Fault Tolerance Logical Network Properties of Irregular Graphs

Christophe Cérin
Université Paris 13 - PRES Sorbonne Paris Cité
LIPN UMR CNRS 7030
99 avenue Jean-Baptiste Clément
F-93430 Villetaneuse (France)
christophe.cerin@lipn.univ-paris13.fr

Camille Coti
Université Paris 13 - PRES Sorbonne Paris Cité
LIPN UMR CNRS 7030
99 avenue Jean-Baptiste Clément
F-93430 Villetaneuse (France)
camille.coti@lipn.univ-paris13.fr

Michel Koskas
Institut National de la Recherche en Agronomie
Département de mathématique, UMR 518 (MIA. INRA)
16, rue Claude Bernard
F-75231 Paris Cedex 05 (France)
michel.koskas@agroparistech.fr

ABSTRACT

Assume a desktop grid middleware or a deployed cloud infrastructure that are both based on a large number of volunteers for computanation-intensive applications or business applications. In this case, the Internet is the communication layer; hence, the communication graph is not regular. Scalability and fault tolerance issues are implicitly present on any platform. For instance, the overlay network that must be built to control the application as part of the run-time support system needs to be scalable and fault tolerant. In this paper, we focus on the fault tolerance properties of large, irregular graphs that may be used as models for the Internet. In a previous work, we presented algorithms and a framework for computing fault tolerance properties of different variants of randomly-generated binomial graphs (BMG). In the present paper we compute various metrics, and among them the node and link connectivities and the fault diameter. We also compare our implementation of the diameter computation with the work of Magnien *et al.*

Keywords

Large scale systems, models for overlay networks, fault tolerance, performance evaluation, performance measurement, graph algorithms

1. INTRODUCTION

1.1 Context

Desktop Grids and Volunteer Computing are well-known terms related to the notion of computing on loosely connected resources, usually personal computers over the Internet but also sometimes clusters, that are controlled by their owners. These terms appeared in the Internet and research community at the end of the 90s. Nowadays, this type of computing platform forms one of the largest distributed computing systems, and currently provides scientists with Petaflops from hundreds of thousands of hosts[1].

SlapOS [20] is an open-source grid operating system for distributed cloud computing. SlapOS combines grid computing and Enterprise Resource Planning (ERP) to provide Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) through a simple, unified API that everyone can learn in a couple of minutes. SlapOS opens new perspectives for research in the area of resilience and security on the Cloud because it is based on similar concepts than Desktop Grids. Indeed, the data centers are located on the home PCs of volonteers.

The extremely large number of nodes implies that faults (for instance hardware component failures) should be handled at any layer of the software stack (middleware, programming libraries, algorithmic level). Distributed problems on a large number of volatile resources with asynchronous communications are difficult to handle; some of them have even been proved impossible.

In traditional, tightly-coupled parallel applications, the survivability of the application is dependent upon the survivability of *all the resources* used by the application. When a single node fails, the application fails; on the other hand, in Dektop Grid computing the failure of a single client has no impact on the survivability of the other clients. The main reason is that Desktop Grid applications are follow the Bag-of-Tasks pattern (independant tasks) that are duplicated in order to tolerate failures. But, what happens if a major disaster appears, for instance if floods devastate eastern Europe and a large zone becomes disconnected from the Internet. For example in 2011, whole of Armenia was disconnected from the Internet by a simple copper wire theft[2].

In this context, we propose in this article, algorithms and

[1]Check the list of available BOINC projects on the Web at http://boinc.berkeley.edu/projects.php

[2]`http://www.bloomberg.com/news/2011-04-06/georgian-detained-for-cutting-georgia-armenia-internet-cabl html`

techniques to evaluate the 'quality' of the virtual topology we use in Desktop Grid applications or in some flavors of Cloud applications. We compute metrics that define the quality according to usual definitions [9, 3, 8, 12]. Among the most useful properties, we compute the node and link connectivity and the fault diameters. Experiments are performed on large irregular graphs.

