Computer Science Journal of Moldova, vol.**14**, no.3(42), 2006

Graph Coloring using Peer-to-Peer Networks*

Adrian Iftene Cornelius Croitoru

Abstract

The popularity of distributed file systems continues to grow in last years. The reasons they are preferred over traditional centralized systems include fault tolerance, availability, scalability and performance. In this paper, we propose a framework for analyzing peer-to-peer content distributed technologies and their applications in the cooperative solving of combinatorial optimization problems. Our approach, which follows the Content Addressable Network model, is scalable, fault-tolerant and self-organizing; we improved also load distribution at the insertion and deletion of nodes. We use this network for the classical "graph coloring" problem, in order to reduce the computational time for its cooperative solving.

Keywords: CAN, graph coloring, peer-to-peer network

1 Introduction

In the last years, Peer-to-Peer system research has grown significantly. Using a large scale a distributed network of machines has become an important element of distributed computing due to the extraordinary popularity of Peer-to-Peer services like Napster, Gnutella, Kazaa and Morpheus. Though these systems are famous mainly for file sharing, we can see how P2P (Peer-to-Peer) systems are becoming a very good area for research.

P2P systems offer a decentralized, self-sustained, scalable, fault tolerant and symmetric network of machines providing an effective balancing of storage and bandwidth resources.

*A preliminary version of this paper was presented at 5^{th} RoEduNet International Conference, Sibiu, Romania, 1-3 June 2006

Unfortunately, most of the current peer-to-peer designs are not scalable. For example, in Napster a central server stores the index of all the files available within the Napster user community and all the user queries for a specific file are sending to this computer. This server sends the answer with IP address, which has the file, and after that the user can download the file directly from this machine. In this way, the process of locating a file is very centralized and makes it very vulnerable (since there is a single point of failure). Gnutella goes one step further and decentralizes the file location process as well. Users in a Gnutella network self-organize into an application-level mesh on which requests for a file are flooded with a certain scope. Flooding on every request is clearly not scalable and, because the flooding has to be curtailed at some point, may fail to find content that is actually in the system.

In this paper we describe the design of a peer-to-peer file system *completely distributed* (it requires no form of centralized control, coordination or configuration), *scalable* (nodes maintain only a small amount of control state that is independent of the number of nodes in the system) and *fault tolerant* (nodes can route around failures) which is used to cooperative solve the graph coloring problem.

Our interest in CANs (Content Addressable Networks) is based on the belief that this abstraction would give a powerful design tool that could enable new applications and communication models.

Section 2 of this paper presents the P2P networks and their main design issues, while section 3 describes our approach based on CAN and specific operations from this class of networks: insertion and deletion of nodes. Section 4 presents the main algorithm used for graph coloring. Then, section 5 presents the results of our tests on classical graphs. Finally, section 6 discusses the feasibility of the approach in practical settings and briefly presents some further developments.

2 P2P Networks

The term Peer-to-Peer (P2P) refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner. Each node from this system has the same re-

sponsibility. Basic P2P system goals are decentralization, immediate connectivity, reduced cost of ownership and anonymity. Androutsellis-Theotokis [1] defined P2P as "*applications that take advantages of resources (storage, cycles, content, human presence) available at the edges of the Internet*".

To make ubiquitous computing become a reality, the computing devices must become reliable, resilient and have distributed access to data. The P2P architecture can help reduce storage system costs and allow cost sharing by using existing infrastructure and bundling resources from different sites. Considering these factors, the P2P model should be very useful in designing the future generation distributed file systems.

2.1 Design Issues in P2P Networks

Peer-to-Peer systems have basic properties that separate them from conventional distributed systems. We describe in this section different design issues and their effect in system performance [2].

Symmetry: Symmetry comes among the roles of the participant nodes. We assume that all nodes have the same characteristics. In client-server systems usually the servers are more powerful than the clients. In our network each node has the ability to be server and client in the same time.

Decentralization: P2P systems are decentralized by nature, and they have mechanisms for distributed storage, processing, information sharing. In this way scalability, resilience to faults and availability are increased. But, in the same time our tries to see global system and its behavior are without success.

Operations with Volunteer Participants: An important design issue is that the participants can neither be expected nor enforced. They haven't a manager or a centralized authority and can be inserted or removed from the system at any time. The system should be robust enough to survive the removal or failure of nodes at any moment.

