Monitoring Pairwise Interactions to Discover Stable Wormholes in Highly Unstable Networks

Luis C.E. Bona, Elias P. Duarte Jr., and Thiago Garrett

Federal University of Paraná - Dept. Informatics
P.O. Box 19018 Curitiba PR 81531-980 Brazil
{bona,elias,garrett}@inf.ufpr.br

Abstract. Users of large-scale testbeds often need a group of nodes with a reasonable level of stability to execute applications and experiments. Although monitoring the stability of nodes themselves is certainly part of the solution, it is important to classify and select groups of nodes according to their ability to communicate among themselves. In this work we call such groups of nodes "stable wormholes", and describe strategies to find those wormholes based on monitoring end-to-end pairwise interactions. Data acquired is used to find five different types of wormholes, each with a different stability pattern. The system was implemented in PlanetLab. Extensive experimental results are reported evaluating the proposed strategies. A comparison with another tool that selects nodes based on node stability alone is also presented. The execution of a MapReduce application shows that nodes selected with the proposed strategy ran the application significantly faster.

Keywords: Testbed Monitoring, PlanetLab, Testbed Node Selection.

1 Introduction

Uncertainty is arguably the major obstacle for developing dependable distributed systems. As even in static systems it is impossible to solve agreement problems when communication and computing time bounds are unknown and at least one node may crash [10], the situation can only get worse in more unstable networks. It is not difficult to see that depending on the level of instability it may be impossible for a distributed application to complete successfully.

As Veríssimo points out in [22], current network environments often present an spectrum of synchrony, that varies from components that present perfectly predictable behavior to those that have a completely uncertain behavior. These properties can be found in time, i.e. during the timeline of their execution systems become faster or slower, presenting lower or higher bounds to execute. The properties can also be found in space: some components are more predictable and/or faster than others, actions performed in or amongst on these nodes have better defined and/or smaller bounds. Veríssimo defined a hybrid distributed system model, Wormholes [21], in which different loci of the system have different properties which correspond to different sets of assumptions.

T. Korakis, M. Zink, and M. Ott (Eds.): TridentCom 2012, LNICST 44, pp. 146–161, 2012.

In this work we describe a monitoring strategy for finding wormholes in hybrid distributed systems, i.e. those that cannot be classified either as synchronous nor as asynchronous. Several different types of wormhole selection criteria are proposed, all of them based on the ability of pairs of nodes to communicate among themselves. At the heart of this strategy, end-to-end interactions between pairs of nodes are monitored, i.e. the response times, measured at the application level. All pairs are monitored, from each node to each other node.

Acquired data is used to build a graph that represents the system from the point of view of the different nodes. This graph is called *stability graph*. An edge of a stability graph indicates a stable communication - according to a criterium - between the corresponding nodes on which the edge is incident. Based on the obtained graph it is possible to find wormholes, i.e. groups of nodes that together have behaved in a stable way. Several wormhole selection strategies were defined, each with a different stability pattern.

We employed the global research testbed, PlanetLab [5] in order to evaluate the proposed monitoring and wormhole selection approaches. In our previous experience of running HyperBone [4] in PlanetLab we had found out that several nodes of this important testbed presented a very unstable behavior. HyperBone is 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 consisting 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. All communication patterns are possible in this environment.

Although there are tools for monitoring and selecting PlanetLab nodes for the execution of experiments [19,15,6,17,13,3], they employ criteria on the stability of the nodes themselves, such as processing load or available memory, for example. None of them selects groups of nodes also considering their ability to communicate among themselves.

We describe extensive experimental results of our wormhole selection strategies obtained from monitoring PlanetLab. The first set of experiments consists in evaluting the different wormhole selection strategies, also checking how predictable each wormhole is, i.e. how it behaves as time passes. We also checked experimentally how nodes of a wormhole selected by our tool fared when executing an experiment. We compared the performance of the wormhole with the performance of nodes selected by another PlanetLab node selection tool, SWORD. Both sets of nodes executed a MapReduce application. Results show that in most cases the wormhole nodes ran the application significantly faster.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 defines the online monitoring strategy, as well as the stability graphs

and the wormhole selection strategies. Section 4 presents experimental results obtained in PlanetLab. Conclusions follow in section 5.

2 Related Work

The original Wormholes hybrid distributed system model was proposed by Veríssimo [22,21]. This is a realistic model that is based on the fact that networks often present an spectrum of synchrony, that varies from components that present perfectly predictable behavior to those that have a completely uncertain behavior. Wormholes correspond to a subsystem - defined in time or space - that behaves in a predictable way.