1.2 Motivations and paper's organization

Failures are inherent to large scale systems [17]. Let s be the number of successful executions and f be the number of failures. Let $n = s + f$ be the number of executions we start on a large scale system. Then the probability of failure is $\frac{f}{s+f}$. As the number of hardware components increases when using large scale systems, so does the probability of failure, for a same n. Researchers made the observation that to ensure an efficient and safe execution of applications on large scale systems, a scalable and fault tolerance framework ought to be used.

Consider now an example. Parallel applications are supported by a run-time environment, which is in charge of deploying, supporting and monitoring the application on remote computing resources. The basic services it provides to the application include deploying the processes on the available resources, enabling communications between processes, I/O forwarding and monitoring.

This run-time environment is a distributed overlay network spanning over the available computing resources that can route out-of-band messages (*i.e.,*signaling messages) for peer-to-peer or collective communications. These communications are not directly used by the parallel application, but they are used internally to support the application. Hence, the core of a run-time environment is a communication infrastructure used to communicate these control and signaling messages between nodes.

Scalable and fault tolerant framework [3] may concern the run-time environment or the applications themselves. Implementing a scalable and fault tolerant component in connection with the application is not a good idea because this leads to develop a component for each application. It is preferable to develop it as a component of the run-time environment, and if possible inside a run-time environment that supports the communication library, such as MPI (Message Passing Interface [6, 7], which is the *de facto* standard for programming parallel applications).

Hence, the run-time environment can be extended to feature fault tolerance capabilities. The overlay network itself must be able to recover its state from failures and maintain an acceptable quality of service during the recovery phase (self-healing property). It also needs to provide the application with some features that will allow it to recover from the failure and proceeds with an application-based fault tolerance mechanism.

Scalable and fault tolerant framework even for Desktop Grid applications may also concern the building of dedicated "control signaling networks" for collecting monitoring information (which node is alive? what is the energy consumption of these nodes? what is the load of node X?), for gathering simulation results, for saving checkpoint information.

Hence, the topology of the communication infrastructure of the overlay network is highly critical. It determines the efficiency of peer-to-peer communications, the scalability of the overlay network itself, and the resilience or self-healing properties it can have.

The aim of this paper is to propose efficient parallel algorithms and use cases for computing fault tolerant properties of irregular topologies, that cannot be analyzed by exact methods and modeling a communication and control infrastructure network, and not exclusively dedicated for massively parallel applications but also for Desktop Grid applications. For the engineer and regarding how much control or how many signaling processes he wants, some fault tolerance properties can be more or less relaxed. For instance, the diameter of the graph may be a more important criteria when we have to collect information about the temperature of a machine occasionally than when we want to implement a collective communication primitive or an efficient deployment framework on top of the Internet.

Thus, the contributions of this paper are new mathematical results concerning the fault tolerance properties of benchmarks that map to large irregular graphs and parallel algorithms for computing the relevant metrics.

The organization of this paper is as follows. Section 2 recalls the useful definitions. Section 3 gives details about the fault tolerance properties we are studying in this paper. In this section we also give refinements of algorithms initially introduced in [5] and we compare the approach of Magnien *et al*in [16] for computing the diameter with our approach. Section 4 presents experiments with large irregular graphs. Section 5 concludes the paper.

2. BINOMIAL GRAPHS FOR INTRODUCING THE PROBLEMS

2.1 Introduction

In [5] we presented the computation of fault tolerant properties of one class of regular graphs, namely, binomial graphs. Originally, the binomial graph logical structure was introduced in [4, 2] with the purpose of satisfying fault tolerant requirements for the communication layer of run-time environments. In general, scalable logical topologies ought to meet the following requirements: (a) low degree – the degree of a node is the number of links incident to the node; (b) regular graph – every node has the same degree such that a 'single' algorithm should be designed to route message for instance; (c) low diameter – the diameter of a graph is the longest shortest path between any two nodes and it gives a bound on the complexity of routing messages; (d) symmetric graph – the average inter-nodal distance should be the same from any source node; (e) no restriction in terms of numbers of nodes to support large scale application.

