capability of the server to manage a large amount of high bandwith clients at the same time. Thanks to the peer-to-peer (P2P) mechanisms, each user can contribute to this data exchange mechanism as a potential server: this extends the sharing capability and features some fault resilient properties because of the data replication and distribution involved.

Structured P2P networks are very well suited for search process that involves a mapping location/value: for instance DHT based strategy are very efficient when a key should be used to retrieve information [9, 6, 7, 11, 5]. Most of the time, structured P2P tend to equally balance the load among peers, even if these peers have different capabilities, as a result the heterogeneity of the network is often poorly taken into account.

Unstructured P2P networks can greatly benefit from this heterogeneity to dynamically optimize the load of the different nodes. Such networks are well suited to generic distance based queries which make no strong assumption of the location of data: the network has to be scanned as exhaustively as possible, to perform complex query evaluations on each visited node.

This implies an exploration strategy that scans each node of the network, at least one time, and avoids (as much as possible) to scan each node more than one time. Queries can quickly reach a large amount of nodes with flooding [2]: even if this strategy features low latency and high parallelism, it generates a large number of messages. On the other hand, random walk [4] reduces network traffic but induces high latency and is less resilient to nodes failure.

This paper presents the design of a P2P architecture featuring a tree-based search strategy. Peers generates an exploration tree that fills their neihborhood: nodes are visited once and only once in a stable environment and the exploration becomes more and more parallelised as the search progress.

This tree filling strategy needs a triangular mesh to proceed; we describe protocols that maintain at low cost this overlay through arrival and failure of nodes. Performances of this exploration strategy increase in a network with heterogeneous node degrees: this approach is therefore well suited to heterogeneity among peers capabilities [8]. The overall number of generated messages per node needed to maintain the overlay remains constant.

Section 2 describes related works, section 3 introduces our filling tree exploration paradigm. Section 4 presents the overlay designed for the walk. In the last section, we evaluate our approach on a simulated network and discuss the results.

2. Related works

As mentionned before, search can be achieved in unstructured P2P networks using flooding, like in Gnutella [2]. A node initiates a query with a TTL (Time To live) strictly positive and forwards the query to all of its neighbors. Nodes receiving the query decrease the TTL by one and if it is still strictly positive, they forward it to all neighbors excepting the sender. This approach induces low latency and high reliability but quickly generates a lot of messages. Although nodes discard queries already processed, flooding generates redundancy that increases with the TTL of the search (the number of detected cycles in the graph increases with the radius of the exploration).

In [3], Song Jiang et al. proposed to manage the redundancy by the mean of a spanning tree overlay. As redundancy is only significant in flooding above a given radius, the algorithm in [3] is divided in two stages. In the first stage, standard flooding is used for the first hops. In the second stage, messages are only flooded in the suboverlay.

Search using flooding is highly parallelised, hence the flooding process cannot be stopped when a node answers the query. To solve this problem, adaptive flooding has been proposed: successive floods with increasing TTL are performed. According to the policy used, they are called expanding rings [4] or iterative deepening [10]. Although these approaches reduce network traffic to retrieve popular data (i.e. with a high number of replicas), they induce significant overhead for rare data as the first rings are useless.

Random walk [4] reduces network traffic, at the expense of reliability and latency. After being processed on a node, queries are forwarded to a neighbour of this node choosen at random. k-random walk [4] increases reliability and reduces latency, at the cost of additionnal network traffic since k walker are launched in parallel. This approach is well suited for popular data, but when searching rare data, walkers may visit twice or more the same node.

A Gnutella-like system has been proposed in [1]. The system relies on measures performed by [8] that show the heterogenity in capabilities of the peers involved in an unstructured P2P network such as Gnutella. This system uses a biased random walk that visits the nodes with a higher degree first. Each node has pointers on data hosted by its neighbors, and each new node that enters the network tries to connect to a high-degree node. These features increase the overall performance of the system by three to five orders of magnitude.

