Finding Stable Cliques of PlanetLab Nodes

Elias P. Duarte Jr., Thiago Garrett, Luis C. E. Bona, Renato Carmo, and Alexandre P. Züge
Federal University of Paraná - Dept. Informatics - P.O. Box 19018 Curitiba PR 81531-980 Brazil
{elias,garrett,bona,renato,alexandrep}@inf.ufpr.br

Abstract

Users of large scale network testbeds often execute experiments that require a set of nodes that behave and communicate among themselves in a reasonably stable pattern. In this work we call such a set of nodes a stable clique, and introduce a monitoring strategy that allows their detection in PlanetLab, a non-trivial task for such a large scale dynamic network. Nodes monitor each other by sampling the RTT (Round-Trip-Time) and computing its variation. Based on this data and a threshold, pairs of nodes are classified as stable or unstable. A set of graphs is generated, on which maximum sized cliques are computed. Three experiments were conducted in which hundreds of nodes were monitored for several days. Results show the unexpected behavior of some nodes, and the size of the maximum stable clique for different time windows and different thresholds.

1. Introduction

As new alternatives for the Internet architecture are proposed, large scale realistic testbeds become increasingly important [14]. These testbeds are heterogeneous wide-area networks in which protocols, distributed applications and services can be deployed and evaluated on supposedly real conditions. PlanetLab [4] is one of such global research networks that supports the development of new protocols and services. PlanetLab is arguably the largest and most important of these wide-area research testbeds. At the time this work was done, PlanetLab consisted of 1060 nodes at about 491 sites, located all over the world. Nodes are TCP/IP hosts connected among themselves through the Internet. Each node is kept by an autonomous organization that is affiliated to the PlanetLab. Different nodes have widely different capabilities and are connected to networks which are configured and managed in various ways, which results in an environment of great instability.

Researchers need a real environment, subject to real conditions, such as occasional loss of connectivity and congestion, in order to evaluate their proposals. Nevertheless, depending on the level of instability it may even become impossible to run an application that involves node communication. In order to execute a protocol or a distributed application it is frequently necessary to have a set of nodes that present a minimum level of stability. This was exactly the case when we executed HyperBone [3] in PlanetLab. HyperBone is a an overlay network that allows the execution of distributed applications on a virtual hypercube.

In order to execute parallel and distributed tasks, HyperBone requires a set of nodes that present a reasonably stable behavior. We found out that it is not trivial to find such a large set of such nodes in PlanetLab. Sometimes it is not easy even to find a set of nodes each of which can communicate with all others. At a given time, a large set of such nodes might not even exist. Another characteristic we found out is that a communication channel is frequently not symmetric: if a node considers another to be stable, the opposite might not be true. Moreover, a given node might consider two other nodes to be stable, but those two nodes may not consider each other stable. Several communication patterns were observed.

We thus developed a monitoring strategy to find a set of reasonably stable nodes in PlanetLab, on which we could execute our experiments. We call such a set of nodes a *stable clique*: if PlanetLab is represented as graph $G = (V, E)$, a clique [6] is a complete subgraph of G in which all edges correspond to communication channels classified as stable. These cliques can be seen as a stable portion of an unstable network.

In order to find a stable clique, nodes continuously monitor each other. A node samples the RTT (Round-Trip-Time) and computes the variation of the perceived RTT to every other node. Based on this data and a threshold, pairs of nodes are classified as stable or unstable. A set of graphs is generated, on which maximum sized cliques are computed.

In this work we describe three experiments in which from 200 to 461 PlanetLab nodes were monitored for several days. A monitoring daemon was run on all nodes in which we could set up the monitoring environment and

were not turned off for the whole experiment time. Each node monitors all others. We present experimental results, describing the unexpected behavior of some nodes, and show the size of the maximum stable clique for different time windows and different stability thresholds.

Related work includes other PlanetLab monitoring tools, such as CoMon [16] and Ganglia [13] which measure the state/load of nodes and slices by themselves – not their interaction, as we do. PlanetFlow2 [8] is a tool for PlanetLab traffic monitoring. netEmbed [12] employs heuristic algorithms for grouping and selecting nodes but requires an external monitoring system. Vivaldi [5] computes the RTT among nodes. MON [11] selects fault-free nodes for executing an experiment and monitors its execution. SWORD [1] runs as a PlanetLab service for selecting nodes for running experiments based on monitoring data obtained from Ganglia and Vivaldi. None of these tools employ historical monitoring data to determine cliques of nodes that present a stable communication pattern.