A BMG (Binomial Graph) is an undirected graph $G = (V, E)$ where V is a set of VErtices ; $|V| = n$ and E is a set of links (edges) ; $|E| = m$. Each node $i \in V$, and $i = 1, 2, \cdots n$ has links to a set of nodes U, where $U = i \pm 1, i \pm 2, \cdots, i \pm 2^k | 2^k \leq n$ in circular space *i.e.,*node i is

connected to its right and to its left to a node at distance 2^k for all $0 \leq k \leq \lfloor \log n \rfloor$.

In [5] we have introduced a variant of binomial graphs. Formally, a probabilistic BMG is a graph where we randomly select n vertices, then from each vertex, we graft links according to a logarithmic method. Intuitively, we try to keep as much as possible the good properties of BMG regarding its structure, whereas some of them are relaxed. In [5] we consider two methods for generating the graphs:

1. Model 0 is the original BMG;

2. Model 1: we graft exactly as with the original BMG. Node i, randomly selected, has links to a set of nodes U, where $U = i \pm 1, i \pm 2, \cdots, i \pm 2^k | 2^k \leq n$ in circular space. The fact is that since we do trials for the choice of the node, this kind of BMG has no more links than the original BMG; The pseudo code for generating this kind of BMG is as follows:

```
Distance *= 2;
while (Distance < NbVertices)
{
  for (int s = 0; s < NbVertices; s++)
  {
    int sp=(int)(((double) rand()/RAND_MAX)
           * NbVertices);
    AddArc(Adjacencies, sp, (sp + Distance)
           % NbVertices);
    AddArc(Adjacencies, sp, (sp - Distance
           + NbVertices) % NbVertices);
  }
  Distance *= 2;
}
```

3. Model 2: the process is as with Model 1, but we select less links when we have to graft links. The number of nodes is divided by two at each iteration. The pseudo code for generating this kind of BMG is as follows:

```
Distance *= 2;
NbV = NbVertices;
while (Distance < NbVertices)
{
  for (int s = 0; s < NbV; s++)
  {
    int sp=(int)(((double) rand()/RAND_MAX)
           * NbVertices);
    AddArc(Adjacencies, sp, (sp + Distance)
           % NbVertices);
    AddArc(Adjacencies, sp, (sp - Distance
           + NbVertices) % NbVertices);
  }
  Distance *= 2;
  NbV /= 2;
}
```

   With this process, we get less links than with Model 1;

Notice that to avoid to have more than one connected component in the graphs, we also connect, for processes 1 and 2, each node to its neighbor, the one at its right and the one at its left (node i is connected to $i-1$ and $i+1$).

2.2 Terminology about the structures of graphs

The minimum degree δ_{min} of a graph is the smallest node degree, while the maximum degree δ_{max} of a graph is the largest node degree. If every node has the same degree ($\delta_{min} = \delta_{max}$) the graph is *regular*. A regular graph also means that all the nodes are equivalent and for each node, we can use symmetrically the same routing and fault-handling algorithms instead of managing special cases. However, notice that the graph of the Internet is obviously not a regular one. The original BMG graph is a regular graph but not the graphs produced with models 1 and 2.

One important question is: how can we compute some properties of graphs with a large number of nodes and vertices? One interesting property is the diameter, which gives a bound on the number of steps required to route a message between two nodes. Computing the diameter of a graph is time-consuming, as we explain in the next section which is devoted to the setting of properties of our graphs.

3. FAULT TOLERANT PROPERTIES AND NEW MATHEMATICAL RESULTS

3.1 Preliminary remark

A mathematical computation of properties describing the graph, for instance the diameter, the shortest path length computation may be difficult for the following reason: the stochastic model we have used in model 1 and 2 implies that many parameters of the graph (average degree and others structural properties) have no closed form that could be estimated easily, in average and in the worst case. This explains why we have introduced in [5] heuristics for the various computations. Some of them are bounded (with lower and upper bounds).