In [18] the authors report the development of end-to-end dependable distributed applications and mobility-aware services in ubiquitous communication scenarios. They assume the use of off-theshelf components (COTS) and unreliable wireless communication links. The proposed strategy is based on a hybrid system architecture, which considers the existence of wormholes: subsystems with better properties than the rest of the system. A wormhole provides specialized timeliness and trustworthiness services that may be used to construct more dependable and resilient applications. The implementation of an embedded wormhole is reported.

The Partitioned Synchronous Distributed System Model [12] is another hybrid model that assumes that a subsystem is timely, i.e. provides known upper bounds on communication and computation times. In [16] the authors describe how to implement perfect failure detectors in this system. The implementation assumes the existence of a timeliness oracle, that classifies processes and channels as timely or untimely.

Another work that employs similar ideas for selecting supernodes in P2P nodes is reported in [14]. The superpeer selection problem is hard because in a P2P network a large number of superpeers must be selected from a huge and dynamically changing network in which neither the peer's characteristics nor the network topology are known a priori. A set of superpeers has similarities with a wormhole. The supernodes must be well-dispersed throughout the network, and must fulfill additional requirements such as load balance, resource needs, adaptability to churn, and heterogeneity.

We applied our proposed wormhole selection strategies to PlanetLab. There are other tools that aim at selecting PlanetLab nodes for the execution of experiments. CoMon [19] is a monitoring system specifically designed for PlanetLab. The objective is to provide information about the environment to users and administrators. CoMon collects information about the nodes themselves, such as CPU and memory usage, for example. All the information gathered by CoMon helps finding "problematic" nodes and slices. Also, CoMon provides a tool for selecting nodes which satisfy given restrictions. This work differs from Comon in respect to the monitored data and the way nodes are selected. CoMon monitors just attributes related to the nodes themselves, while the monitoring strategy presented in this work monitors the interaction between pairs of nodes. The node

selection in CoMon uses just the last data obtained from nodes, while the node selection strategies proposed in this work can use data relative to any period of time.

Vivaldi [6] is a fully distributed synthetic coordinate system whose objective is to predict the RTT between hosts, i.e. determine the RTT between two hosts without having one host effectively communicating with the other. Vivaldi's algorithm assign synthetic coordinates to each host, in such a way that the distance between the coordinates of two hosts corresponds to the RTT between them. The monitoring strategy described in this work is similar to Vivaldi as both employ the RTT as the basic monitoring metric. But in Vivaldi the RTT is an estimation that, even with good precision, does not consider faults and network problems. Furthermore, in our strategy the RTT is measured in both ways, i.e. we do not consider the RTT to be symmetric.

SWORD [3] is an application for resource discovery. It allows users to describe the desired resources with requirements related to nodes themselves and their interaction. In this way, SWORD is a tool for selecting nodes which satisfy various criteria specified by the user. But SWORD itself performs just the node selection. It is necessary to obtain the node's monitoring data from another system. Using that data SWORD is capable of selecting the best nodes that satisfy the given restrictions. There is an implementation of SWORD in PlanetLab which uses data from CoMon.

SWORD is similar to the work described in this paper, as both are strategies or selecting nodes in which applications experiments will be run. Experiments comparing the nodes selected by SWORD and by the proposed node selection strategies were executed and are described. Since it uses data from CoMon, the version of SWORD available for use in PlanetLab has the same differences to our strategy as CoMon, i.e. it uses only data related to the nodes themselves, not their interaction. Furthermore, the data corresponds to the last measurement obtained from nodes, while we considered data sampled during a whole time frame.

In [9] we describe results of our observation of PlanetLab based on a offline monitoring strategy, based on which we extracted cliques of stable nodes. In the current work we present a different, on line monitoring strategy and formally define both the monitoring strategy and five different wormhole selection strategies, one of which is the clique.

3 Online Monitoring and Node Selection

This section describes the proposed monitoring strategy. We then define the stability graph that is built by applying a stability criterium to the monitoring data, and finally define five different types of stable wormholes that can be obtained from the stability graph.

3.1 Monitoring Pairwise Interactions

Consider a network, represented as directed graph $G = (V, E)$, where V is the set of vertices corresponding to network nodes, and E the set of edges such that

the ordered pair (i, j) is an edge if node i communicates directly with node j, i.e. without employing intermediate nodes.

The purpose of the proposed monitoring strategy is to provide data that can be used to select a set of nodes W, called a stable wormhole, such that for any two nodes $r, s \in W$, $W \subseteq V$, the communication initiated by r with s can be considered stable. In other words, wormhole nodes are able to communicate among themselves in stable way, which of course depends on the stability criteria defined below.

