Computer Communications 110 (2017) 35–47

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Improving the performance and reproducibility of experiments on large-scale testbeds with *k*-cores

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

Department of Informatics, Federal University of Parana, P.O. Box 19018, Curitiba, PR 81531-980, Brazil

ARTICLE INFO

Article history:
Received 17 January 2017
Revised 19 May 2017
Accepted 28 May 2017
Available online 31 May 2017

Keywords:
Testbed node selection
Experiment reproducibility
k-core
PlanetLab

ABSTRACT

Large-scale testbeds provide realistic environments for the experimentation and evaluation of new protocols and distributed applications. In order to be successful, these experiments must be executed on sets of nodes that present a reasonable level of stability, and it is important to ensure their reproducibility. In this work we describe strategies to select sets of testbed nodes based on monitoring information. The system is modeled as a stability graph in which the vertices correspond to testbed nodes and there is an edge between two vertices if their communication is classified as stable. We investigate the performance of different topologies embedded in the stability graph to run experiments on PlanetLab. Results show that the *k*-core outperforms the other strategies in terms of their impact on the performance and reproducibility of the experiments. A *k*-core is a maximal subgraph of *G* in which all vertices have degree at least *k*. The average execution time of distributed applications executed on a *k*-core was up to 59% lower, and the variation on the results obtained was reduced by up to 29% when compared to other alternatives.

1. Introduction

In order to evaluate new network protocols and distributed applications, experiments must be executed under realistic conditions. Large-scale testbeds such as PlanetLab [1], OneLab [2], and Geni [3] allow the evaluation of new proposals under real-world conditions. However, testbeds are often too unstable, to the point that it is sometimes difficult even to select a set of testbed nodes that are able to communicate with each other during the whole experiment execution [4]. Depending on how unstable the testbed is, it may not be possible to run distributed applications or to obtain meaningful results. Furthermore, it is highly unlikely that an experiment executed on a set of testbed nodes selected randomly is reproducible [5].

Network testbeds (actually most network environments) present a spectrum of synchrony [6]: while a subset of the system presents perfectly predictable temporal behavior, other subsets are completely unpredictable. This range of behavior can be observed in time and space, i.e. a single part of the system may change its observable behavior along the time. At a given time, a large set of nodes all of which can communicate with each other according to a predictable pattern might not even exist.

In this work we describe strategies to select sets of testbed nodes to run reproducible experiments that produce meaningful results. At the heart of these strategies, end-to-end interactions between pairs of nodes are monitored, with round-trip-times measured at the application level. The communication between all pairs of nodes is monitored. Monitoring data is used to build a stability graph that represents the system from the points of view of the different testbed nodes, which correspond to the graph vertices. If there is an edge between two vertices of a stability graph, they can communicate in a stable way—according to a pre-defined criterium, i.e. there is a high level of synchrony or temporal predictability between those nodes. Based on the stability graph it is possible to find sets of nodes that can be considered to be stable as a group. A node selection strategy returns a subset of the vertices of the stability graph. Five node selection strategies were defined, namely: stable clique, minimum degree, highest minimum degree, *k*-core and maximum *k*-core, described next.

The stable clique strategy returns a clique of the stability graph, a subset of the vertices that induces a complete subgraph. The minimum degree strategy receives as input a lower bound on the vertices degrees, and returns a subset of vertices that has degree equal or higher to that bound. The highest minimum degree strategy receives as input the number of vertices to return, and finds the highest degree that allows that constraint to be met, i.e. all nodes returned have degree greater than or equal to that highest degree. The *k*-core strategy finds the *k*-core of the stability graph, i.e. a subgraph in which all vertices have degree at least *k*, in other

* Corresponding author.
E-mail addresses: tgarrett@inf.ufpr.br, thgarrett@gmail.com (T. Garrett), bona@inf.ufpr.br (L.C.E. Bona), elias@inf.ufpr.br (E.P. Duarte Jr.).

http://dx.doi.org/10.1016/j.comcom.2017.05.016
0140-3664/

words, a subset of nodes which can communicate in a stable fashion with at least k other nodes in the same subset. The maximum k-core strategy receives as input the number of vertices to return and searches for the highest possible value of k such that the corresponding k-core has at least the desired number of vertices.

We employed the global research testbed, PlanetLab [1] to evaluate the proposed node selection strategies. PlanetLab is a highly unstable environment [7]. Although there are tools for monitoring and selecting PlanetLab nodes [8–13], most employ criteria on the stability of the nodes themselves, e.g. node CPU load or available memory. To the best of our knowledge no other strategy is able to select a set of nodes based on their ability to communicate among themselves in a stable way.

We describe several experiments conducted in PlanetLab in order to evaluate the proposed strategies and their impact on the performance and reproducibility of the experiments. Three groups of experiments were conducted. In the first group, we evaluated the five different node selection strategies. The maximum k-core strategy presented the best results.

In the second group of experiments, we compared the quality of nodes selected by the maximum k-core strategy with the quality of nodes selected with another PlanetLab tool, SWORD, when executing a MapReduce application. The average execution time of a distributed application executed on nodes selected by the maximum k-core strategy was up to 50% lower than the average execution time of the same application executed on nodes selected by SWORD.

In the third group of experiments, we evaluated the impact of selecting PlanetLab k-cores on the reproducibility of the experiments. We employed different types of distributed applications, featuring either network-intensive or CPU-intensive. These applications were executed a large number of times for a period of 40 days, on both k-cores and on nodes selected by an alternative strategy. We then evaluated the precision of the results obtained by each set of nodes. The k-cores presented not only significantly lower average execution times but, most important, up to 29% smaller coefficients of variation, for all types of applications. In other words, our strategy was able to improve the reproducibility of the experiments by improving the precision of the results. Furthermore, our strategies were also able to select sets of nodes that remained stable for longer periods of time.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 specifies the monitoring strategy, the construction of stability graphs and the five strategies for node selection. Section 4 presents the three groups of experiments and is followed by the conclusion in Section 5.

2. Related work

This section starts with an overview of distributed systems models that assume the existence of a set of stable nodes within a large possibly unstable distributed system. Next, we present related testbed monitoring tools including those that select testbed nodes to run experiments. Then we describe works which have a focus on the reproducibility of experiments. This is followed by a few representative works on running experiments in large-scale testbeds. Then we conclude this section with other works about the k-core topology.

A few theoretic distributed system models have been proposed that are related to our work. The Wormholes hybrid distributed system model proposed by Veríssimo [6] is based on the fact that networks often present a 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. The existence of a subset of stable nodes within a large network is also considered in [14]. The authors describe the development of end-to-end dependable distributed applications and mobility-aware services in ubiquitous communication scenarios, assuming the use of off-the-shelf components (COTS) and unreliable wireless communication links. In the proposed strategy, a set of stable nodes provides specialized timeliness and trustworthiness services that can be used to construct more dependable and resilient applications. The Partitioned Synchronous Distributed System Model [15] is another hybrid model that assumes that there is a timely subsystem which 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.