3.2 Definition of the properties

The *cost* of a particular topology [15] can be defined as the product of the diameter and the number of links.

The average distance $\bar{d}$ of graph is given by Equation 1:

$$\bar{d} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} d(i,j)}{n * (n-1)} \tag{1}$$

To bypass the n^2 calls to the distance computation, we perform $N = \lfloor n * 0.05 \rfloor$ random trials of pairs of vertices, we compute the distance over the two vertices, then we sum the distances and we divide by N.

We now explain the computation of specific metrics related to fault tolerance, such as connectivity and fault diameter of large scale graphs.

The *node connectivity* κ of a graph is defined as the minimum number of nodes of which removal can result in a graph with at least two connected components.

The node connectivity metric can be computed according to the following optimizations: when we remove a vertex we have also to adjust the adjacency lists of the nodes that are present in the adjacency list of the node we remove. If we

maintain an ordered list of vertices and another ordered list for the adjacency lists, we can accomplish the work according to a logarithmic factor in the size of the adjacency lists and also in $\log(n)$. Moreover, instead of renumbering the whole graph after removing a vertex, which would require an $\mathcal{O}(n^2)$ time complexity, it is sufficient that the nodes be marked as not present. This marking is done in constant time.

As you can notice, many programming optimizations are possible and they are dependent of the initial choice of the graph's representation. We do not give more details here and we invite the reader to examine the codes[3] corresponding to the metrics.

The *link connectivity* λ of a graph is defined as the minimal number of links of which removal can result in disconnecting the network. The algorithm presented in [5] is introduced as algorithm 1 in this paper.

Algorithm 1 Link connectivity as published in [5]

```
1: for i = 1 to δ_min do
2:    for NumChoice = 1 to MaxChoiceLinks do
3:       choose i links;
4:       remove the links;
5:       if graph is not connected then
6:          return i
7:       end if
8:    end for
9: end for
```

Note that in Algorithm 1, we can stop the iteration at δ_{min} which is low in practice with BMG like graphs.

The *fault diameter* F is the largest diameter of the network when there are $\kappa - 1$ node failures (a maximum number of failure nodes before the network becomes bipartite, κ being the node connectivity).

Algorithm 2 Fault diameter as published in [5]

```
1: for NumChoice = 1 to MaxChoiceDiameter do
2:    choose κ − 1 vertices;
3:    remove the vertices and compute the diameter;
4:    keep the maximal value for diameters;
5: end for
```

Note that Algorithm 2 does not implies exhaustive searches, so it is not an exact method for computing the associated metric. In general, the exact time for executing the computation depends on a parameter describing the graph (for instance the minimal degree in the Link connectivity algorithms). In practice for BMGs, since the degree is small comparing to n, we may drastically decrease the execution time of the computation comparing to a full enumeration of combinations over all the vertices or links. Algorithms provide an estimate of the associated metric: it is an heuristic method, bounded by δ_{min} for instance.

[3]The current release of our implementation is available on `http://www.lipn.fr/~cerin/code.tar.bz2`

Here again, it seems difficult to have a closed formula for the different complexities. However, the exact computation of the node connectivity metric for the initial algorithm can be bounded by the sum of binomial coefficient computation (because we have to compute n times the choice of i vertices among n in all possible ways) times the time complexity of detecting if we have at least 2 connected components. The complexity is exponential and we cannot expect to solve the problem by exhaustive searches, hence our heuristics that do sampling over all the combinations are more realistic.