3. Filling tree exploration

3.1. Principle

We propose to design walkers able to build filling trees to explore a network area. They keep the sequence of the visited nodes, recording a kind-of Ariadne sequence composed with successive visited nodes. This sequence ensures that a node is visited once and only once. On a plannar graph, this sequence builds incrementally an uncrossable fence for the walker. When encountering this limit, the walker splits, as shown in figure 1, and each new walker inherits the previously recorded path. The sequences of paths taken by the walkers is a tree that fills the neighbourhood of the source node P_s, ensuring that all nodes are scanned at least and at most one time.

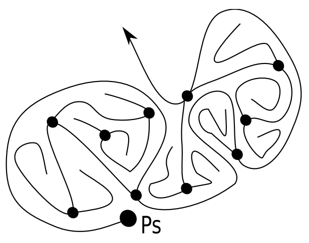

Figure 1. Principle of filling tree algorithm.

Since walkers create clones, this algorithm can be viewed as a kind of k-walk. The advantage of this approach is that, thanks to the properties of the network overlay and the gathering of information about visited nodes, all spawned walkers will not visit the same nodes, whereas in traditional k-random walk this may happen. Contrary to flooding (or LightFlood [3]) filling trees do not need a cache on nodes to store previous seen queries, as there is no discard process.

3.2. Cloning mecanism

Let $N(p)$ be the 1-hop neighborhood of peer p. When a walker visits p, it clusters unvisited nodes of $N(p)$. Because of the triangular mesh, all nodes in $N(p)$ describe a ring. By removing on this ring already visited nodes and their connections, there are between 0 and $x = \lfloor \frac{|N(p)|}{2} \rfloor$ remaining connected components. A clone of the walker is spawned in each of these remaining connected component. The figure 2 shows an example with two connected components. The behaviour of walkers is described in algorithm 1

For $x-1$ connected components, there is a cycle in the graph made from visited nodes (current visited peer is acounted as visited). Walkers that explore these connected components are trapped in deadlocks and cannot explore the remainings of the network. This ensures that no node will be visited more than once. The last walker explores the remaining of the network still accessible.

Algorithm 1: Walker behaviour during exploration

```
Input: start, $V_p$, $TTL_p$
current ← start;
// visited nodes
$V \leftarrow V_p$;
TTL ← $TTL_p$;
while $TTL > 0$ do
    $V \leftarrow V \cup \{\text{current}\}$;
    // unvisited neighbors
    $W \leftarrow N(current) \setminus V$;
    TTL ← TTL - 1;
    if $W = \emptyset$ then
        TTL ← 0;
    else if $TTL > 0$ then
        // nodes to visit
        $T \leftarrow \emptyset$;
        while $W \neq \emptyset$ do
            // connected component
            $C \leftarrow \{randomElement(W)\}$;
            $W \leftarrow W \setminus C$;
            clusterGrow ← true;
            while clusterGrow do
                clusterGrow ← false;
                forall $p \in cluster$ do
                    if $N(p) \cap W \neq \emptyset$ then
                        $C \leftarrow C \cup (N(p) \cap W)$;
                        $W \leftarrow W \cap N(p)$;
                        clusterGrow ← true;
                    end
                end
            end
            dmax ← $max(|N(p)|)\forall p \in C$;
            $C \leftarrow \{p \in cluster, |N(p)| = dmax\}$;
            $T \leftarrow T \cup \{randomElement(C)\}$;
        end
        current ← $randomElement(T)$;
        $T \leftarrow T \setminus \{current\}$;
        forall $p \in T$ do
            launchWalker(p,$V$,TTL);
        end
    end
end
```

High degree nodes are likely to be connected together, so that walkers are likely to hit fences and to clone themselves. Our approach takes advantage of this property to increase parallelism by visiting the largest 2-hops neighborhood first. We use the 2-hops neighborhood instead of a 1-hop neighborhood because many high degree nodes are connected through a third party node.

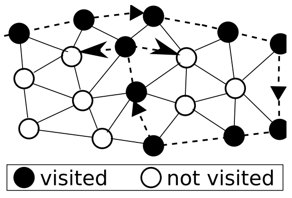

Figure 2. Example of a cloning case.

When the request carried out by the walker is fullfilled, it returns to the initial node using the inverse path it has recorded. The walker terminates if all neighbors are already visited (no connected component is detected) or if its TTL reaches 0.