The majority of testbed monitoring tools [8,11,12,17] provide information about both the network and testbed nodes—such as CPU usage, memory usage and network traffic, for example. Among these systems, CoMon [8] was specifically designed for PlanetLab. CoMon is a monitoring system designed to collect information about PlanetLab nodes. All data collected by CoMon helps finding "problematic" nodes and slices, or selecting nodes that match given restrictions. CoMon was launched in 2004, but the service was shut down in 2012.

The monitoring strategy presented in this work differs from CoMon—and other testbed monitoring systems—with respect to the monitored data. CoMon just monitors attributes related to a single node, while the monitoring strategy proposed in the present work monitors the interaction between pairs of nodes.

Vivaldi [10] is a tool that claims to provide a scalable approach to obtain the RTT (Round-Trip-Time) between pairs of PlanetLab nodes. After measuring the RTT for some pairs of nodes, Vivaldi predicts the RTT for the other pairs. Vivaldi employs a fully distributed synthetic coordinate system that allows the RTT prediction. The system assigns synthetic coordinates to each host, in such a way that the distance between the coordinates of two hosts corresponds to the RTT between them. Other strategies based on the prediction of pairwise measurements can be found in [18,19]. They employ distributed structures and matrix factorization in order to improve scalability and precision. The monitoring strategy described in the present 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 if it presents good precision, does not take into account faults and network problems.

Other tools allow testbed node selection [8,9,12,13]. SWORD [13] is one of such tools, available in PlanetLab. This tool allows users to describe the desired resources as well as requirements related to nodes themselves and their interactions. In this way, SWORD is a tool for selecting nodes which satisfy various criteria specified by the user. Note that SWORD itself only performs the node selection, it is not a monitoring system, and must obtain node monitoring data from another system. Using that data SWORD is capable of selecting the best nodes that satisfy the user criteria. The PlanetLab implementation of SWORD[1] uses data from CoMon. However, since CoMon is unavailable since 2012, it is no longer possible to use SWORD to select PlanetLab nodes.

SWORD is similar to the node selection strategies proposed in this work, as both select testbed nodes. However, since SWORD uses data from CoMon, node selection is based on data related to the nodes themselves, not their interaction. Furthermore, the data used by SWORD corresponds to the last measurement obtained from nodes, while we consider data sampled during a whole time frame. Although SWORD is not available anymore (because CoMon

[1] http://sword.cs.williams.edu/.

has been shut down) we present experimental results comparing SWORD to our strategy in one set of experiments in Section 4.

Another related node selection strategy is for choosing superpeers in P2P networks [20]. The superpeer selection problem is hard because in a P2P network a large number of superpeers must be selected from a huge and dynamic network in which neither the peer's characteristics nor the network topology are known a priori. A set of superpeers must be well-dispersed throughout the network, and must fulfill additional requirements such as load balance, resource requirements, adaptability to churn, and heterogeneity.

The reproducibility of PlanetLab experiments is discussed in [5]. The authors argue that experimental results obtained in PlanetLab are hard to reproduce, since the testbed is not a fully controlled environment. They conclude that in the case of long-running applications, researchers should be able to identify network performance patterns and understand the level of performance and reliability their application achieves. On the other hand, for short-term experiments, they suggest researchers should search for resources on the network that fulfill their needs for the timeframe of the execution. The node selection strategies proposed in the present work aim at selecting nodes capable of increasing the performance and reproducibility experiments, independent of being long or short-term experiments.

A framework for automated distributed experimentation, Weevil, is presented in [21]. Weevil focuses on controlling parameters and workload generation over multiple runs in order to obtain repeatable results. The authors state that controlling the experimentation environment is necessary but obtaining such control over a testbed such as PlanetLab is not an easy task.

A survey of experiment management tools for distributed systems is presented in [22]. These tools help researchers to control the workflow of their experiments, automating tasks such as deployment, execution and gathering results. In [23], the authors describe a methodology for conducting repeatable experiments in public testbeds, while in [24] the authors present a measurement study, performed in the PlanetLab testbed, which highlights some issues that researches must be aware of when conducting experiments.

The k-core was first applied to networks in the social networks context [25]. Several results have been published on the application of k-cores on complex networks. The topological organization of complex networks based on k-cores is presented in [26–28]. In [29] the authors present a model of the Internet structure, at the Autonomous System level, decomposing the Internet itself into k-cores—called a k-shell decomposition. A similar k-core decomposition of Internet nodes as infrastructure nodes on top of which an overlay network is embedded is presented in [30]. The authors show that their strategy achieves good results, improving TCP performance, for example. k-cores have also been used to identify good nodes to disseminate information [31,32] using epidemic algorithms—the idea is that nodes in k-cores can have a higher spreading influence.

3. Selecting testbed nodes to run experiments

This section is organized as follows. The proposed monitoring strategy is described in Section 3.1. Then the stability graphs built from the monitoring data are defined in Section 3.2 and finally five strategies for selecting nodes of the stability graphs are presented in Section 3.3.

3.1. Monitoring pairwise interactions

Let $G = (V, E)$ be a complete undirected graph representing a network, in which V is the set of vertices corresponding to the network nodes, and E the set of edges corresponding to the network links. An edge (i, j), $i, j \in V$, means that node i can reach node j without employing intermediate nodes. We consider communications at the application layer, thus every node can directly reach every other node in the network and G is a complete graph.

The purpose of the proposed monitoring strategy is to acquire data to build a subgraph $S = (V, E')$ of G, $E' \subseteq E$, i.e. data obtained with the monitoring strategy is used for removing edges from E, resulting in subgraph S. The construction of this subgraph, based on a criterium applied to the monitoring data, is detailed in Section 3.2.

In the proposed monitoring strategy, each node executes a daemon responsible for periodically sampling the round trip time to each other node: $\forall i, j \in V$, node i sends a message to each other node j and waits for a reply, computing the round trip time $rtt_{i,j}$ using the local clock. Tuple $(i, j, rtt_{i,j}, timestamp)$ is stored locally. The *timestamp* corresponds to the local time instant at which the measure was obtained. Clocks are assumed to be roughly synchronized, for instance with the level of accuracy that is obtained with the Network Time Protocol (NTP) [33] in the Internet.

Since the $rtt_{i,j}$ is computed at the application level, the measurements obtained vary not only because of network conditions, but also due to multiple factors affecting the node itself. CPU usage, number of processes on the scheduler queue, context switching and interrupt handling are examples of factors that may also affect $rtt_{i,j}$ in addition to the network.

All data collected by each node is periodically sent to a central server. This central server can then build stability graphs as defined next in subsection 3.2.