The last property we encounter in the papers related to the domain is about resilience. There are two classes of graph G distinguished by the relationship between the fault diameter F and the diameter D of the graph [13], called *strongly resilient* and *weakly resilient*. A graph is considered strongly resilient if there exists a constant ϕ such as $F(G) \leq D(G)+\phi$ for all graph sizes n, where $n \in \mathbb{N}$. A graph is considered weakly resilient if there exists a constant ϕ such as $F(G) \leq D(G) \times \phi$ for all graph sizes n, where $n \in \mathbb{N}$. The original BMG is considered as strongly resilient by Angskun, Bosilca and Dongarra [2] with $\phi = 2$. Note that the proof is made with experiments and it is not a formal one. Under the experimental conditions of the paper, the result indicates that, even under faulty conditions, the performance of BMG will not be severely degraded.

3.3 Computational issues and related work for diameter computation

In this section, we explain one computational issue and solutions. The distance $d(i, j)$ between a node i and a node j in a graph is defined as the length of the shortest path from i to j in the graph. The diameter D of a graph is given by $\max\{d(i, j)\}$ over all possible pairs (i, j) of nodes in the graph. The diameter D is the longest shortest path between any two nodes in the graph.

It is not difficult to see that in the case of BMG, which is a regular graph, the diameter is in $\mathcal{O}(\log n)$, and as pointed out in [4, 2], the BMG has the lowest diameter among Hypercube [15], Chord [21], 4-ary Hypercube topologies [18, 8]. This property explains why we have studied, initially, the BMG.

But computing all distances from one vertex to all the others has $\Theta(m)$ time and space costs using a breadth-first search (BFS). In order to compute the diameter, one has to compute the distance between all pairs of vertices, which therefore involve a $\Theta(nm)$ time and a $\Theta(m)$ space cost using a BFS. Using matrix products, one may achieve the computation in $\mathcal{O}(n^{2.376}\text{polylog } n)$ time and $\mathcal{O}(n^2)$ space [1, 19]. Therefore, BFS approach is too slow for graphs with a large number of vertices, and matrix approach has also a supplementary and prohibitive space cost.

Different solutions have been proposed in a recent past as well as implementations to solve this problem. We can mention [11, 10] but we do prefer to introduce the work and the implementation [16] done by Magnien, Latapy and Habib. The key points in the methods used by Magnien is to compute lower and upper bounds and to iterate them from different initial vertices in order to obtain tighter bounds, with a linear cost for each step. Authors also use heuristics to

choose in an intelligent way the vertices able to provide with a tight bound.

Notice that to our knowledge, the metrics, in particular the computation of the metrics related to fault tolerance has not been investigated in the context of large scale non-regular graphs.

3.4 New approaches and results for computing the fault tolerant metrics

In this section, we first improve the algorithms for Node Connectivity and Link Connectivity according to a dichotomous approach as follows then we focus on the diameter computation.

At the beginning of the Node and Link Connectivity algorithms, we remove $n/2$ vertices. If the graph is not connected, then we check the connectivity property between 1 and $n/2$ vertices. If the graph is still connected, we check the property on the interval $n/2$ and n vertices. As with any dichotomous approach, we stop the process as soon as the 'lower bound' crosses the 'upper bound'. The corresponding algorithm is Algorithm 3. In doing this we limit the use of arbitrary constants to bound the iterations number.

Notice that with Algorithm 3, we can replace the word 'vertices' by 'edges' when we have to remove objects in order to get the link connectivity. Technically speaking, we have also to replace the call for replacing a vertice by the call for replacing an edge.

Algorithm 3 New node connectivity algorithm

Require: Low = 1 ; Up = MAX.
Require: (MAX = NbVertices or MinDegree).
set $K = \log n$
while $Up - Low > 1$ **do**
  m = (Low + Up) / 2
  repeat
    Remove m vertices (or edges) randomly
    if graph not connected **then**
      Up = m
    end if
  until K times
  if all graphs are connected **then**
    Low = m
  end if
end while
return Low + 1

Notice that for the computation of Link and Node Connectivity we have to check if the graph is connected. We have implemented this operation as follows. The key ideas of the Algorithm 4 and the forthcoming algorithms are in the intensive use of the BFS (Breadth First Search) algorithm slightly modified.