Fast Resource Location: One of the most important P2P design is the method used for resource location. Because resources are distributed in diverse peers, an efficient mechanism for object location becomes the deciding factor in the performance of such system. Currently, object location and routing systems include Chord, Pastry, Tapestry and CAN. Pastry and Tapestry use Plaxton trees, basing their routing on address prefixes that is a generalization of hypercube routing. However, Pastry and Tapestry add robustness, dynamism and self-organizing properties to the Plaxton scheme. Chord uses the numerical difference with the destination address to route messages. The Content Addressable Network (CAN) uses a d-dimensional space to route messages. Another important location strategy used in few systems is Distributed Hash Table (DHT). It uses hashing of file or resource names to locate the object.

Load Balancing: Load balancing is very important in one robust P2P system. The system must have optimal distribution of resources based on capability and availability of node resources. The system should prevent the building of locations where the load is high and also should be possible to re-balance the system based on usage patterns.

Anonymity: With scope to prevent censorship we need to have anonymity in our system. Our network must assure anonymity for producer and for consumer of information.

Scalability: The number of peers from network should be of any value (hundreds, thousands or millions), and that should not affect the network behavior. Unfortunately actual systems are not scalable over few hundreds or thousands of nodes.

Persistence of Information: Methods from our system should be able to provide persistent data to consumers, the data stored in the system is safe, protected against destruction, and highly available in a transparent manner.

Security: Security from attacks and system failure are design goals for every system. The system should be able to assure data security, and this can be achieved with encryption, different coding schemes, etc.

2.2 Existing Systems

Designing a system that can implement all properties from above is very difficult. Most of existing systems utilize specific properties or mechanisms, and that involves some advantages and also some disadvantages.

FreeNet: This system presented in [3] is an adaptive P2P that enables the publication, replication and retrieval of data while protecting the anonymity of the authors, readers and data location. It uses probabilistic routing for that, and it works over many individual computers that allow files to be inserted, stored and requested. The file identification is made with a hash function. Any request may be locally processed, or on failure may be routed to the closest neighbor accordingly to the routing table. Communications between Freenet nodes are encrypted.

CFS: Cooperative File System (CFS) [4] is a peer-to-peer system developed by MIT with the following design goals: provable for efficiency, robustness, load balancing and scalability. CFS uses DHT for storage of blocks. The file system is being built like a set of blocks distributed over the CFS servers. CFS has three layers: FS, which interprets blocks as files and presents an interface to applications; Dhash - the distributed hash table that stores unstructured data blocks; and *Chord*, which maintains routing tables for lookup and query manager. CFS is read only system from user's perspective and authentication is based on keys.

OceanStore: OceanStore [5] is a prototype system to provide distributed access at persistent data in a uniform way. It is designed using a cooperative utility model in which consumers pay the service providers to ensure access to persistent storage. Using mainly untrusted servers, OceanStore caches data anywhere in the network, with encryption. This provides high availability and prevents attacks. Persistent objects are uniquely identified by a Global ID (GUID) and are located by either a non-deterministic but fast algorithm or a slower deterministic algorithm.

Farsite: Farsite [6] is a symbiotic, server-less, distributed file system.

It works between the cooperating clients, but not completely trusting the each other. Data stored in the servers is encrypted and replicated in a non-Byzantine way. Farsite first encrypts the contents of the files to prevent unauthorized reads. Digital signatures are used to prevent an unauthorized user to write a file. Replication provides reliability through long-term data persistence and immediate availability of requested file data.

3 CAN

The term CAN stands for *Content Addressable Network* [7], which is used like a distributed hash table. The basic operations performed on a CAN are the insertion, lookup and deletion of nodes. Our implementation provides a completely distributed system (we haven't centralized control, coordination or administration), scalable (every node has a small number of neighbors that is independent of the total numbers of nodes from system), and fault-tolerant (we support node failures).

3.1 CAN Construction

Our design is over a virtual *d-dimensional* Cartesian coordinate space. This space is completely logical and we haven't any relation between it and any physical system. In this *d-dimensional* coordinate space, two nodes are neighbors if their coordinates concur overlap along d-1 dimensions and differ over one dimension. In this way, if all zones from this space are approximatively the same, every node has $2d$ neighbors. To allow the CAN to grow, a new node that joins this space must receive its own portion (zone) of the coordinate space.

3.2 CAN Insertion

The process has three steps:

- The new node must find a node that is already in CAN (*source node*).