3.2. Building a stability graph

The central server uses monitoring data to build a so-called *stability graph* S for a time frame $T = [t_0, t_1]$, i.e. S is a graph representing the system during the period of time indicated by T, according to a stability criterium. For building S, it is necessary to classify each pairwise interaction as *stable* or *unstable* according to that criterium, as defined below.

Let (i, j) be a pair of nodes monitored as described above. A threshold θ is defined as an upper bound for the $rtt_{i,j}$ samples. Let $R_{i,j} = \{(i, j, rtt_{i,j}, timestamp) : t_0 \leq timestamp \leq t_1\}$ be the set of tuples acquired by node i regarding node j during the time frame T. Let $R_{i,j,\theta} = \{(i, j, rtt_{i,j}, timestamp) : (i, j, rtt_{i,j}, timestamp) \in R_{i,j}, rtt_{i,j} \leq \theta\}$ be the set of tuples from $R_{i,j}$ such that each value $rtt_{i,j} \leq \theta$. Let p be the frequency in which the $rtt_{i,j}$ samples should fall below θ, $0 \leq p \leq 1$. Node i is said to consider node j to be stable within T if $\frac{|R_{i,j,\theta}|}{|R_{i,j}|} \geq p$, i.e. at least a fraction p of the obtained values must satisfy $rtt_{i,j} \leq \theta$. In our PlanetLab experiments we employed $p = 0.9$, i.e. at least 90% of the RTT samples must be less than or equal to the threshold for a node to consider another to be stable.

For building the stability graph $S = (V, E')$ each pair of nodes is checked: an edge $(i, j) \in E'$ if and only if node i considers node j to be stable *and* node j also considers node i to be stable, i.e. the classification must be symmetric. Just for the note, in [4] we show that from extensive data obtained from monitoring hundreds of PlanetLab nodes, around 10% of the pairs of nodes have asymmetrical classification, i.e. node i considers node j to be stable but node j does not consider node i to be stable. From this definition, it is possible to say that a stability graph S is the complete graph G with the "unstable" edges removed.

3.3. Selecting nodes

Node selection is performed based on the stability graph defined above. In this subsection we describe five strategies to se-

lect a subset of nodes $W \subseteq V$, which are "stable enough" to run distributed applications that require a "reasonable" level of stability. Each node selection strategy receives as input different parameters and returns W, the subset of selected nodes.

Before defining each strategy we give our notations. $S = (V, E')$ is a stability graph. $d_S(v)$ is the degree of vertex v in S. $\delta(S)$ is the minimum degree of all vertices of S, i.e. $\delta(S) = \min_{\forall i \in V} d_S(i)$. $S[U]$ is the subgraph of S induced by the subset of nodes $U \subseteq V$. W is the set of selected nodes. The vertices of a graph are interchangeably called nodes along the text.

In our early work [4], we assumed that a clique in the stability graph was the best approach to select testbed nodes. In other words, there should be an edge between any two vertices of W in the graph $S[W]$, i.e. every pair of selected nodes classified each other as stable according to our criterium during the time frame considered when creating S. We believed that such hard restriction would result in a high quality subset of nodes, i.e. a cohesive set of nodes in terms of stability and efficiency. However, we empirically found out that less strict structures could result in more stable and/or predictable sets of nodes that can last over time [34]. Moreover, a clique is an overly restricted structure which results in small sets of nodes often not large enough to run real experiments.

Thus, we defined five node selection strategies: stable clique, minimum degree, highest minimum degree, k-core and maximum k-core. Each strategy is described below.

Stable clique

The stable clique strategy returns a clique of a stability graph. A clique [35] is a subset of pairwise adjacent vertices, i.e. a subset of vertices that induces a complete subgraph. The clique W returned by this strategy may not be the largest clique on the stability graph S, since finding such clique is a NP-hard problem [36]: it can be impracticable to search for the maximum clique, specially considering that stability graphs can have a large amount of edges (i.e. they are dense graphs), as we found out in [4].

To address the processing time issue, we defined a strategy that receives two input parameters: the minimum size of the clique and the maximum processing time. A depth-first search algorithm is used to find maximal cliques in stability graphs [37]. This algorithm is executed either until a clique with size higher or equal to the given minimum size is found or the processing time limit is reached. If no such clique is found within the time limit, the largest clique found is returned.

Minimum degree

The minimum degree strategy receives a single input parameter, the minimum degree d_{min}. It returns a set of nodes $W = \{i : i \in V, d_S(i) \geq d_{min}\}$, i.e. all nodes returned have degree at least d_{min} in the stability graph S. The rationale behind this strategy is that it may suffice to find nodes that can communicate in a stable fashion with a large number of other nodes. However, it is possible that two nodes in a subset returned by this strategy are not able to communicate with each other in a stable way.

Highest minimum degree

The highest minimum degree strategy receives a single input parameter: the minimum number of nodes l to be selected. A binary search is executed on a stability graph S in order to find the maximum minimum degree D_{min}, such that selecting nodes with the minimum degree strategy, described above, results in a subset W, $|W| \geq l$, i.e. this strategy selects a subset of nodes W with a specified minimum size and with the highest possible minimum degree for that size.

k-core

The k-core strategy finds a k-core W on a stability graph S. A k-core is the largest subset of nodes $W \subseteq V$ that induces a subgraph $H = S[W]$ of the stability graph S that has minimum degree equal to k, i.e. $\delta(H) = k$. In other words, this strategy selects a set of nodes that have a minimum degree among themselves, i.e. each selected node has a degree higher than or equal to k in the subgraph induced by the selected subset W. Therefore, the input parameter for this strategy 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 selected subset. We used the standard polynomial algorithm to find a k-core [25]: the k-core is the subgraph remaining after repeatedly removing vertices of degree less than k.

Maximum k-core

The maximum k-core strategy receives as input parameter the minimum number of nodes l to be selected. This strategy returns a k-core W with the highest value of k such that $|W| \geq l$, i.e. it maximizes k such that the corresponding k-core has at least the desired number of vertices. The value of k is found by repeatedly selecting nodes with the k-core strategy, increasing k, until the resulting k-core has size less than l: the previous k-core is then returned.