4. Overlay

We assume that each node on the network has a unique identifier. Nodes refresh the knowledge of their 2-hop neighborhood by sending their identifier and their 1-hop neighbors identifiers to their neighbors. Futhermore, peers maintain a list of all the triangles they belong to. The size of this list is equal to the node's degree.

The overlay is maintained through the use of protocols for node arrival and failure/departure. These protocols induce a number of messages that are proportional to node degrees. Because the average node degree in a triangular mesh is nearly constant (or tends to six), the bandwidth that is used and the number of generated messages remains very limited. Not only this adaptive mesh layer is important to support our strategy of exploration, but also, it offers another interesting property: it allows to promote high resource nodes with more connexions, thus performing locally load balancing.

4.1. Node arrival

A peer p joining the network contacts a random node p_1 in the network. The way this random node p_1 is obtained can be achieved in various ways (for instance with a central server registering few nodes taken at random) and is not addressed here. A triangle (p_1, p_2, p_3) in the list of triangles containing p_1 is then taken at random. New connections are created between $p - p_1$, $p - p_2$ and $p - p_3$.

The lists of triangles of p_1, p_2 and p_3 are then updated. When a node connects to the network, it has a de-

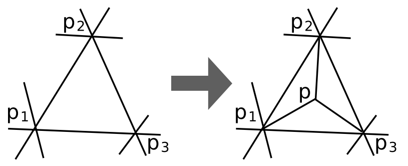

Figure 3. Node arrival.

gree of three and increases the degree of the three nodes by one. Hence the average node degree tends to six as the size of the network grows.

4.2. Node departure/failure

We make the assumption that a node leaving the network is equivalent to a failure of this node. A failure on a node p_f is detected by its neighbors when they do not receive a ping message of p_f within a given timeout. If a failure is detected, then the overlay needs to be repaired in order to maintain the triangular mesh, except if the failing node had a degree of three. In this case, the node departure can be seen as the inverse of a node arrival, as shown in figure 3.

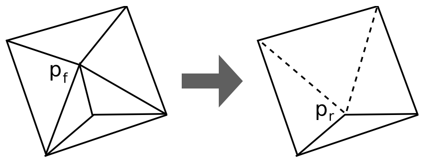

Figure 4. Topology repairment after the failure of node p_f. The two neighbors of p_r cannot be candidate for handling the repairment because they have three neighbors in $N(p_f)$.

The departure/failure of a node p_f with a degree higher than three makes a hole in the mesh. This hole is delimited by $N(p_f)$ and its size is $s = \|N(p_f)\| - 3$. This hole can be repaired by creating s connections. The repairment process is handled by a node p_r that connects to other nodes of $N(p_f)$ excepts its neighboors and itself, as shown in figure 4. As it creates $\|N(p_f) \setminus N(p_r)\| - 1$ connections, it must have only two neighbors in $N(p_f)$ to be able to create s connections. Moreover, its two neighbors in $N(p_f)$ must have a minimal degree of 3 so that there are no nodes with a degree of two after the repairment. However breaking this second property invalidates the first one, as a node with a degree of 2 will have its 2 neighbors in $N(p_f)$ connected together, hence the first property is sufficient: $\|N(p_f) \cap N(p_r)\| = 2$.

To select P_r, we assume an order relationship between nodes identifier. The node in $N(p_f)$ with the lowest identifier checks if it can handle the hole repairment. This node knows it has the lowest identifier in $N(p_f)$ because of the neighborhood knowledge he previously received from p_f. If the node cannot handle the repairment process, it launches a token on the ring (nodes in $N(p_f)$ describe a ring - see section 3.2). Each node on the ring checks if it can handle the repairment process, and if it cannot, the token is forwarded on the ring.

If all nodes in $N(p_f)$ have a degree of two, they are all disconnected from the remaining of the network. Let p_c be a node that has a connection with another node not in $N(p_f)$. Because of the triangular mesh, the two neighboor of p_c in $N(p_f)$ are connected to this other node and there is a sequence of 3 nodes in $N(p_f)$ that have at least a degree of 3. Because the overlay is a planar graph, p_c cannot be connected to other nodes in $N(p_f)$ and satisfy the first constraint. This ensure that at least one node fullfilling the two requested features belongs to the ring (and will be found by the repairment process). This repairment process ensures that the network keeps the same overall shape.