The rest of this paper is organized as follows. Section 2 defines the proposed monitoring strategy. Section 3 describes the algorithm employed for finding stable cliques. Experimental results are given in section 4. The conclusion follows on section 5.

2. Monitoring Strategy

In this section we describe the PlanetLab monitoring strategy. Each node executed a monitoring daemon, which periodically sent a query to all other nodes. As a reply arrived, the node computed and recorded the Round-Trip-Time (RTT) and the RTT variation, using a approach based on van Jacobson's TCP *TimeOut* (TO) interval [9]. Each experiment lasted from 1 to 2 weeks. After the conclusion, we downloaded the data recorded by all nodes. We used this data to model the system as a set of undirected graphs. A graph $G_t = (V, E_t)$ was computed for time instant t, where V is the set of nodes which ran the experiment and E_t the set of *stable* edges that were present at time t. An edge between two nodes represents the fact that they can communicate with each other. Thus for an edge to be included in the graph, the communication test must have succeeded in both ways. We found several instances in which a given node i could communicate with j but j could not communicate with i. In these cases edge $(i, j) \notin E_t$.

Figure 1 shows the perceived RTT variation of a node classified as unstable by most nodes during the whole monitoring period. These particular RTT samples were obtained from the point of view of the node that presented the highest degree in all graphs generated in experiment 1.

After graph G_t is built, we run an algorithm for finding

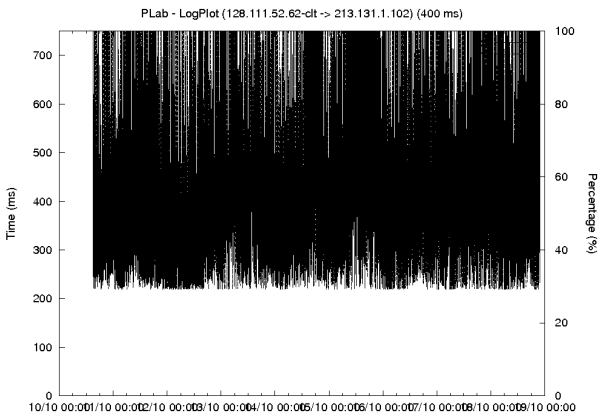

Figure 1. RTT variation of an unstable node

what we call a *stable clique* on G_t, i.e., a subgraph of G_t in which there is an edge from every node to every other node. In the experiments, a graph was generated every 15 minutes. As mentioned above, in order to determine whether a given pair of nodes presents a stable communication pattern, we considered the RTT variation as the parameter of choice. The strategy used to classify the node communication as stable or not employs van Jacobson's TO, which heavily relies on the observed RTT variation. Besides the TO itself, our classification employs an adjustable threshold value which is computed empirically. If a function of the TO of a given pair of nodes is below the threshold, then the pair of nodes is classified as unstable. Otherwise it is classified as stable. Note that as time passes the classification of a specific pair of nodes may change from stable to unstable and vice versa. We evaluated clique sizes for several thresholds.

The TO is updated for each RTT sample i. Let TO_i, be the weighted mean of previously computed TO values and the current RTT sample. This mean acts as a statistical filter to remove noise from the TO curve, making it easier to find the TO valleys, described below. The TO is computed with the following expression. In this expression $\Delta RTT_i)$ is the weighted mean of the RTT samples. $|\Delta(RTT_i) - RTT|$ corresponds to the difference of the last RTT sample and the weighted mean. In the experiments we used $\alpha = 0.9$ and $\beta = 4$.

$$TO_i = \alpha * TO_{i-1} + (1-\alpha) * (\Delta(RTT_i) + \beta * |\Delta(RTT_i) - RTT|)$$

It is important to compute a "fair" threshold which allows nodes to be classified as stable/unstable. Considering the TO curve, an example of which is shown in figure 2, it often presents a series of peaks and valleys. A valley corresponds to lower values of the TO, and the variation of RTT is also low. A peak corresponds to periods in which

there is a higher variation of consecutive samples of the RTT. The threshold is determined by observing the variation of the RTT and the TO curves. Initially the curve computed for a pair of nodes is smoothed with a statistical filter. The communication between the pair of nodes is considered to be stable during the periods in which the valleys of the smoothed curve are below the threshold.