4. Experimental results

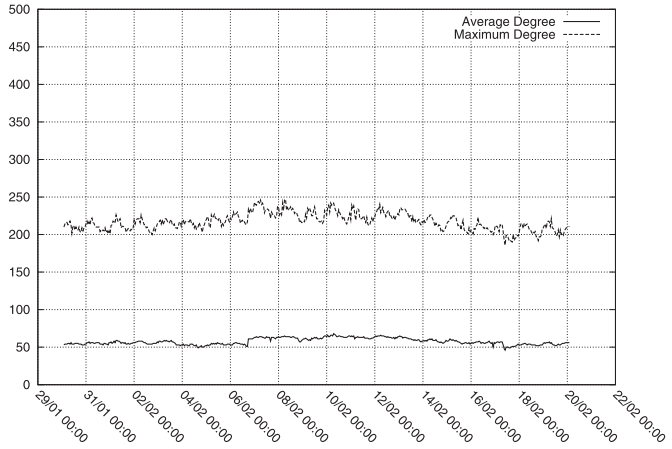
In this section we describe experimental results conducted in PlanetLab. These experiments evaluated the quality of nodes selected by the proposed strategies and their impact on the reproducibility of the experiments. For all experiments reported in this section, the monitoring strategy was executed on about 1000 PlanetLab nodes. Although all these 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 in which the experiments were executed. The sampling rate of RTT measurements was 5 min, i.e. each node measured the RTT to each other node every 5 min. Three groups of experiments were conducted as described next.

In the first group of experiments, we first evaluated how the stability graphs and all proposed node selection strategies fared during a long observation period. This group of experiments is described in Section 4.1. From the evaluation of the five proposed node selection strategies we concluded that the maximum k-core strategy was the best approach for selecting nodes. We therefore conducted a second group of experiments for evaluating the quality of the nodes selected by the maximum k-core strategy when executing a distributed application. For this evaluation, we executed the same application on both nodes selected by the maximum k-core strategy and by another PlanetLab node selection tool, SWORD, comparing the results. These experiments are described in Section 4.2.

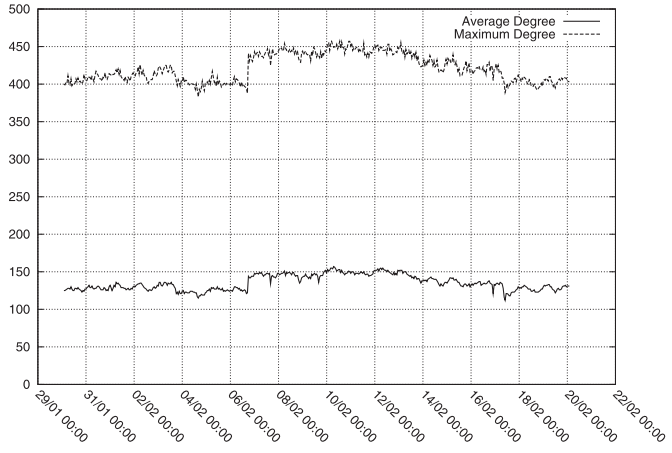
In the third group of experiments, we experimentally evaluated the impact of selecting PlanetLab k-cores on the reproducibility of the experiments. Different distributed applications with different requirements in terms of network/CPU were executed repeatedly in PlanetLab on nodes selected both by the maximum k-core strategy and by another alternative strategy. This group of experiments is described in Section 4.3.

4.1. Evaluation of the node selection strategies

The purpose of this group of experiments is to evaluate the stability graphs and all five proposed node selection strategies. 983 PlanetLab nodes were monitored for 21 days. The time frame for creating the stability graphs from the monitoring data was of 1 h, i.e. a stability graph was created using data obtained during each hour. The threshold θ employed for generating the stability graphs



(a) Threshold (θ) 0.1s



(b) Threshold (θ) 0.2s

Fig. 1. Average and maximum degrees of the stability graphs.

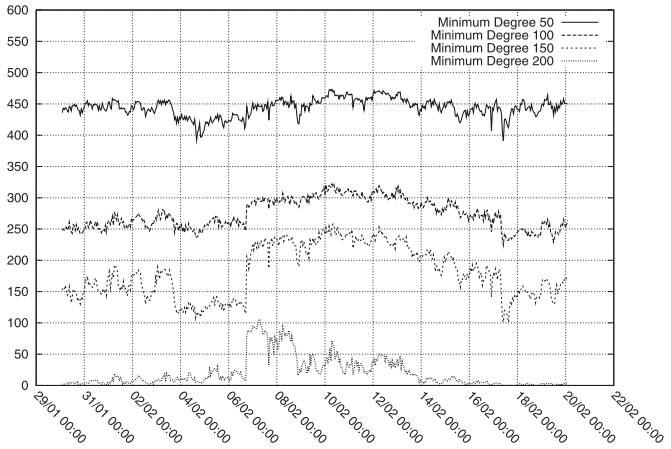


Fig. 2. Minimum degree strategy: number of nodes selected for $\theta = 0.1$ s and different values of d_{min}.

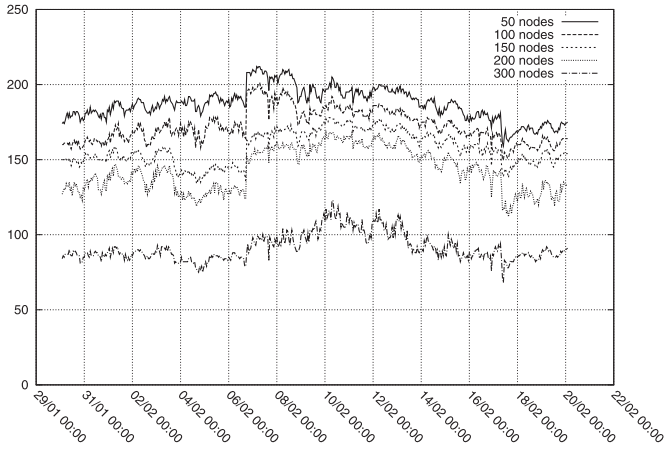


Fig. 3. Highest minimum degree strategy: D_{min} for $\theta = 0.1$ s and different values of l.

was equal to 0.05 s, 0.1 s, 0.15 s and 0.2 s, thus four graphs were built per hour, for a total of 2016 stability graphs.

In this subsection we present (1) experimental results showing the impact of the threshold (θ) on the average and maximum degrees of the stability graphs. Then we present (2) an evaluation of the minimum degree strategy in terms of the number of nodes selected when the minimum degree required varies from 50 to 200. In (3) the highest minimum degree strategy is evaluated, we show the minimum degree obtained when the number of nodes varies from 50 to 300. The stable clique is evaluated as follows (4). We first find a clique, then we observe what happens to this clique as time passes. This is exactly how we evaluate the k-cores in (5), but we also show the number of nodes selected and the minimum degree of the k-cores. Finally, the overall conclusions of this group of experiments are presented in (6).

4.1.1. Stability graphs

At first, the average and maximum degrees of the stability graphs for the whole period were evaluated: they are shown in Fig. 1 for threshold (θ) equal to 0.1 s and 0.2 s. It is clear that for both metrics their values are significantly higher for $\theta = 0.2$ s. For instance, the average degree was about 60 during the whole period for $\theta = 0.1$ s while for $\theta = 0.2$ s, the average increased to about 130. Results for higher values of θ are not reported: a high threshold may not filter "unstable" edges, resulting in stability graphs too similar to the network complete graph G.