4.3. Peers connectivity

Peer-to-peer networks feature heterogeneity amongs peer resources as shown in [8]. As some peers have more bandwidth or CPU, they can process more queries than other peers. Therefore, nodes should have a number of neighbors proportionnal to their amount of available resources. This heterogeneity among peers connectivity is achieved by switching peers neighborhood.

If a peer p_1 has more resources and a degree smaller than one of its neighboor p_2, it may disconnect from its current neighbors except p_2, and connect to all neighbors of p_2, except itself. p_2 performs the same process and exchanges its neighborhood with the one of p_1. As the repairment, this optimizing process ensures that the network overall shape remains unmodified.

5. Experiments

Experiments have been performed on a dedicated simulator developped in Java. At initialization, the network is made of four nodes connected together: the overlay is a tetrahedron, with all nodes having a degree of four. The average node degree gets closer to six as more nodes are added to the network with the process described before.

We compare our approach with flooding [2], k-random walk [4] and LightFlood [3]. We run the algorithms on a random power-law graph with an average degree of six. Simulated networks contain 100000 nodes. Figure 5 shows an example of unfolded tree generated while exploring a part of the network. In our simulation, one third of the nodes have a degree of three, hence, as described in section 4.2, their failure do not need to be repaired.

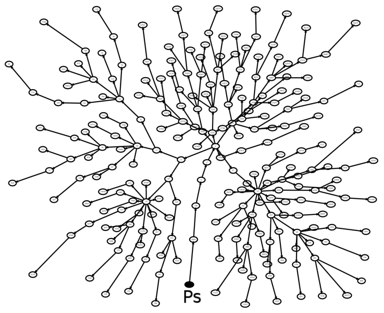

Figure 5. Filling tree resulting of an exploration with a TTL = 12 starting from node P_s. 236 nodes were visited. The node positions in the unfolded tree do not reflect their position in the overlay.

5.1. Coverage

Figure 6 shows the number of visited nodes according to the TTL of the exploration. Flooding has the best coverage for the lowest TTL, however this is as the cost of redundant messages, as shown in figure 7. LightFlood features good coverage at low TLL : 95% of nodes are discovered with a TTL of 10.

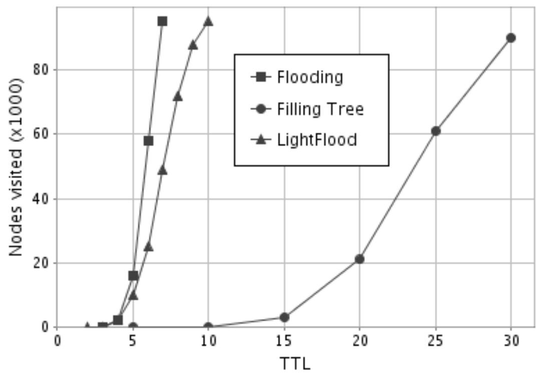

Figure 6. Network coverage.

Our approach offers the same coverage as the LightFlood approach at the expense of an extended TTL : but unlike Lighflood, our system does not need a cache storage on each visited node. A TTL of 30 remains quite acceptable for a network of 100000 nodes. Random walk is omitted here as it has poor coverage performances.

Moreover, it takes about 40 hops for filling trees to fully cover the network, while LightFlood covers 98.2% of the network with a TTL of 100. This lack of exhaustivity in LightFlood is explained in [3]: flooded messages may collide with each other within the spanning tree and be discarded.

5.2. Redundancy

Let v be the number of visited nodes and m be the number of generated messages. The percentage of redundant messages is evaluated to $100 \times (\frac{m}{v} - 1)$. Figure 7 shows the percentage of redundant messages according to the number of visited nodes on a static overlay. We measure messages redundancy for flooding, k random walk with 100 walkers, LightFlood with 4 hops of pure flooding and our approach.