The example in figure 2 shows the use of a 400ms threshold to determine the stability of a node. This TO curve was computed for a node that was monitored for 4 hours and 30 minutes in experiment 1 (details are given in section 4). The little circles show the valleys of the TO. Until 03:30 of October 15th, the RTT presented a high variation, and the TO valleys are also high. The RTT variation then reduces, and so do the TO valleys. In the period in which the TO valleys are mostly above the threshold, the node is classified as BAD (*unstable*). Otherwise, when the TO valleys are below the threshold, the node is classified as GOOD (*stable*). The graph also shows that our classification criterion does not take into account brief variations of the TO, which could lead to a misclassification.

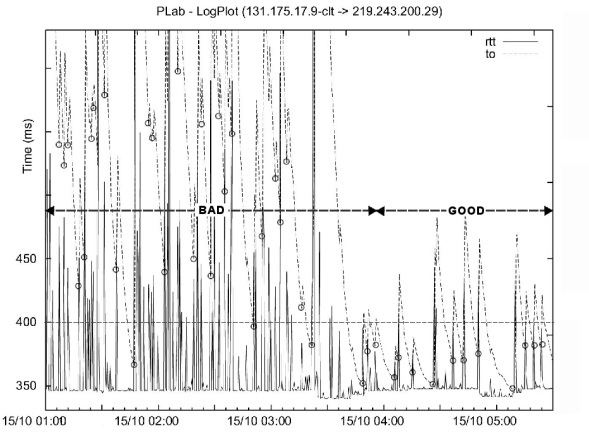

Figure 2. A threshold is used to classify nodes as stable

Based on this data the algorithm described in the next section is employed for computing the stable cliques.

3. Computing the Node Cliques

As is well known, the problem of computing a maximum clique (MC, for short) in an arbitrary graph is $\mathcal{NP}$-hard. Indeed, the corresponding decision problem ("given a graph G and an integer k, does G has a clique of size k?") is one of the 21 problems in [10], as well as one of the "six basic $\mathcal{NP}$–complete problems" chosen as "the 'basic core' of $\mathcal{NP}$–complete problems for the beginner" in [7].

There are several algorithms for the exact solution of MC (see, for example, [18] and [15], or [2] for a survey on the subject). Reported experimental results show that many instances of practical interest of the problem can be solved with reasonable computational resources. Fortunately, this is the case of the graphs generated by the experiment at hand.

Some of the most successful approaches in solving practical instances of MC are Branch & Bound based ones which can be described as follows. Let G be an undirected graph and let K be a clique in G. Consider the set N_K given by the intersection of the neighborhoods of the vertices in G, that is, $N_K = \bigcap_{v \in K} \Gamma(v)$. Note that K is a maximal clique in G if and only if $N = \emptyset$. Otherwise, for every $u \in N_K$, the set $K \cup \{u\}$ is a clique in G and $N_{K \cup \{u\}} = N_K \cap \Gamma(u)$.

The following (schematic) algorithm for finding a maximum clique in a given graph G is based on the remarks above. The algorithm works keeps a clique C in G and a list S of pairs (K, N_K). Initially, C is empty and S contains only the pair $(\emptyset, V(G))$. At each step, the algorithm removes a pair (K, N_K) from S. If N_k is empty, then K is a maximal clique in G. If $|K| > |C|$, the algorithm lets $C \leftarrow K$. If N_k is not empty, the algorithm computes an upper bound b on the size of the maximum clique in $G[K \cup N_K]$. If $b \leq |C|$, the pair (K, N_K) is discarded; otherwise, a vertex v is chosen from N_k and the pairs $(K \cup \{v\}, K \cap \Gamma(v))$ and $(K, N_K - \{v\})$ are added to S.