4.1.2. Minimum degree strategy

In order to observe how the minimum degree strategy fared over time, four sets of nodes were selected using this strategy in each stability graph built, each employing a different input parameter for the minimum degree d_{min}: 50, 100, 150 and 200. Fig. 2 shows the number of nodes selected by the minimum degree strategy for each value of d_{min} within the 21 days period. In this experiment, $\theta = 0.1$ s. For $d_{min} = 50$, for example, about 450 nodes were selected. Note that this number is close to the number of nodes effectively running the monitoring at any given time—which ranged from 500 to 600.

4.1.3. Highest minimum degree strategy

Five sets of nodes were selected with the highest minimum degree strategy on each stability graph in order to observe the performance of this strategy. The input parameters for the minimum number of nodes l were 50, 100, 150, 200 and 300. Fig. 3 shows, for $\theta = 0.1$ s, the highest minimum degree D_{min} found by the strategy on each stability graph and for each value of l. For $l = 300$, for example, D_{min} was about 100 during the whole period, while for $l = 50$ it was about 180.

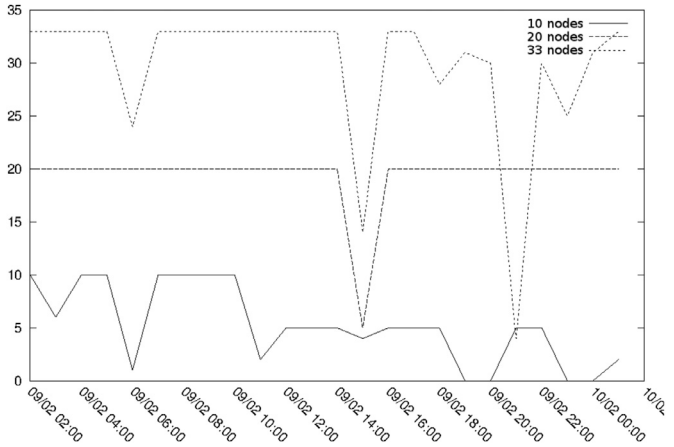


Fig. 4. Stable clique strategy: number of nodes with the highest degree possible during one day for three different initial stable cliques.

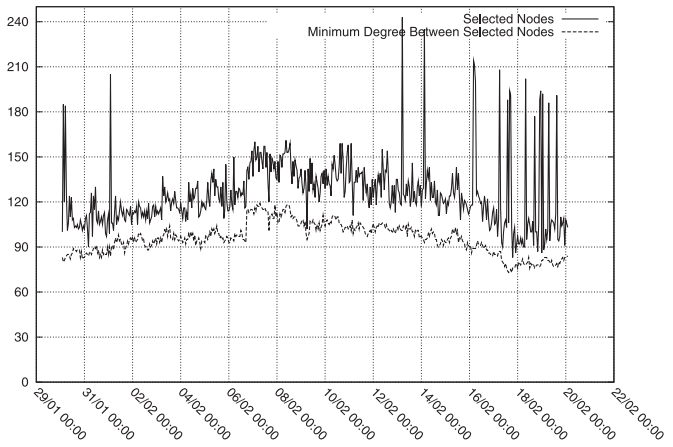


Fig. 5. Maximum k-core strategy: number of nodes selected and the minimum degree k for $\theta = 0.1$ s.

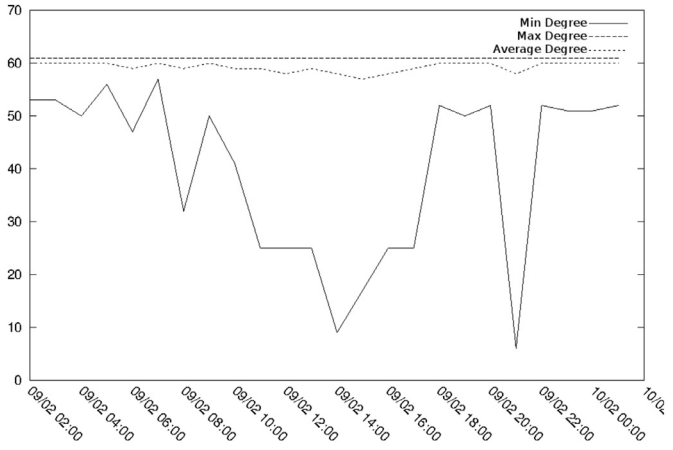


Fig. 6. Maximum k-core strategy: average, minimum and maximum degrees of the same k-core during one day for $\theta = 0.1$ s.

4.1.4. Stable clique strategy

The stable clique strategy was evaluated by observing their performance (did the selected nodes still form a clique?) as time passes. For this evaluation, we used stability graphs corresponding to the period of one day and θ equal to 0.05 s and 0.1 s. Three sets of nodes were selected using the stable clique strategy in the stability graph corresponding to the first hour of the period, with $\theta = 0.05$ s. The input parameters employed were 10, 20 and 30 for the minimum size and 10 min for the maximum processing time. The validity of these sets of nodes as cliques was then checked on each stability graph with $\theta = 0.1$ s, i.e. for each stability graph with $\theta = 0.1$ s it was checked if each set of nodes was still a valid clique. The reasoning for using a lower threshold and then checking the cliques with a higher threshold is that nodes selected with a more restrictive threshold may have a more stable behavior when observed using a higher threshold, keeping themselves as a valid clique during larger periods of time. Fig. 4 shows, for each set of nodes, the number of nodes with the highest degree possible within the set (number of nodes minus 1), in a period of one day and $\theta = 0.1$ s. Whenever all nodes in a set have the highest degree possible, the set is a clique. In Fig. 4, only the set with 20 nodes kept itself as a valid clique for most of the observed period. In contrast to our early assumptions [4], nodes selected with the stable clique strategy are more likely to not last that long as a stable set of nodes, despite the fact that they were selected using a very restrictive criterium.

4.1.5. k-core strategies

In order to evaluate both the k-core and maximum k-core strategies, nodes were selected in each stability graph by searching for the highest value of k such that the resulting k-core is not empty. This was done by using the maximum k-core strategy with the input parameter for the minimum number of nodes l equal to 1, i.e. the highest value of k such that the resulting k-core has at least one node. Fig. 5 shows the size of each k-core selected and the minimum degree k of the corresponding induced subgraphs. Again in this experiment the stability graphs were computed with $\theta = 0.1s$. The size of the selected k-cores presented a high variability, in contrast to k, which presented lower variability. Furthermore, the values of k were mostly close to the size of the k-cores, thus the subgraph induced by the selected nodes had high density, i.e. there are edges between most pairs of nodes, which means that each node considered the majority of the others as stable.