Every node of the monitored network executes a monitoring daemon, specified as follows. Periodically, $\forall i \in V$, node i sends a message to each other node and waits for a reply. Upon receiving the reply message, the Round Trip Time (RTT) is computed using the local clock, and tuple $(i, j, rtt_{i,j}, timestamp)$ is stored. The *timestamp* corresponds to the local time instant at which the measure was obtained. We assume that clocks are roughly synchronized, for example with the level of accuracy that is obtained with the Network Time Protocol (NTP) [2] in the Internet.

Monitoring is supposed to run at the application level, thus RTT measurements vary not only because of network issues, but also due to the situation of the node himself, for instance the number of processes on the scheduler's queue and CPU usage.

Even if a daemon stores recently colleted measurements locally, at some point data is sent to a *central monitor* that eventually collects measured data from all daemons. Due to the large amount of data that this strategy generates, and the fact that recent data is more important that previous samples, the central monitor summarizes sequences of tuples for predefined time frames. For example, in our PlanetLab experiment described in the next section, we summarized data per hour, day, month and year. Summarization computes both the mean and standard deviation of measurements for a given pair of nodes within a given time frame. Thus, the summary for ordered pair (i, j) for a given time frame $T = [t_0, t_1]$ and set of tuples $(i, j, rtt_{i,j}, timestamp)$ is the average $\mu_{i,j,T} =< rtt_{i,j} | t_0 \leq timestamp \leq t_1 >$, plus the standard deviation $\sigma_{i,j,T}$ used as a measument of the dispersion.

Given two consecutive time frames it is also possible to compute the average and standard deviation of the most recent timeframe taking into account the summary obtained for the previous time frame.

3.2 Building the Stability Graph

From the monitoring data acquired, the central monitor builds a so called *stability graph* S for time frame $T = [t_0, t_1]$. Before the stability graph is formally defined, we give the definition of "stability" itself, i.e. we define the criteria used to classify a given pairwise interaction as stable or unstable. Consider the ordered pair of nodes (i, j) monitored as described above. A threshold θ is defined as the maximum value allowed for an RTT sample. We then compute the frequency in which the round trip times fall below θ, considering the two sets of tuples $(i, j, rtt, timestamp)$ and $(j, i, rtt, timestamp)$, such that $t_0 < timestamp < t_1$.

Node i is said to consider node j to be stable if at least a fraction p of the obtained $rtt_{i,j} \leq \theta$. The analogous procedure is used to check whether node j considers node i to be stable. In our PlanetLab experiments we employed $p = 90\%$. Stability Graph $T = (V_T, E_T)$ is an non-directed graph defined as follows. An edge $(i, j) \in E_T$ (thus node $i \in V_T$ and node $j \in V_T$) if an only if both node i considers node j to be stable within T, and node j considers node i to be stable within T.

3.3 Finding Stable Wormholes

After the stability graph S is built, we reach the final stage, i.e. searching for wormholes, i.e. a connected set of nodes with a reasonable level of stability, that could be employed to run distributed applications that require a "reasonable" level of stability. Although at first we assumed that finding a wormhole was equivalent to finding a clique in S, i.e. there should be an edge between any two vertices in V_T of S, experience taught us less strict criteria could lead to more stable, more predictable wormholes. In particular, the clique represents an overly restricted criteria, and the number of nodes (wormhole size) is often not large enough. In this subsection we present the five different types of stable wormholes we defined and evaluated: Minimum Degree, Highest Minimum Degree, K-Core, Core and Stable Clique.

Minimum Degree. The least strict strategy for finding wormholes is the one based on Minimum Degree. This strategy returns a stable wormhole W, such that node $i \in W$ if the degree of node i in S $deg_S(i) \geq d_{min}$, for a given minimum degree d_{min}. Thus the Minimum Degree strategy filters nodes by their degrees in the stability graph. Only nodes with degree higher than the minimum degree which is entered as an input parameter are selected.

The rationale behind this strategy is that it may suffice to find nodes that can communicate in a a stable fashion with a large number of other nodes. However it is possible that two nodes in a Minimum Degree wormhole are not able to communicate with each other in a stable way.

Highest Minimum Degree. The Highest Minimum Degree strategy receives as input parameter the desired number of nodes in the stable wormhole, m. Given that parameter, an algorithm is executed on stability graph S to find such a group of m nodes with the highest possible minimum degree, D_{min}. This strategy thus selects a wormhole with a specified minimum size (number of nodes) and with the highest possible minimum degree for that size. A trivial polynomial algorithm can be used to find a Highest Minimum Degree wormhole.