The idea is to start from a vertex and to consider its neighbors and their neighbors and so on. But the seen vertices are pruned of the list of vertices that we have to visit. So we stop the process when there is no new neighbor. The new algorithm for computing the connected components of a graph is the Algorithm 4.

Algorithm 4 New implementation for checking if a graph is connected

Start with the first non visited vertex
Visit its connected component according to the previous technique
Restart until there is no more vertice to visit
(that is to say that all vertices have been visited)
The number of times we do step 3 = number of connected components

Starting with this idea, we have also implemented the IAmconnected() procedure for checking if a graph is connected. Here again, the idea is to start from a (random) vertex and we visit its connected component according to the previous technique. If all the vertices have been visited, we return True, otherwise we return False.

We have also revisited the computation of the diameter done by Magnien in [16] according to the following algorithm (see Algorithm 5).

Algorithm 5 New algorithm for the diameter

set $Diameter = 0$
repeat
  Select randomly a vertex, name it 'current vertex'
  and mark it as visited. Set 'Current diameter' to 0
  while current vertices have non visited neighbors **do**
    a) Compute the non visited neighbors of current vertices
    b) Replace the current vertices by their non visited vertices
    c) Add 1 to 'Current diameter'
  end while
  if 'Current diameter' > 'Diameter' **then**
    'Diameter' = 'Current diameter'
  end if
until 'some' vertices have been visited
return 'Diameter'

Here again, the tricky part is to accelerate the computation in visiting the neighbors of the neighbors that have not yet been visited. It is a strategy similar to the 'doubling strategy' for the PRAM (Parallel Random Access Memory) paradigm.

The diameter computation is used in the fault diameter computation that is not modified since our initial approach in [5]. See Algorithm 2 for the fault diameter computation.

4. EXPERIMENTAL RESULTS

We used the data set available online[4] and provided by Clémence Magnien and Matthieu Latapy. The online data set is composed of 4 files that lead to a wide variety of real-world graphs coming from different contexts. It may be considered as representative of the variety of cases that we found in complex network studies [14].

We use the following three benchmark files:

[4]See: http://data.complexnetworks.fr/Diameter/

- An Internet topology graph (inet) obtained from traceroutes ran daily in 2005 by Skitter[5] from several scattered sources to almost one million destinations, leading to 1,719,037 vertices and 11,095,298 edges; The number of connected components is 23,378 and the largest one has 1,694,616 vertices and 11,092,661 edges. We compute the fault tolerant properties for that component;
- A peer-to-peer graph (p2p) in which two peers are linked if one of them provided a file to the other in a measurement conducted on a large eDonkey server for a period of 47 hours in 2004[6], leading to 5 792,297 vertices and 142,038,401 edges; The number of connected components is 411,757 and the largest one has 5,380,491 vertices and 142,038,351 edges. We compute the fault tolerant properties for that component;
- A traffic graph (ip) obtained from MetroSec[7] that captured each ip packet header routed by a given router during 24 hours, two ip addresses being linked if they appear in a packet as sender and destination, leading to 2,250,498 vertices and 19,394,216 edges; The number of connected components is 45 and the largest one has 2,250,046 vertices and 19,393,724 edges. We compute the fault tolerance properties for that component;

Table 1 shows the results for the diameter estimation in comparing our estimate with the estimates of Magnien and Latapy in [16]. Since Magnien and Latapy provide five bounds, we give them explicitly in Table 1. Lowest values are for lower bounds, highest values are for upper bounds on the diameter. We found that in any case, our estimates fit between the lower and upper bounds of Magnien because we try with only 20 vertices.

The originality of the Magnien's approach lies in the fact that there is a guarantee that the actual diameter is within the bounds they find, but there is no guarantee on the tightness of these bounds. The tricky part of Magnien's implementation is to start the estimate with vertices with low degrees because in that way, the underlying BFS tree has a much greater height than if we start with vertices with high degree. In another words, we have more chance to extend the distance between vertices. We have not yet implemented this heuristics. In our case, we take a random vertice (potentially, it may have a high degree), and this could explain the difference in the result for the diameter computation of the inet benchmark.