- After that, it knows CAN dimensions and it generates a *d-point* in this space (this point is in a node zone - *destination node*). We route from *source node* to *destination node* following the straight-line path through the Cartesian space. The *destination node* then splits its zone in half and assigns one half to the new node.

- In the last step, the neighbors of the split zone must be notified about new node from CAN.

Insertion - Main Procedure

The new computer is *wil* and the destination computer is $nameMin$.

void CanInsertion(WriteLocalImpl wil){
  if ($nameMin$ is available){
    Let $maxCoordinate$ be the first coordinate with max width;
    Create a new hyper-parallelepiped for the new node *writeImplLocal* with the following dimensions:
      * all dimensions except $maxCoordinate$ are those of the destination computer;
      * $maxCoordinate$ is half of the destination computer;
    $neighborhoodsListRestore()$;
  }
}

Interesting is the *neighborhoodsListRestore* procedure which inserts (at beginning of the list) all neighbors, after obvious positive tests.

3.3 CAN Deletion

When nodes leave the CAN, we need to ensure that their zones are taken by the remaining neighbors. If the zone of a neighbor can be merged with the node's zone to produce a valid single zone, then this is done. If not, then the zone is splitted in zones accordingly with neighborhoods structure. In some cases, it is possible that this zone to remain non-allocated, but at the first occasion it will be used.

```
void CanDeletion(WriteLocalImpl wil){
  deleted = false;
  for(int i = wil.getNeighborhoodsNumber(); i >= 1; i--){
      if(wil.getNeighborhoodIsAccesible(i) == true){
          nameServer = wil.getNeighborhoodName(i);
          wis = WriteLocalHelper.narrow(ncRefAux.resolve_
          str(numeServer));
          wis.delNeighborhoods(nameComp);
          neighborhoodsListRestore();
      }
  }
}
```

4 Graph Coloring

The Graph Coloring Problem (GCP) can be simply stated as follows: *given a graph $G(X, E)$, assign a color to each vertex in X such that no two adjacent vertices have the same color and such that the number of colors used is as least as possible.* This minimum possible number of colors is known as the chromatic number of the graph G and it is denoted by χ(G).

The GCP is well studied and has many applications including scheduling, timetabling and the solution of sparse linear systems. However it is also known to be one of the most difficult combinatorial optimization problems. Not only is the problem of finding χ(G) NP-hard, but Lund and Yannakakis [8] have even shown that, for some $\epsilon > 0$, approximate graph coloring within a factor of N^{ϵ} is also NP-hard.

We want to improve existing parallel solutions (e.g. [9]) with a new objective: speed in finding solutions. We consider a peer-to-peer cooperative approach to the graph coloring problem and look at its potential to aid the search for good solutions in reasonable time. Our algorithm divides the initial problem in sub-problems and involves all neighbors of main node in the solution search.

4.1 Algorithm Phases

In *first phase*, the input vertex set X of the graph $G = (X, E)$ is partitioned into p sub-sets $\{X_1, X_2, ..., X_p\}$, cardinality balanced. The subgraphs induced by each subset are sending then to p available neighbors of the main node, for solving.

In *second phase*, every node that receives one subgraph solves directly the problem (if the size of the problem is less than one specific value) or, recursively, splits the problem and sends the sub-problems to its available neighbors.

In *third phase*, the main node receives the partial solutions from its neighbors and tries to combine them into final solution. In this phase the main node finds "conflicts".

Last phase solves the "conflicts" and builds the final solution (the solution for the initial problem).

4.1.1 Phase I

For splitting, we used three methods:
1) Separator Theorem (Tarjan & Lipton, 1979) ([10]):
For every planar graph G with n vertices, there is a partition of $X(G)$ in disjoint sets A, B, S with the following properties:

- A and B are $S-$separated: in $G - S$ there is no edge linking a vertex from A to a vertex from B.
- $|A| \leq \frac{n}{2}, |B| \leq \frac{n}{2}$
- $|S| \leq 4\sqrt{n}$

We start from node s and we perform a breadth first search (BFS) on this graph. We mark every node with its level from BFS tree. $L(t), 0 \leq t \leq l + 1$, denotes the set with nodes from level t, and t_1 is the middle level (the level containing the node $\frac{n}{2}$).