***k*-Core.** The k-Core strategy finds a k-core on the graph representing the system. A k-Core is the largest group of nodes that form a subgraph C of the stability graph S that has minimum degree equal to k. In other words, this strategy selects a group of nodes that have a minimum degree among themselves, i.e. each selected node has a degree higher than or equal to k within the wormhole. The

input parameter in this case is the minimum degree k. This strategy is more restrictive than the Minimum Degree and Highest Minimum Degree, since the degrees in those two other strategies may involve nodes not in the wormhole.

The algorithm to compute a k-core from S is polynomial. It starts by building subgraph $S_1 = (V_1, E_1)$, such that node $i \in S_1$ iff $deg_S(i) \geq k$. Note that an edge $(i, j) \in E_T$ is also in E_1 if both $i, j \in S_1$. The process is then repeated: create subgraph $S2 = (V_2, E_2)$ selecting from all nodes of S_1 those that have degree greater than or equal to k. Eventually the resulting subgraph is equal to the original graph, this is the k-core, the largest set of nodes in the stability graph S that have minimum degree k among themselves.

Core. The Core strategy finds the stability graph core. A *core* is the largest group of nodes that form a subgraph with the highest minimum degree possible; i.e. the core of a graph is the k-Core with the highest value of k that is not empty. This strategy thus returns the largest set of nodes that have the highest minimum degree among them.

In order to find the Core wormhole the algorithm described above in the k-Core strategy is executed on a binary search for the largest k that returns a non-empty set of nodes. Initally $k = n/2$, half the number of nodes in the system; the lower bound (b_l) is 0 and the upper bound (b_u) is n. If the k-core is non-empty, then k is set to $(b_u + k)/2$ otherwise k is set to $(b_l + k)/2$. This process is repeated until an empty set of nodes is returned; the core is the last non-empty k-core.

Stable Clique. The Stable Clique strategy finds a clique [8] - a complete subgraph - of the stability graph. In a clique there is an edge between every pair of nodes. Finding cliques is a NP-hard problem [11]. Therefore finding the largest clique in a graph with a large number of edges, such as the graphs created by the strategy described in this work, is impracticable, since the nodes - in which experiments will be run - must be selected quickly. To address this problem, the Stable Clique strategy uses two parameters: the minimum size of the clique and the maximum processing time. We employed a well-known depth-first search algorithm for generating all maximal cliques of an undirected graph, that employs pruning and is feasible in practice [20]. The algorithm is executed until a clique with a size higher than or equal to the specifed minimum size is found. Alternatively, if such clique is not found within the specified time limit, the largest clique found so far is selected. This strategy is the most restrictive one, since there must be an edge between every pair of selected nodes.

4 Experimental Results

This section describes experimental results obtained from a PlanetLab implementation of the wormhole selection strategies. When the experiments were executed in 2011, PlanetLab consisted of 983 nodes. Although the complete set of nodes were monitored, the number of nodes that actually ran the system varied

from 500 and 600 nodes. This was because a large number of nodes continuously alternated being online and offline, and some other nodes remained unreachable during the whole period of the experiments. In all experiments described in this section each node measured the RTT to each other node every 5 minutes.

Two groups of experiments were conducted. The first group consisted of an evaluation of the node selection strategies. Several stability graphs were built using monitoring data obtained during continuous periods each consisting of 3 weeks. The second group of experiments, described in subsection 4.2, aimed at comparing nodes selected by our strategy with nodes selected by another PlanetLab monitoring tool. This comparison was performed by executing a MapReduce application on nodes selected both with our tool and with the other tool.

4.1 Evaluation of the Wormhole Selection Strategies

This experiment was based on monitoring data collected from January 30, 2011 at 00:00, to February 19, 2011, at 23:59 (UTC -3), for a total of 21 days. Stability graphs were built at each hour during this period. The RTT thresholds employed for generating the stability graphs were 0.05s, 0.1s, 0.15s and 0.2s; 4 graphs were built per hour, for a total of 2016 graphs. Both the average and maximum degrees of all stability graphs were computed. Figure 1 shows the average and maximum degrees observed considering a threshold of 0.1s. Figure 2 shows the analogous results when a threshold of 0.2s was employed. It is not difficult to see that the values for both the average and maximum degrees were significantly higher when the threshold was 0.2s. For instance, the average degree was about 60 during the whole period in which the threshold was 0.1s; as the threshold was increased to 0.2s, the average degree increased to 130. Higher threshold values are not reported because the return meaningless results: a very high threshold does not filter nodes and interactions which are highly unstable (variations fall within the threshold).

After the stability graphs were generated, we executed the several wormhole selection strategies. Each strategy was executed to find wormholes on the stability graphs. The purpose of this experiment is to evaluate the performance of the different strategies as time passes.