Since cliques usually do not stay valid for a significantly period of time, we found out that the k-core topology, which is less restrictive than cliques, might be a better approach so that the selected set of nodes endure longer in terms of stability. In order to evaluate how k-cores behave as time passes, we selected a set of nodes using the maximum k-core strategy, with $l = 1$, in the stability graph corresponding to the first hour of a period of one day, with $\theta = 0.05$ s. We then observed the behavior of the exact same set of nodes in each subsequent stability graph computed at each hour of the one day period, with $\theta = 0.1$ s, following the same procedure used to the clique evaluation above. A set of 62 nodes was selected in this way. Fig. 6 shows the average, minimum and maximum degree for the same set of nodes during the one day period. The maximum degree stayed the same for the whole observed period, while the average degree had low variation. The minimum degree had a much higher variation. However, the average degree was not affected by this high variation on the minimum degree, which indicates that just a few nodes had their degrees lowered. Furthermore, the average degree was close to the maximum degree, indicating that most nodes in the k-core selected presented a stable behavior among themselves during the whole observed period.

4.1.6. Conclusions

In this subsection we described evaluation results obtained from experiments executed on 981 PlanetLab nodes for 21 days. The question we aim to answer is: given the evaluation results, which strategy is the best? We claim that the best strategy is the one that selects the largest set of nodes that present the highest degree among themselves—which means that this is a large set of

nodes that are able to communicate with each other, i.e. a large set of nodes that present a high cohesion.

The minimum degree and highest minimum degree strategies were able to select the largest sets of nodes, as expected, since those strategies employ the weakest restrictions regarding the cohesion of the selected nodes. The selected nodes that present the highest cohesion were obtained with the stable clique strategy. However this is a very small set of nodes and it quickly looses its properties as time passes. Thus, overall, we conclude that the maximum *k*-core strategy is the best, as it gives large sets of nodes that keep their properties for long periods of time. However, note that depending on the nature of the experiment, the number of nodes needed or the virtual topology that is embedded in the testbed, different strategies and parameters can be employed. For instance, some experiments may require the largest possible number of nodes even if they present low cohesion.

4.2. Quality of selected nodes

In the second group of experiments, we evaluated the performance of a distributed application when executed both on nodes selected by the proposed strategies and on nodes selected by SWORD [13], a tool described in Section 2.

The distributed application was based on MapReduce [38] running *wordcount*, which counts the occurrences of each word of a text file. MapReduce is a software framework for distributed computing on large data sets. A MapReduce application requires two functions: 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. Input data is divided among several nodes. Each node applies the mapping function to its data and/or divides the data again and assigns parts of the data to other nodes, creating a highly distributed, parallel execution.

The implementation of MapReduce used in the *wordcount* application was Apache Hadoop [39]. Besides using the Hadoop implementation of MapReduce we also used the Hadoop Distributed File System (HDFS). Upon starting, Hadoop instantiates the HDFS in all nodes. Input data must be first inserted in the distributed file system so that the MapReduce application has access to any part of the data on any node.

The node selection strategy chosen for the comparison with SWORD was the maximum *k*-core, since it proved to select a proper sized set of nodes that can keep itself reasonably stable for longer periods of time. For the remainder of this subsection, the maximum *k*-core strategy is referred to as MKC, and the nodes selected by MKC are referred to as MKC nodes. Similarly, nodes selected by SWORD are referred to as SWORD nodes.

Two sets of experiments were conducted in PlanetLab. In both sets of experiments the MapReduce *wordcount* application was executed on 100 PlanetLab nodes. For selecting MKC nodes, we employed as input the minimum number of nodes (100) and stability graphs were created with threshold $\theta = 0.1$ s. The input parameters for SWORD were the response time of the CoMon server which was less than or equal to 0.3 s and the one minute load of the nodes which was less than or equal to 10, in both experiments. These values were as restrictive as possible so that we still selected 100 nodes—smaller values resulted in less than 100 nodes. These parameters were chosen for SWORD because they are the most similar to the ones we employed in our strategies. The *wordcount* input data—a text file—had size 1 GB in both sets of experiments, and was created randomly.

Each set of experiments executed consisted of five steps. First, nodes are selected using SWORD or MKC. Then Hadoop is started on all selected nodes. Next, the input file is inserted in the distributed file system. The *wordcount* application is then executed 15 times. Finally, measurements are obtained and Hadoop is stopped.

Table 1
Execution times for the first set of experiments.

	SWORD	MKC
Average	22 min 59 s	9 min 25 s
Standard deviation	12 min 18 s	6 min 3 s
Confidence interval (95%)	± 6 min 13 s	± 3 min 4 s
Median	18 min 54 s	7 min 34 s
Lowest	6 min 47 s	4 min 21 s
Highest	58 min 19 s	28 min 49 s

Table 2
Execution times for the second set of experiments.

	SWORD	MKC
Average	20 min 17 s	8 min 18 s
Standard deviation	17 min 33 s	1 min 49 s
Confidence interval (95%)	± 8 min 53 s	± 55 s
Median	16 min 57 s	8 min 23 s
Lowest	7 min 46 s	5 min 10 s
Highest	83 min 25 s	11 min 42 s

The metrics considered were the average execution time, standard deviation, 95% confidence interval, median, and the highest and lowest values observed.

Experiments of the first set were first executed on 100 SWORD nodes. Then, they were executed again on 100 MKC nodes. Table 1 shows the results for the 15 executions employing each set of nodes. MKC nodes ran the application significantly faster. Except for the lowest values observed, which was 6 min for SWORD and 4 min for MKC, all values corresponding to SWORD nodes were about two times larger than the corresponding values for the MKC nodes.

The second set of experiments was also executed in the same way as described above. Table 2 shows the results obtained from the 15 executions employing each set of nodes. MKC nodes ran the application significantly faster as in the first set of experiments: the average execution time for SWORD nodes was about two times larger than for MKC nodes. However, the standard deviation was much larger for SWORD nodes in the second set of experiments—about nine times larger. Furthermore, the highest execution time for SWORD nodes was far higher (83 min versus 11 min) comparing to the highest execution time for MKC nodes.

MKC nodes ran the MapReduce *wordcount* application significantly faster than SWORD nodes in both sets of experiments. These results show that the strategies defined in the present work were able to select sets of nodes which were faster and more consistent in terms of the range of values observed.

4.3. Reproducibility of experiments

In this group of experiments, we evaluated the reproducibility of experiments in PlanetLab when employing nodes selected both by our strategies and nodes selected by an alternative strategy. Distributed applications with different resource requirements were executed on these sets of nodes. We measured the precision of the results obtained with each set of nodes in order to evaluate the level of reproducibility achieved with each one.

The experimental results showed in this section feature the coefficient of variation (CV) as the measurement of precision for the execution times of the distributed applications, i.e. we measured the precision of the obtained results by computing the CV of the execution times measured when executing the applications several times on sets of nodes selected in different ways. The CV is the ratio of the standard deviation to the mean, thus it is a better measurement for the precision than the standard deviation alone, since it can be used to compare data sets with widely different means.