MaximumClique(G)

```
C ← ∅
S ← push(∅, V(G))
while S ≠ ∅ do
    (K, N) ← pop(S)
    if N = ∅ then
        if |K| > |C| then
            C ← K
    else
        if |K| + Bound(G, N) > |C| then
            v ← pop(N)
            S ← push(K, N)
            S ← push(K ∪ {v}, N ∩ Γ(v))
return C
```

In the algorithm, $S \leftarrow \mathsf{push}(e)$ denotes the operation of adding the element e to set S. Likewise, $e \leftarrow \mathsf{pop}(S)$ denotes the operation of removing some element from set S and storing this element in e. $\mathsf{Bound}(G, N)$ returns an upper bound on the size of the maximum clique in $G[N]$. Using the schematic algorithm $\mathsf{MaximumClique}(G)$ above as a reference, different concrete algorithms for MC result from choosing the data structures implementing sets S, K and N (and their respective insertion/deletion policies), and

the bounding function Bound(G, N).

For the determination of the cliques in the graphs presented in this work, we have implemented[1] an algorithm along the lines of the one described in [18]. In our implementation, set S is implemented as a stack and the sets K and N as balanced search trees. The function Bound(G, N) computes a (not necessarily minimal) coloring of $G[N]$ and returns the number of colors used in this coloring.

4. PlanetLab Experiments

PlanetLab is a global research network that supports the development of new network services [4]. As of December 2009, PlanetLab consisted of about 1060 nodes located at 491 sites all over the world, connected to each other through the Internet. Nodes have widely different capabilities and are connected to networks which are configured and managed in various ways, which results in an environment of great instability.

Three experiments in which the monitoring scheme described in section 2 were executed, we refer to these as experiments 1, 2 and 3.

Experiment 1 was started at October 2008, lasted 7 days, from October 11th 2008, 00:00:00 (GMT -3) until October 18th 2008, 00:00:00 (GMT -3) and involved 519 nodes, of which only 200 are considered here due to the reasons explained in section 2; experiment 2 was started at July 2009, lasted 8 days, from July 8th 2009, 00:00:00 (GMT) until July 16th 2009, 00:00:00 (GMT) and involved 631, of which only 400 are considered; experiment 3 was started at October 2008, lasted 12 days, from October 18th 2009, 00:00:00 (GMT) until October 30th 2009, 00:00:00 (GMT) and involved 638 nodes, of which only 461 are considered. In each of the 3 experiments, the time interval between snapshots was 15 minutes. Therefore, experiment 1 comprises $7 \times 24 \times 4 = 672$ snapshots, experiment 2 comprises $8 \times 24 \times 4 = 768$ snapshots, experiment 3 comprises $12 \times 24 \times 4 = 1152$ snapshots.

As described in section 2, a threshold is employed in order to classify a node as stable or unstable from the point of view of another node, given the monitoring data. This allows the computation of characteristics such as asymmetric views, in which a node is considered to be stable by another, but the opposite is not true. Figure 3 shows the percentage of asymmetric views obtained in experiment 1, for different values of the threshold, considering all node pairs that were monitored.

[1]The implementation was coded in C++, using the Boost Graph Library [17]. The resulting code was executed in set of Debian/GNU Linux systems using different hardware platforms available at C3SL (`http://www.c3sl.ufpr.br`)

After the stability is computed for all pairs of nodes, an undirected graph is built corresponding to a snapshot. In this graph, the vertices are the nodes themselves and there is an edge adjacent to nodes u and v if and only if both nodes classify each other as stable, i.e. they do not present asymmetric views.

We studied the behavior of the system for different values of the threshold. For experiment 1, threshold values of 400ms, 600ms, 1000ms and 2000ms were used; for experiments 2 and 3 threshold values of 200ms, 400ms, 600ms were used. In experiment 1, $672 \times 4 = 2688$ graphs were built, in experiment 2, $768 \times 3 = 2304$ graphs were built, and experiment 3 generated $1152 \times 3 = 3456$ graphs. In total 8448 graphs were built. These are the graphs on which we compute the maximum cliques using the algorithm described in section 3.

Figures 4, 5 and 6 show the size of the maximum clique of each graph in experiments 1, 2 and 3. As expected, the maximum clique size increases as the threshold increases. It should be noted however, that as the threshold increases, the distinction between stable and "not so stable" becomes blurred, as several communication patterns fall within the allowed level of stability. Indeed, when a higher threshold is employed several pairs of nodes presenting different levels of RTT variations are classified as stable; while a lower threshold would set them apart.