The Minimum Degree is the simplest strategy: we counted the number of nodes in the stability graphs with degree greater than or equal to the minimum specified. The minimum degrees evaluated were 50, 100, 150 and 200. Figure 3 shows the number of nodes selected with the Minimum Degree strategy for several days period using a RTT threshold of 0.1s. For minimum degree 50, for example, the number of nodes selected remained close to 450 during the observation time, note that this is about 80% of the total number of monitored nodes, which ranged from 500 to 600 nodes.

Next we evaluated the performance of the Highest Minimum Degree strategy. In this strategy we specify the number of nodes, and verify the maximum degree of any such group in the stability graphs. Figure 4 shows, for a threshold of 0.1s, the highest minimum degree identified over 21 days for groups of 50, 100, 150,

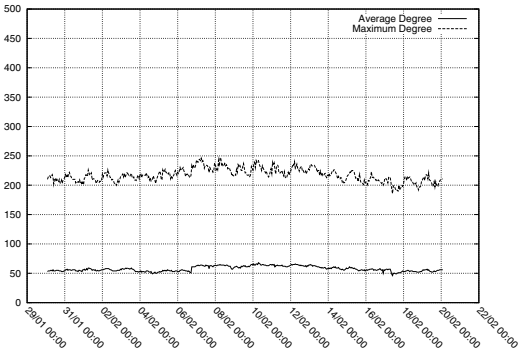

Fig. 1. Average and maximum degrees of the stability graphs built with threshold of 0.1s

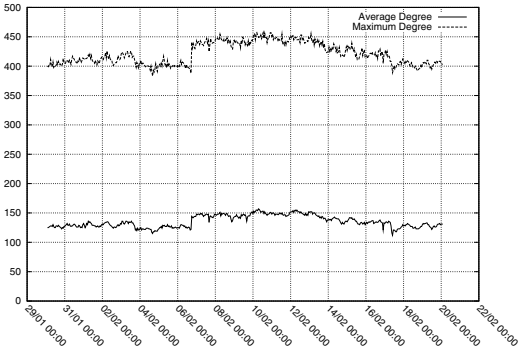

Fig. 2. Average and maximum degrees of the stability graphs built with a threshold of 0.2s

200 and 300 nodes. For 300 nodes, for example, the highest minimum degree was about 100; for groups of 50 nodes the highest minimum degree was about 180.

In order to evaluate the Core strategy, cores were extracted from all stability graphs. The number of selected nodes and the minimum degree among them were recorded. Figure 5 shows the results for a threshold of 0.1s, considering the whole observation period. We can observe that the number of varied widely, while the minimum degree presented a low variability. Furthermore, frequently the mininum degree was close to the number of nodes selected, thus the subgraph induced by the selected nodes presents high density, i.e. there are edges between most node pairs, which means that in the graph cores obtained each node considers the majority of the others to be stable.

Our main purpose in evaluating the Stable Clique strategy was to check whether a Stable Clique maintains itself as time passes. Three Stable Cliques were computed on the first stability graph considering a 1-day monitoring

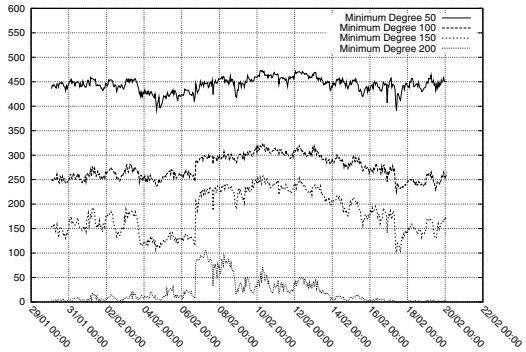

Fig. 3. Number of nodes selected with the Minimum Degree Strategy for a threshold of 0.1s

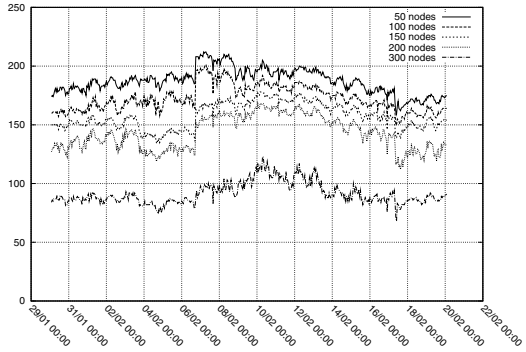

Fig. 4. Highest Minimum Degree for groups of different sizes, threshold of 0.1s

period. During that day we computed for each of the remaining stability graphs if that Stable Clique was still there. The RTT threshold used to compute the initial clique was equal to 0.05s; in order to check whether the clique was still there we used a threshhold of 0.1s. The reason we computed the initial clique with this low threshold (the most strict of all thresholds employed in all experiments, employed to find both an initial clique as well as a initial core in the next experiment we report below) was that it would allow us to initially select nodes that were interacting in a very stable pattern. Then when checking the clique we would employ a larger threshold to allow for some variation.