For the CV and all others metrics reported in this section we employed 2.5% trimmed statistics in order to improve the robustness of the results, excluding outliers [40]. In other words, we excluded the 2.5% lowest and highest values when computing our metrics.

In contrast to the single MapReduce application employed in the experiments described above, in this group of experiments we employed three different applications, each with different resource requirements, ranging from high network traffic and low CPU usage to high CPU usage and low network traffic. The purpose on varying the resource requirements is to evaluate how they impact the precision of the obtained measurements. The MapReduce framework is used again in two of these applications, while the third is based on the BitTorrent[2] protocol. The MapReduce implementation employed was Mincemeatpy.[3] This implementation uses a single central node, the master, which distributes map and reduce functions as tasks to all the other nodes, the workers. The three applications—called Triangular, Bit and Torrent—are described next.

The Triangular application, built using the MapReduce framework, computes the first n triangular numbers. The nth triangular number T_n is given by $T_n = \sum_{i=1}^{n} i$. In our experiments, this application computes the first 8192 triangular numbers, featuring low CPU usage and high network traffic, since computing T_n values has low complexity but there is a large amount of (*key, value*) pairs to be transmitted between nodes. For instance, in the Triangular application the map function generates over 33 million (*key, value*) pairs.

The Bit application is based on the Bitcoin[4] decentralized payment system and was built using the MapReduce framework. The Bitcoin system makes use of a virtual currency with the same name—the bitcoin. All transactions are done directly between users, without any intermediates or a trusted central authority. The security of these transactions relies on cryptography, and it requires high computing power in order to validate transactions. It is not feasible to validate an incorrect transaction. The validation is done on a public P2P network: each participating node offers its computing power validating transactions and receives bitcoins in return as reward. This is known as *mining* and users willing to gain bitcoins by validating transactions are known as *miners*. The validation process consists of repeatedly computing a SHA-256 hash function [41] on top of the transaction data and a parameter v until the value of v that satisfies a certain criterium is found. The Bit application employed in our experiments implements part of the validation process: each node computes the SHA-256 hash function for a subset of v values and a random given key – which would be the transaction data on a real validation. From all (*key, value*) pairs generated by the map function only a few are returned by the worker nodes to be further reduced: the purpose was to create a CPU-intensive application with low network traffic. For instance, the SHA-256 hash was computed for about 7 billion values of v, but just about 6000(*key, value*) pairs were transmitted between nodes, ignoring the rest.

The Torrent application is based on the well known P2P file sharing protocol, *BitTorrent*. The implementation of this protocol employed in our application is Transmission,[5] the default BitTorrent client in several modern GNU/Linux distributions, such as Ubuntu and Fedora. The Torrent application transfers a 27 MB file, generated randomly, from one node to all other nodes using the Transmission BitTorrent client, during a fixed amount of time which was set to 10 min in our experiments. We used as metric for this application the average time taken by the first 80 nodes to receive the complete file, i.e. we measured the execution time from the start of the application until the transfer of the file completes in each node, then we computed the average execution time for the first 80 nodes to completely receive the file. The Transmission client was set to allow each node to communicate with all other nodes simultaneously, thus the Torrent application features high network traffic and low CPU usage.

The three applications were executed in four different sets of PlanetLab nodes, each with size 150. Two of these sets were selected using the maximum k-Core strategy, referred to as `core-150ms` and `core-150ms-everyround` for the remainder of this subsection. The other two sets were selected with an alternative strategy, referred to as `ping.c3sl` and `ping.uk` for the remainder of this subsection. These sets are described next.

The `core-150ms` and `core-150ms-everyround` sets were selected using the maximum k-core strategy and the parameter for the minimum number of nodes was set to 150. The stability graphs for selecting these sets were created with a time frame of one hour and $\theta = 150$ ms. The difference between these two groups is that the nodes of the `core-150ms` set were selected before the experiments started and remained the same, while the nodes of the `core-150ms-everyround` set were selected each time the applications were executed, using monitoring data from the last hour. The minimum degree of the subgraphs induced by the selected sets of nodes was about 120 in all experiments. These two sets of nodes are referred to as "k-core sets" for the rest of this subsection.

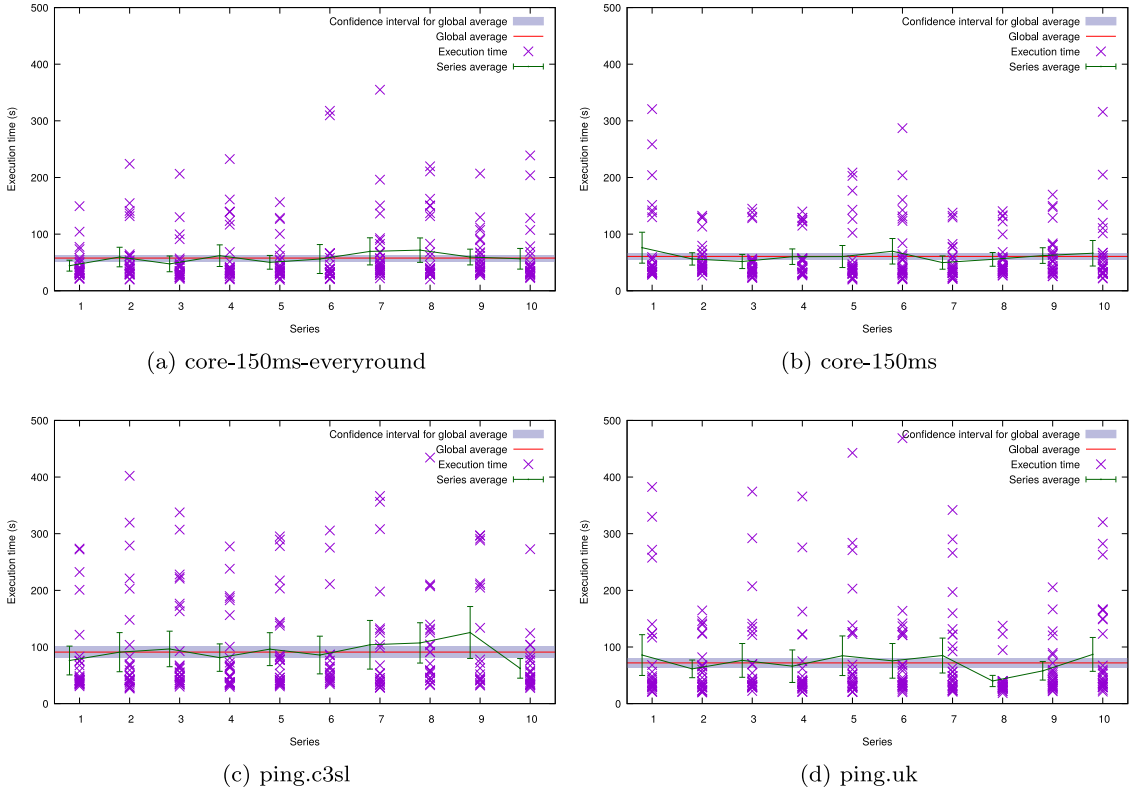
The `ping.c3sl` and `ping.uk` sets were selected using an alternative strategy based on *ping*, which computes the round trip time from a host to another and is based on the ICMP echo request/reply message. In this strategy, a central node measures the average round trip time—from five samples—from itself to each other node using *ping*. The 150 nodes with the lowest values are then selected. Both sets of nodes were selected before the experiments started and remained the same. In the previous experiments, described in Section 4.2, we employed SWORD to select nodes. However, at the time we executed the reproducibility experiments, presented in the current subsection, CoMon was no longer available, thus it was not possible to use SWORD again. For selecting the `ping.c3sl` set we employed a central node in Brazil. For selecting the `ping.uk` set we employed a central node in the UK, which was always among the nodes selected by the maximum k-core strategy. These two sets of nodes are referred to as "*ping*" sets for the rest of this subsection.