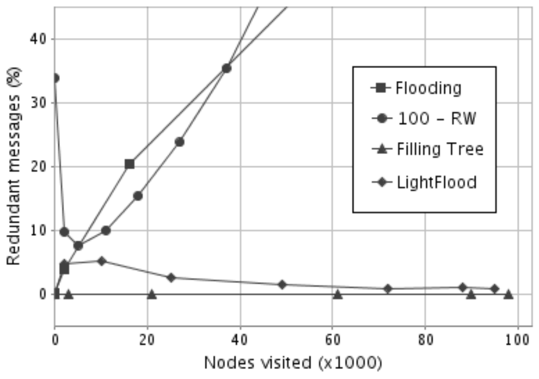

Figure 7. Message redundancy.

This experiment shows that random walk and flooding do not scale well. While they are suited for visiting few nodes (up to 20k nodes in our simulation) the percentage of redundant messages increases with the number of visited nodes. LightFlood generates up to 5 % of redundant messages after 4 hops of pure flooding. Then redundancy decreases until reaching roughly one percent for visiting 95k nodes. As shown on Figure 7, filling tree still does not generate redundant messages.

5.3. Node departure

We measure the coverage of filling trees as more and more nodes (taken at random) leave the network. We evaluate our approach both on a fixed overlay (each time a node is removed, the overlay is repaired) and on a broken overlay (never repaired after a node departure). We compare our approach with LightFlood with a repair process that keeps average node degree constant. To have similar coverage for both approaches, we set the TTL for LightFlood to 8 after 4 hops of pure flooding, and set the TTL of filling trees to 25. Results of this experiment are plotted on figure 8.

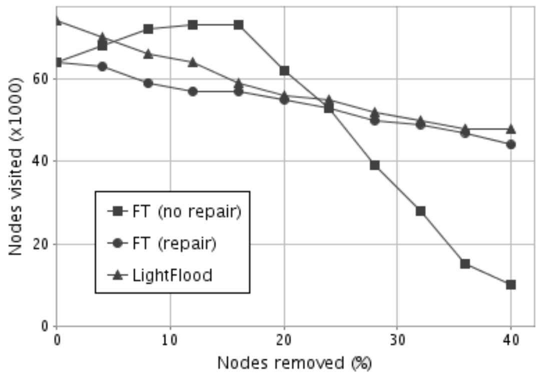

Figure 8. Coverage with node departures

When repairing topologies, the number of nodes covered by LightFlood and filling trees decreases as more and more nodes are removed from the network. Because of the settings we use, the number of nodes covered by LightFlood is slightly greater than the number of nodes covered by filling trees but the gap between the two approach is reduced as more and more nodes are removed. This shows the efficiency of our repairment process.

On a broken overlay, the number of nodes covered by filling trees increases while up to 15% of nodes are removed but decreases quickly when more of 15% of nodes are removed. Having holes into the mesh increases walker cloning rate, since holes increase the number of different connected components of unvisited nodes while analyzing the neighborhood of nodes. However, because of the shape of the network, there is still no redundancy. Of course, while the number of covered nodes increases, the number of reachable nodes (with an infinite TTL) slightly decreases.

This feature is very interesting because as node churnning rate creates holes, a dynamic overlay is never fully repaired. As our approach benefits from this property, this shows that it is very well adapted to dynamic environments.

6. Conclusion and future work

We introduced a strategy for the search in an unstructured P2P network based on the unfolding of an exploration tree. This tree features the evolution of a collection of walkers that explores the mesh in a parallel way, speeding up the search. We designed a low cost maintained overlay to support this process. We show that our approach takes into account the natural heterogeneity of the underlying network. The experiments suggest very promising preliminary results on both stable and dynamic environments.

7. Acknowledgements

This research was supported by Region Bretagne.

References

[1] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable, 2003.

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

[3] Song Jiang, Lei Guo, Xiaodong Zhang, and Haodong Wang. Lightflood: Minimizing redundant messages and maximizing scope of peer-to-peer search. *IEEE Transactions on Parallel and Distributed Systems*, 19(5):601–614, 2008.

[4] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks, 2001.

[5] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[6] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

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

[8] S. Saroiu, K. Gummadi, and S. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts, 2003.

[9] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[10] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[11] 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, April 2001.