Table 1: Comparison between estimated diameters

	Magnien and Latapy results [16] tlb - dslb - hdtub - rtub - tub	Our results
inet	29-31-34-34-38	25
p2p	8-9-10-10-10	8
ip	9-9-9-9-10	8

Table 2 presents our estimates on the major fault tolerance metrics, namely the link and node connectivity and the fault diameter. As expected, the link and node connectivity are low, showing that the graphs are not resilient. The estimated fault diameters are also estimated to the values of the diameters which is also expected. It is because we have to check the connectivity of graphs when we remove no more than two vertices among million of vertices, and this process has a low probability to disconnect the graphs. However, it is important to notice that our algorithms run in few hours for computing **all** the parameters of a single graph on a 'normal computing platform' based on a Xeon processor with 96GB of memory (16GB is enough in our cases). The current version of our code is available upon request.

Table 2: Metrics for Fault Tolerance

	Link co.	Node co.	Fault diameter
inet	2	2	24
p2p	2	2	8
ip	2	2	8

At least, note that it is the first time, to our knowledge, that the metrics presented in this paper are computed for Internet graphs. As a consequence we do not provide a broad comparison in terms of execution times. We are only able to show some results about the diameter in terms of returned bounds provided by our method versus Magnien's method.

5. CONCLUSION

This paper presents estimators for two very useful graph metrics: diameter and connectivity. Then, we use these metrics in conjunction with previous research related to fault tolerance properties of different types of graphs. With these estimators, we compare the estimators against other published results using a publicly available suite of sample graphs, which represent various network topologies.

The algorithms described in this paper are for large, and irregular graphs, and are parallel probabilistic algorithms. Experimental results presented in Section 4 demonstrate that the algorithms work and compare favorably to previous approaches that required higher computational complexity.

It is the first time, to the best of our knowledge, that fault tolerant properties are exhibited for the benchmarks that map to large scale graphs. In particular we conducted the experiments on real life graphs extracted from the Internet.

The initial objective of this work is to evaluate the suitability of any regular topologies to be used as a basis for a resilient, scalable communication infrastructure. Examples of such infrastructure are the run-time environment of a parallel, distributed system on top of the Internet or any controlling network (for monitoring loads, temperature of CPUs). In this paper, we demonstrate that our implementation is also good for working on irregular structures and large scale graphs.

Based on our experiments, we conclude that our methods and algorithms are operational for estimating the metrics of fault tolerant overlay networks. One issue would be to compute them in a distributing way, we mean with local interaction, only.

[5] `http://www.caida.org/tools/measurement/skitter/`
[6] `http://www-rp.lip6.fr/~latapy/P2P_data/`
[7] `http://www2.laas.fr/METROSEC/`

The core objective of this work is related to Internet computing. In the future we will focus on a holistic approach to explore and to understand the 'behaviors' of complex systems such as clouds or large scale grids in case of disasters. The work will focus on one aspect of safety (the condition of being protected against physical, social, financial, political, or consequences of failure, damage, error, accidents) in Cloud industry. Safety is part of the security area. The scenario we envision is the following one: floods devastate eastern Europe. Internet is no longer available in Dresden area. Data centers are destroyed.

Some rescue missions are sent with tablet PCs or Net PCs. How can we restore access to the Cloud despite such disaster? How to measure the quality of the network being reconstructed? The issue is about resilience which is the ability of a system or network architecture to continue to operate in case of failure and the computational methods of this article will serve to estimate and to drive the rebuilding of a new topology.

6. REFERENCES

[1] Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. Witnesses for boolean matrix multiplication and for shortest paths. In *FOCS*, pages 417–426. IEEE, 1992.