Let t_0 be the higher node with $|L(t_0)| \leq \sqrt{n}$ and $t_0 \leq t_1$. Let t_2 be the lower node with $|L(t_2)| \leq \sqrt{n}$ and $t_1 \leq t_2$. Consider $C = \cup_{t<t_0} L(t), D = \cup_{t_0<t<t_2} L(t), E = \cup_{t_2<t} L(t)$ (like in the following figure).

Figure 1. *Node Separation with Tarjan & Lipton*

If $|D| \leq \frac{2}{3}n$ we can take $S = L(t_0) \cup L(t_2)$, A – the set of maximum elements from C, D, E and B – the union of the other two. If $|D| > \frac{2}{3}n$, then we must find a new separator for D.

With this method, we can split our initial problem only into three problems, and we lose a lot of time with splitting. The main advantage for this method is actually in the following phases, because we don't have many edges between our sets of nodes.

2) Separation based on neighbor's number

In this case the input vertex set X of the graph $G(X, E)$ is split into p sub-sets as $\{X_1, X_2, ..., X_p\}$ with the same number of nodes ($X_1 = \{1, .., \frac{n}{p}\}, X_2 = \{\frac{n}{p}+1, ..2\frac{n}{p}\}, ..$). p is the number of neighbors of current node, which can be involved in problem resolution.

The separation is made very fast, but we have a lot of work to do in the following phases, since, usually, we have many edges between our sets of nodes.

3) Random separation

In this case the input vertex set X of the graph $G(X, E)$ is split into

p sub-sets as $\{X_1, X_2, ..., X_p\}$, where p is the number of neighbors of current node, which can be involved in problem resolution. Every set has random nodes, but their cardinalities are balanced.

The separation is done very fast, but in the following phases we have omnibus-volume of work that depends of the current step.

For each of the above methods we have situations in which this method is better and obtains the minimal coloring.

Phase II

For coloring we use **Greedy Coloring Algorithm** ([11]): visit the list $X(G)$ from the left to the right and for each current node assign a color with the minimum value unused by its neighbors.

Phase III

Two nodes are in "conflicts" if they are in different sub-sets and receive the same color, and are adjacent in the initial graph.

Phase IV

"Conflicts" elimination can be done in two ways:
-**On-line**, when a conflict is detected, it is solved immediately;
-**Off-line**, only after all conflicts are detected, they are solved one by one.

Each method has advantages and disadvantages and depends on the input graph. For the *on-line* method we have the advantage that it destroys future conflicts. The disadvantage appears because number of graph colors is increased and the solution decreases in quality. The same kind of problems could appear for the *off-line* method.

4.2 Algorithm Code

We used a program that generates random graphs of large order and provides the output in XML format. The main procedure for coloring is the following:

public void graphColoring(graph G){
    if(G is too big){
        $graphSeparation(nNeighbors)$;
        for($int\ j = 1; j <= nNeighbors; j++$){

```
                graphColoring(G_j);
            }
        }
        else{
            graphColoring();
        }
        solutionsCombination();
        conflictsElimination();
}
```

This code is for the "off-line" case, in the "on-line" case procedure *conflictsElimination* appears inside of procedure *solutionsCombination*.

4.3 Tests and Results

In order to evaluate our algorithms we used a number of classical graphs, from the DIMCS Challenge([12]). We can see that the first method of separation problems (Separation Theorem (Tarjan & Lipton, 1979)) is the better method in comparison with second method of separation (Table 1), but sometimes the random method is able to give us the lower number of colors.

However, for every method of separation we have a graph in which it provides the better solution. We have the same behavior for "on-line" and "off-line" methods.

We work in a 2-dimensional CAN and we remark that the number of final colors depends on the peer-to-peer network structure in linear way and also, depends on the number of neighbors involved in resolution. In Table 1 we put with bold the results for random case which appear often.

Last two columns from Table 1 contain the execution time results obtained: first – "*Local*" – doesn't use our network advantages and solves the problem locally, and the second – "*P2P*" – uses a 2-dimensional network for solving.