Figure 6 shows how each of the three fared during a representative time frame (one day). We show for each group of nodes, the number of nodes that remained fully connected among themselves. The threshold was of 0.1s. It is possible to see that only one group of nodes remained as a clique for most of the time (not all the time though). From our observations we reached the conclusion that the Stable Clique employs a criterium that is too restrictive, and the selected group of nodes most probably will not hold the desired properties for long.

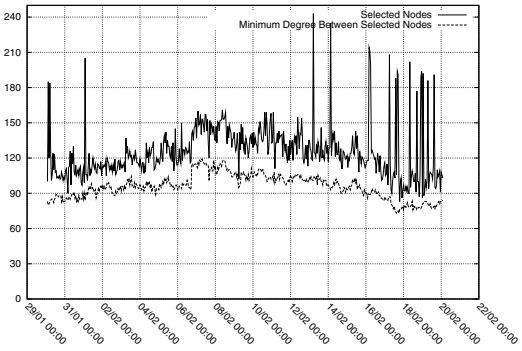

Fig. 5. Minimum degree and number of nodes selected with the Core strategy, threshold 0.1s

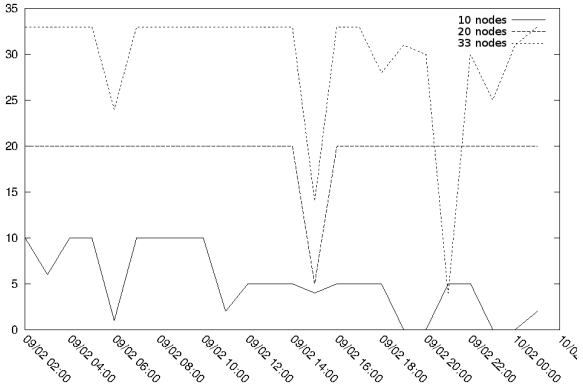

Fig. 6. The performance of Stable Cliques during one representative day

The Core strategy - which is less restrictive than the Stable Clique - proved to be a better choice for selecting a group of nodes that presented among themselves a stable communication pattern for a longer time frame. In order to check how the nodes selected using the Core strategy behave as time passes, we selected nodes using this strategy the first stability graph of a one day period. Again for the initial selection we employed a strict threshold of 0.05s. 62 nodes were selected. Figure 7 shows the average, minimum and maximum degree for these 62 nodes during a 1-day observation period, with threshold equal to 0.1s. Note that as in the Stable Clique experiment above here we employed a larger threshold to monitor the core, in comparison with the original threshold with which the core was selected at first. This makes room for some fluctuation in the stability among nodes.

The average degree had a very low variation during the whole observation period, as well as the maximum degree - which remained constant. The minimum

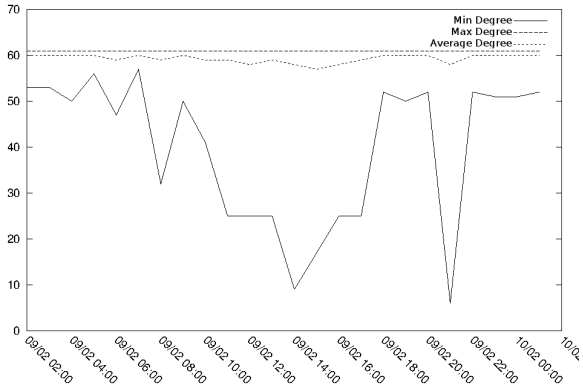

Fig. 7. Average, minimum and maximum degrees of nodes selected with the Core strategy

degree, however, had a much higher variation. However, even when the minimum degree varied this did not impact the average degree. This indicates that few nodes were affected by some instability. Also, the average degree was close to the maximum degree during the whole period, showing that the majority of the nodes in the cores presented a stable behavior among themselves during the whole period.

Discussion. Although the Minimum Degree and Highest Minimum Degree strategies consistently return a larger number of nodes, these strategies do not guarantee that the group of nodes all present a stable communication pattern among themselves. On the other hand, the Stable Clique strategy only selects a group of nodes such that *all* of them are able to communicate in a stable pattern with all others. However this proved to be too restrictive, and difficult to sustain as time passes. The results show that the Core strategy is the best: it selects good sized groups of nodes that are able to interact among themselves in a pretty stable way and remained so for longer.