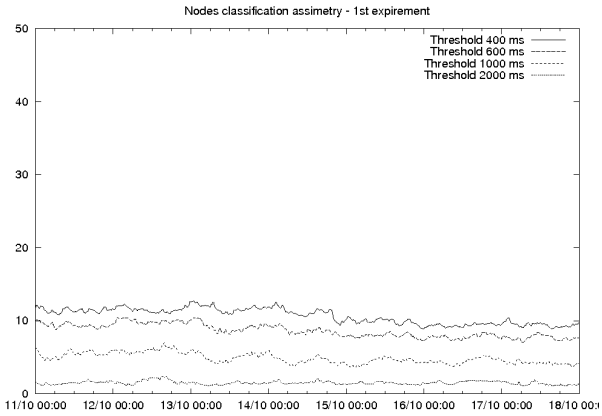

Figure 3. Node classification asymmetry

Another interesting result was obtained when the the maximum clique was computed from the intersection of all the graphs of a given experiment with a given threshold. This clique corresponds to a group of nodes that remained as a clique along the whole experiment, i.e., each node in the clique classifies each other as stable in all graphs. Table 1 shows the maximum clique sizes for each experiment and threshold.

Yet another interesting result is the size of the maxi-

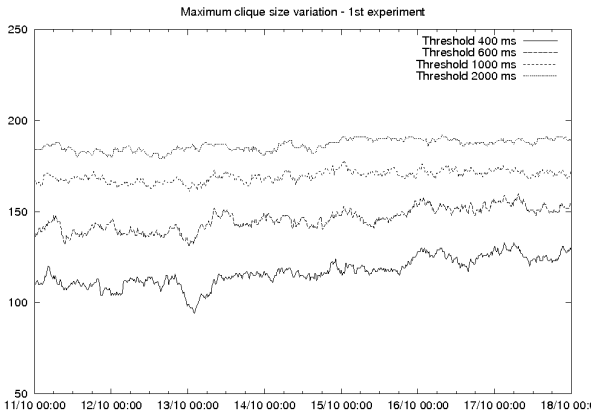

Figure 4. Maximum clique size variation for experiment 1

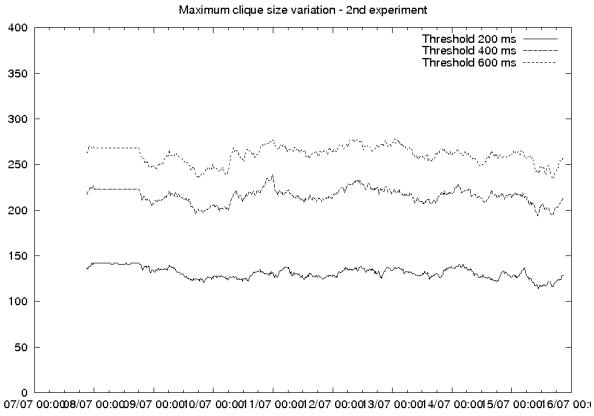

Figure 5. Maximum clique size variation for experiment 2

mum clique in the graph resulting from the intersection of consecutive graphs of each experiment. Obtaing this result was motivated by the fact that some distributed applications need very stable nodes but run for time intervals which are much shorter than the length of our experiments. In such cases, the knowledge of the largest group of nodes that forms a clique for a short time interval is the information required.

For each of the experiments and thresholds, we computed the maximum clique in the graphs that were built during one day and one hour. The results are shown in table 2 and 3, respectively. Table 2 shows the average maximum clique size computed every day for each experiment and threshold. Table 3 shows the average maximum clique size computed every hour for each experiment and threshold.

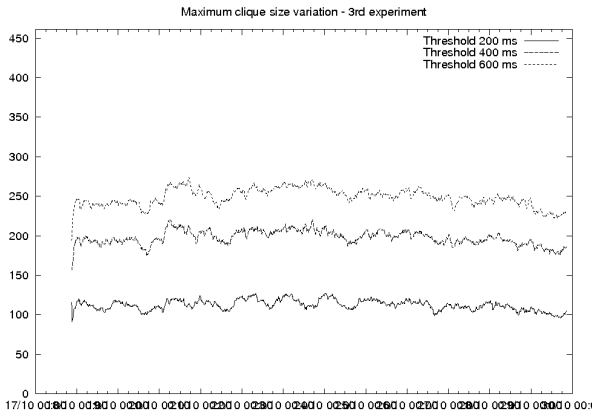

Figure 6. Maximum clique size variation for experiment 3

Experiment	Threshold	Size
1	400	59
1	600	91
1	1000	117
1	2000	149
2	200	78
2	400	153
2	600	196
3	200	42
3	400	85
3	600	114