Our algorithm solutions are the optimal solutions for small graphs, but aren't in almost all cases for big graphs. First off all, because **Greedy Coloring Algorithm** doesn't offer the optimal solution, and

Table 1. Test Results

ID	Nodes	Edges	χ	Theorem	Order	Random	Local	P2P
Myciel4	23	71	5	5	6	**5**, 6	0s	0s
Myciel5	47	236	6	6	7	**6**, 7	0s	0s
Myciel7	95	755	7	7	8	7, **8**	0s	0s
Anna	138	493	11	12	12	11, **12**	1s	2s
David	87	406	11	12	12	11, **12, 13**	0s	0s
Huck	74	301	11	11	11	11	0s	0s
Jean	80	254	10	11	11	**10**, 11	0s	0s
Games120	120	638	9	10	9	**9**, 10	1s	1s
Queen5-5	25	160	5	7	8	6, 7, **8**, 9	0s	1s
Queen6-6	36	290	7	9	13	**9**, 10, 11	0s	1s
Queen7-7	49	476	7	12	13	10, **11**, 12	0s	1s
Queen8-12	96	1368	12	15	17	**15**, 16, 17	0s	1s
Queen8-8	64	728	9	13	16	**12, 13**, 14	0s	1s
Queen9-9	81	2112	10	14	16	13, **14**, 15	0s	1s
Queen10-10	128	3216	12	18	17	14, **15**, 16	1s	2s
Queen16-16	256	12640	16	25	32	25	30s	27s
Le450-5a	450	5714	5	13	15	13, **14**	21s	14s
Le450-5b	450	5734	5	13	15	14	20s	13s
Le450-5d	450	9757	5	18	18	18	44s	26s
Le450-15c	450	16680	15	30	33	32	110s	79s
Le450-15d	450	16750	15	30	32	32	129s	85s
Le450-25c	450	17343	25	36	37	39	145s	90s
Le450-25d	450	17425	25	35	36	38	127s	78s

secondly, when we combine solutions and we have conflicts, the procedure $conflictsElimination$ can increase the number of colors.

According with the results, our approach is not relevant for small graphs, where the times are similar, but is very useful for large graphs with a big number of nodes and edges, where results are very promising.

5 Summary and Future Work

First phase (initial problem splitting) and last phase (conflict elimination) can be improved, but these depend on graph generation. Our algorithm used for graph generation obtains the same structure of graphs and we have not a large variation of graph types.

Also, it would be interesting to see what happens in fail-over sit-

uations: in this case the sub-problem must be sent again to another neighbor for solving. In the future, this kind of approach can help us in graph coloring with a better speed and with a solution close to the optimal.

With characteristics like decentralization, symmetry, robustness, availability and persistence of data, the P2P distributed file system is now an important part of file system research and can be involved with success in future in combinatorial solving of problems.

References

[1] A.T. Stephanos, S. Diomidis. *A survey of peer-to-peer content distribution technologies*, ACM Computing Surveys, Vol. 36, No. 4, December 2004.

[2] H. Ragib, A. Zahid. *A survey of peer-to-peer storage techniques for distributed file system*, National Center for Supercomputing Applications Department of Computer Science, April 2005.

[3] I. Clarke. *Freenet: a distributed anonymous information storage and retrieval system*, Workshop on Design Issues in Anonymity and Unobservability, pp. 46–66., 2000.

[4] F. Dabek. *Wide-area cooperative storage with CFS*, Usenix SOSP, 2001.

[5] J. Kubiatowicz. *OceanStore: an architecture for global- scale persistent storage*, ACM ASPLOS, 2000.

[6] A. Adya. *FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment*, Usenix OSDI, 2002.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A scalable content-addressable network*, In Proc. ACM SIGCOMM 2001, August 2001.

[8] C. Lund, M. Yannakakis. *On the hardness of approximating minimization problems*, Journal of the ACM, 41(5):960–981, 1994

[9] A. H. Gebremedhin, F. Manne. *Scalable parallel graph coloring algorithms*, University of Bergen, Norway, Concurrency: Pract. Exper, 2000.

[10] R.J. Lipton, R. E. Tarjan. *A separator theorem for planar graphs*, SIAM Journal on Applied Mathematics 36, 2, 177–189, 1979.

[11] J. Culberson. *Iterated greedy graph coloring and the difficulty landscape*, Tech. Rep. 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, 1992.

[12] Graphs used in the DIMACS Challenge (Discrete Mathematics & Theoretical Computer Science). DIMACS Center CoRE Building Rutgers, The State University of New Jersey 96 Frelinghuysen Road Piscataway, NJ 08854-8018. Graphs are available at address: http://mat.gsia.cmu.edu/COLORING02/index.html

Adrian Iftene, Cornelius Croitoru Received October 10, 2006

"Al. I. Cuza" University, Faculty of Computer Science,
General Berthelot, 16, 700483, Iasi
E-mail: *adiftene@infoiasi.ro*