4.2 Comparison with Another Tool

In order to compare the performance of nodes selected with the proposed strategies and those selected with another tool, we executed experiments using MapReduce. MapReduce [7] is a software framework aimed at distributed computing on large data sets. MapReduce requires the definition of a mapping function, which transforms the data in key/value pairs, and a reduction function, which gathers the key/value pairs to obtain the final result. A MapReduce application can be automatically parallelized and executed in a set of hosts. A node divides the input data between several other nodes, each of which then repeats the process. The set of nodes form a tree. When a node completes its mapping, the result

is sent back to its parent in the tree. This process continues until the first node receives all mapped data and performs the final computation (reduction).

The MapReduce implementation used in this experiment was Apache Hadoop [1]. Hadoop is a framework for the execution of distributed applications. It provides a distributed file system, HDFS (Hadoop Distributed File System), and an implementation of MapReduce. When Hadoop is started the HDFS and the system responsible for executing the applications (including MapReduce ones) are instantiated in all nodes. In order to execute MapReduce applications with Hadoop, the input data must be inserted in HDFS, since all applications will be run on top of this file system. All nodes then have access to any part of the data, which can be effectively stored in any other node.

For the sake of comparing the performance of nodes selected by the proposed strategies we employed SWORD [3], which was described above in section 2, and provides the closest functionality to our tool, being specifically designed for selecting PlanetLab nodes for running experiments. We selected our Core strategy for this comparison, since it gave our best results when all strategies were compared. The values chosen for all parameters used, both for SWORD and for the Core strategy, were as restrictive as possible. The attributes used in SWORD for selecting nodes were the response time of the nodes (based on the CoMon server), and the one minute load which should be less than or equal to 10 in all experiments.

The MapReduce application used was a *wordcount*, which counts how many occurences of each word appear in an input file. The text file used in the experiments had size 1GB, and was created randomly. The experiment consisted in executing this application several times, both using nodes selected by SWORD and selected by the Core strategy. For every measured metric we computed the mean and the standard deviation, as well as the maximum median, and minimum value.

Two different experiments were conducted, in both cases we employed 100 PlanetLab nodes. After the nodes were selected, Hadoop was started on all nodes. The input file was then inserted in HDFS. Then, the MapReduce application was run 15 times, one after the other. After these executions completed, measurements were recorded. This procedure was performed twice (for a total of 30 runs), each time using a group of nodes selected differently.

The first experiment was executed on July 15th 2011. 100 nodes were first selected using SWORD at 10:50 (UTC -3), after that the experiment was run on those nodes. Later 100 nodes were selected using the Core strategy at 18:00 (UTC -3) and the experiment was executed again. Table 1 shows the results obtained from the 30 executions (15 using each set of nodes). Nodes selected by the Core strategy ran the application significantly faster. Except for the standard deviation, all values corresponding to the nodes selected by SWORD were about two times the values obtained for the nodes selected by the Core strategy.

The second experiment was executed on July 18th 2011. This experiment consisted of the same procedures employed for the first one, but with an important difference: we first executed the application on the nodes selected by the Core

Table 1. Comparison results (first experiment): execution time

	SWORD	**Core**
Average	22min 59s	9min 25s
Standard Deviation	12min 18s	6min 3s
Median	18min 54s	7min 34s
Lowest	6min 47s	4min 21s
Highest	58min 19s	28min 49s

Table 2. Comparison results (second experiment): execution time

	SWORD	**Core**
Average	20min 17s	8min 18s
Deviation	17min 33s	1min 49s
Median	16min 57s	8min 23s
Lowest	7min 46s	5min 10s
Highest	83min 25s	11min 42s

strategy. This was done to check whether there was an impact of the period of the day in which the experiments were run. In this experiment, 100 nodes were first selected using the Core strategy at 11:28. After the execution finished, 100 nodes were selected using SWORD at 14:40 and the experiment was run again. Table 2 shows the results obtained from the 30 executions (15 per set of nodes). As in the first experiment, nodes selected by the Core strategy ran the application significantly faster, but in this experiment the standard deviation was much higher for nodes selected by SWORD. Also, the highest execution time for nodes selected by SWORD was by far higher than the highest execution time for nodes selected by the Core strategy. The low standard deviation might indicate that the nodes selected by the Core strategy presented a "good" stability during the whole experiment duration.

Discussion. In both experiments, nodes selected by the Core strategy ran the application significantly faster than nodes selected by SWORD. These results show that even on a highly unstable network such as PlanetLab, the strategies described in this paper were able to select nodes that presented a reasonable stable communication pattern among themselves.