Table 1. Maximum clique size on the intersections

5. Conclusions

Based on our experience of running a large scale network overlay on PlanetLab we found out it is hard to select a group of nodes that present a reasonably stable behavior and can fully communicate among themselves. In this work we described and evaluated an approach for finding stable cliques of PlanetLab nodes. All pairs of nodes of a stable clique can be considered to be reasonably predictable, i.e. based on the monitoring history it is not unrealistic to bet that these nodes are good choices for running experiments in PlanetLab. The monitoring strategy is based on having nodes measure their RTT to other nodes and compute the RTT variation. We ran three experiments, monitoring hundreds of nodes for several days on three different occasions in 2008 and 2009. Based on the monitoring data we checked several thresholds in order to classify nodes as stable, build the corresponding graphs, and run an algorithm for finding a

Experiment	Threshold	Average size
1	400	90.142
1	600	118.285
1	1000	147.000
1	2000	173.714
2	200	103.375
2	400	185.500
2	600	228.125
3	200	79.416
3	400	151.250
3	600	196.250

Table 2. Average maximum clique size for the period of one day

Experiment	Threshold	Average size
1	400	114.130
1	600	143.113
1	1000	167.541
1	2000	185.720
2	200	128.322
2	400	212.307
2	600	257.505
3	200	108.805
3	400	192.642
3	600	243.465

Table 3. Average maximum clique size for the period of one hour

clique on those graphs. We measured the size of the cliques for different time windows employing different thresholds, results include the sizes of the largest cliques and the number of nodes present in all cliques.

Future work includes developing a tool for PlanetLab users that accepts as input the size of a desired clique, as well as other desired parameters, and returns a suggested set of nodes to be employed. The node load and other performance metrics are certainly parameters that have to be considered. One of the challenges of finding cliques on-line is the complexity of the algorithms for finding cliques themselves, and alternatives must be investigated. Another concept that can be expanded in the future is the classification of stability. The use of an adaptive threshold seems to be an attractive alternative for on-line continous monitoring.

Acknowledgments

This work was partially supported by grants 311221/2006-8, 485671/2007-7, and 308692/2008-0 from the Brazilian Research Agency (CNPq).

References

[1] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson. Design and implementation trade-offs for wide-area resource discovery. *ACM Trans. Internet Technol.*, 2008.

[2] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization*, volume 4, pages 1–74, 1999.

[3] L. C. E. Bona, K. V. O. Fonseca, E. P. D. Jr., and S. L. V. de Mello. Hyperbone: A scalable overlay network based on a virtual hypercube. *Proc. of the 8th IEEE Int. Symp. Cluster Computing and the Grid (CCGRID)*, 2008.

[4] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 2003.

[5] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2004. ACM.

[6] R. Diestel. *Graph Theory*. Springer-Verlag, 3rd edition, 2005.

[7] M. Garey and D. Johnson. *Computers and intractability*. Freeman San Francisco, 1979.

[8] M. Huang, A. Bavier, and L. Peterson. Planetflow: Maintaining accountability for network services. *SIGOPS Oper. Syst. Rev.*, 2006.

[9] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 1988.

[10] R. Karp. Reducibility among Combinatorial Problems. *Complexity of computer computations: proceedings*, 1972.

[11] J. Liang, S. Y. Ko, I. Gupta, and K. Nahrstedt. Mon: On-demand overlays for distributed system management. In *Proceedings of USENIX WORLDS*, 2005.

[12] J. Londoño and A. Bestavros. netembed: A network resource mapping service for distributed applications. In *Proceedings of the IEEE/ACM IPDPS High-Performance Grid Computing Workshop*, Miami, Florida, USA, 2008.

[13] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, 2003.

[14] S. Ortiz. Internet researchers look to wipe the slate clean. *IEEE Computer*, 41(1), Jan. 2008.

[15] P. R. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 2002.

[16] K. Park and V. S. Pai. Comon: A mostly-scalable monitoring system for planetlab. *SIGOPS Oper. Syst. Rev.*, 2006.

[17] J. Siek, L. Lee, and A. Lumsdaine. *The boost graph library: user guide and reference manual*. Addison-Wesley Professional, 2002.

[18] E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 2007.