[2] Thara Angskun, George Bosilca, and Jack Dongarra. Binomial graph: A scalable and fault-tolerant logical network topology. In Ivan Stojmenovic, Ruppa K. Thulasiram, Laurence Tianruo Yang, Weijia Jia, Minyi Guo, and Rodrigo Fernandes de Mello, editors, *ISPA*, volume 4742 of *Lecture Notes in Computer Science*, pages 471–482. Springer, 2007.

[3] Thara Angskun, Graham E. Fagg, George Bosilca, Jelena Pjesivac-Grbovic, and Jack Dongarra. Scalable fault tolerant protocol for parallel runtime environments. In Bernd Mohr, Jesper Larsson Träff, Joachim Worringen, and Jack Dongarra, editors, *PVM/MPI*, volume 4192 of *Lecture Notes in Computer Science*, pages 141–149. Springer, 2006.

[4] Thara Angskun, Graham E. Fagg, George Bosilca, Jelena Pjesivac-Grbovic, and Jack Dongarra. Self-healing network for scalable fault-tolerant runtime environments. *Future Generation Comp. Syst.*, 26(3):479–485, 2010.

[5] Christophe Cérin, Michel Koskas, and Yu Lei. Computing properties of large scalable and fault-tolerant logical networks. In Alberto F. De Souza and Lucia Catabriga, editors, *IEEE SBAC*. IEEE, 2011.

[6] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, Department of Computer Science, University of Tennessee, April 1994. Tue, 22 May 101 17:44:55 GMT.

[7] Al Geist, William D. Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing L. Lusk, William Saphir, Anthony Skjellum, and Marc Snir. MPI-2: Extending the message-passing interface. In Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *1st European Conference on Parallel and Distributed Computing (EuroPar'96)*, volume 1123 of *Lecture Notes in Computer Science*, pages 128–135. Springer, 1996.

[8] A. Ghafoor and T.R. Bashkow. A study of odd graphs as fault-tolerant interconnection networks. *Computers, IEEE Transactions on*, 40(2):225 –232, feb 1991.

[9] Bernd Mohr Jesus Labarta, Barton P. Miller and Martin Schulz. Program development for extreme-scale computing. Technical report, Dagstuhl Seminar 10181, http://www.dagstuhl.de/10181, 2010.

[10] U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations. In *SDM*, pages 548–558. SIAM, 2010.

[11] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. Pegasus: mining peta-scale graphs. *Knowl. Inf. Syst.*, 27(2):303–325, 2011.

[12] Jong-Seok Kim and Hyeong-Ok Lee. Comments on "a study of odd graphs as fault-tolerant interconnection networks". *Computers, IEEE Transactions on*, 57(6):864, june 2008.

[13] M. S. Krishnamoorthy and B. Krishnamurthy. Fault diameter of interconnection networks. *Computers & Mathematics with Applications*, 13(5-6):577 – 582, 1987.

[14] Matthieu Latapy and Clémence Magnien. Complex network measurements: Estimating the relevance of observed properties. In *INFOCOM*, pages 1660–1668. IEEE, 2008.

[15] Ahmed Louri, Brent Weech, and Costas Neocleous. A spanning multichannel linked hypercube: A gradually scalable optical interconnection network for massively parallel computing. *IEEE Trans. Parallel Distrib. Syst.*, 9(5):497–512, 1998.

[16] Clémence Magnien, Matthieu Latapy, and Michel Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[17] Daniel A. Reed, Charng da Lu, and Celso L. Mendes. Reliability challenges in large systems. *Future Generation Computer Systems*, 22(3):293 – 302, 2006.

[18] Y. Saad and M.H. Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, 37:867–872, 1988.

[19] Raimund Seidel. On the all-pairs-shortest-path problem. In *STOC*, pages 745–749. ACM, 1992.

[20] Jean-Paul Smets-Solanes, Christophe Cérin, and Romain Courteaud. Slapos: A multi-purpose distributed cloud operating system based on an erp billing model. In Hans-Arno Jacobsen, Yang Wang, and Patrick Hung, editors, *IEEE SCC*, pages 765–766. IEEE, 2011.

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