5 Conclusions

Based on the fact that current network environments often present a spectrum of synchrony, that varies from components that present perfectly predictable behavior to those that have a completely uncertain behavior, in this work we described strategies to find sets of nodes that can be considered to be stable according to various criteria. Such groups of nodes are called stable wormholes and the criteria for discovering wormholes are based on monitoring end-to-end

pairwise interactions. Monitoring data is used to build stability graphs which in turn are used to find five different types of wormholes, each with a different stability pattern, ranging from cliques to sets of nodes with a minimum degree. The system was implemented in PlanetLab, and we report results of a comparison between different wormhole selection strategies. Experiments comparing the performance of nodes selected by the proposed strategies with nodes selected by a tool that is based on node stability alone are also presented. The execution of a MapReduce application on those nodes show that, in most cases, nodes selected by the proposed strategies ran the application significantly faster.

Future work includes developing a tool for PlanetLab users that accepts more input parameters, such as the size of a desired wormhole. Developing new strategies for finding wormholes, such as the connectivity of the subgraph is also a new research direction. Another issue that can be expanded in the future is the classification of stability using other criteria such as an for instance adaptive thresholds.

Acknowledgments. This work was partially supported by grant 308692/2008-0 from the Brazilian Research Agency (CNPq).

References

1. Apache Hadoop, `http://hadoop.apache.org` (accessed at July 29, 2011)
2. NTP: The Network Time Protocol, `http://www.ntp.org/` (accessed em April 18, 2011)
3. Albrecht, J., Oppenheimer, D., Vahdat, A., Patterson, D.A.: Design and Implementation Trade-offs for Wide-area Resource Discovery. ACM Trans. Internet Technol (2008)
4. Bona, L.C.E., Fonseca, K.V.O., Duarte Jr., E.P., Mello, S.L.V.: HyperBone: A Scalable Overlay Network Based on a Virtual Hypercube. In: Proc. of the 8th IEEE Int. Symp. Cluster Computing and the Grid, CCGRID (2008)
5. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: PlanetLab: An Overlay Testbed for Broad-coverage Services. SIGCOMM Comput. Commun. Rev. (2003)
6. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A Decentralized Network Coordinate System. In: SIGCOMM 2004: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New York, NY, USA. ACM (2004)
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, Berkeley, CA, USA, vol. 6. USENIX Association (2004)
8. Diestel, R.: Graph Theory, 3rd edn. Springer (2005)
9. Duarte Jr., E.P., Garrett, T., Bona, L.C.E., Carmo, R.J.S., Zuge, A.: Finding Stable Cliques of PlanetLab Nodes. In: The 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010 DCCS (2010)
10. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM (1985)

11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)
12. Gorender, S., de Araújo Macêdo, R.J., Raynal, M.: An Adaptive Programming Model for Fault-Tolerant Distributed Computing. IEEE Transactions on Dependable and Secure Computing (2007)
13. Liang, J., Ko, S.Y., Gupta, I., Nahrstedt, K.: MON: On-demand Overlays for Distributed System Management. In: Proceedings of USENIX WORLDS (2005)
14. Lo, V., Zhou, D., Liu, Y., GauthierDickey, C., Li, J.: Scalable Supernode Selection in Peer-to-Peer Overlay Networks. In: Proceedings of the 2005 Second International Workshop on Hot Topics in Peer-to-Peer Systems, HOT-P2P 2005 (2005)
15. Londoño, J., Bestavros, A.: 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)
16. Macedo, R.A., Gorender, S.: Perfect Failure Detection in the Partitioned Synchronous Distributed System Model. In: International Conference on Availability Reliability and Security, ARES (2009)
17. Massie, M.L., Chun, B.N., Culler, D.E.: The Ganglia Distributed Monitoring System: Design, Implementation And Experience. Parallel Computing (2003)
18. Ortiz, H., Casimiro, A., Veríssimo, P.: Architecture and Implementation of an Embedded Wormhole. In: Proceedings of the 2007 Symposium on Industrial Embedded Systems. IEEE Industrial Electronics Society (2007)
19. Park, K., Pai, V.S.: CoMon: A Mostly-scalable Monitoring System for PlanetLab. SIGOPS Oper. Syst. Rev. (2006)
20. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science (2006)
21. Veríssimo, P., Casimiro, A.: The Timely Computing Base model and architecture. IEEE Transactions on Computers. SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory) (2006)
22. Veríssimo, P.: Travelling through Wormholes: a new look at Distributed Systems Models. SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory) (2006)