Selecting nodes based on measurements from a central node is a common strategy used by other node selection tools—including SWORD. Therefore, the objective of selecting nodes based on a *ping* from a central server is to compare that strategy with our proposed strategy that is based on pairwise monitoring.

The experiments were organized in rounds and series. A series is made of 30 rounds of experiments and each round consists of executing the three applications on each set of nodes. A round of experiments takes about 150 min to complete and the next round starts immediately after the previous one, with no restriction on the time of the day or week day. The series are started manually and lasted for about 3 days. We performed 300 rounds of experiments organized in 10 series.

Next, we report facts about the composition of the selected sets of nodes. In one of the series, for the `core-150ms-everyround` set, 262 distinct nodes were selected in all 30 rounds. Among these 262 nodes, 59 were present in all rounds and 123 were present in more than 20 rounds. On the other hand, 42 nodes were selected five times or less and only 11 were selected just once. Sets `ping.uk` and `core-150ms` were similar to each other, since about 100 nodes were the same for both. However, set `ping.c3sl` was mostly different from the others: just 11 nodes were also present in `ping.uk` and 27 in `core-150ms`.

[2] http://www.bittorrent.org/beps/bep_0003.html.
[3] https://github.com/michaelfairley/mincemeatpy.
[4] http://bitcoin.org/.
[5] http://www.transmissionbt.com.



(a) core-150ms-everyround (b) core-150ms

(c) ping.c3sl (d) ping.uk

Fig. 7. Execution times for the Triangular application.

Table 3
Metrics for the Triangular application, in seconds.

Set of nodes	core-150ms-everyround	core-150ms	ping.c3sl	ping.uk
Average	52.79	56.36	85.08	65.59
Standard deviation	37.26	36.65	71.57	60.09
Minimum	21.02	21.69	29.13	20.95
Maximum	206.89	202.69	307.26	320.42
Coefficient of variation	0.705	0.650	0.841	0.916

The execution times of the Triangular application for all series of experiments and each set of nodes is shown in Fig. 7. The execution times in each round are plotted and series are shown from 1 to 10. The green line shows the average execution time of each series along with the 95% confidence interval The red line with a gray shadow shows the average execution time for all 300 rounds and the 95% confidence interval. It is possible to see in the figure that there are several execution times that are significantly higher than the corresponding series average. However, this is more frequent for *ping* sets. Furthermore, the distance between the average execution time of each series and the average execution time of all series is smaller for the *k*-core sets.

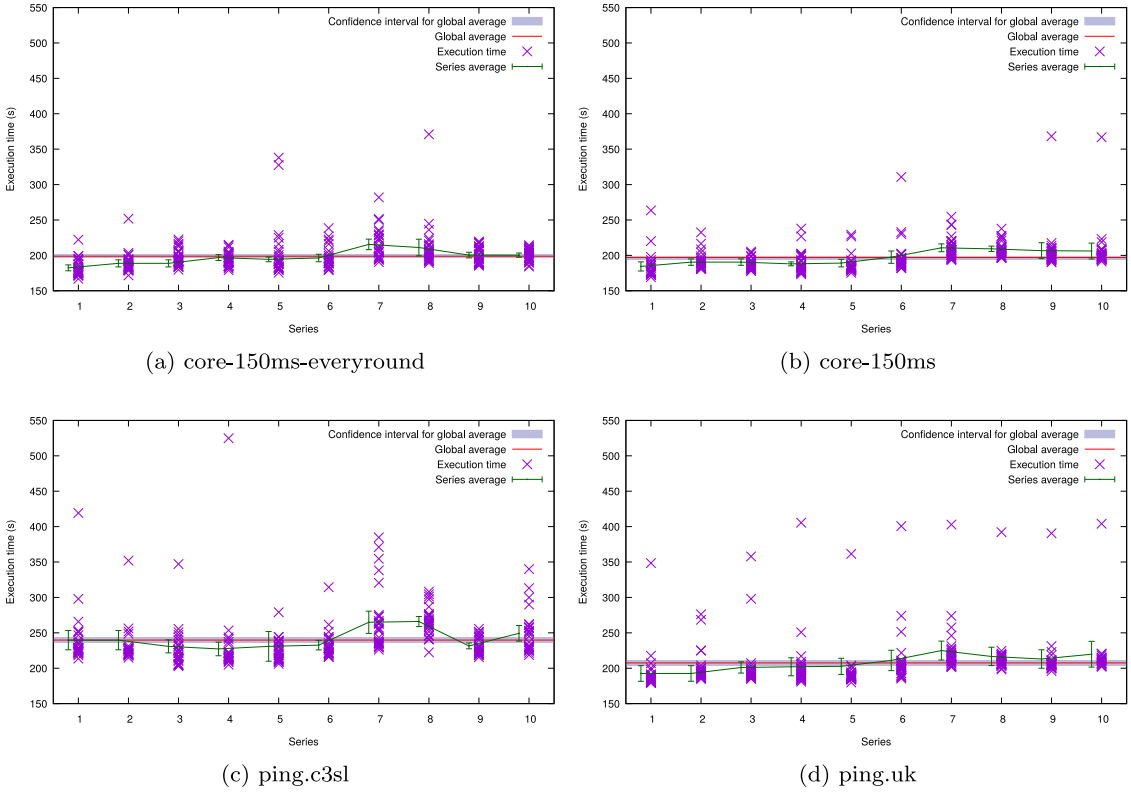
In order to further evaluate the different sets of nodes selected, we compiled metrics regarding the execution time of the Triangular application considering all 300 rounds, for each set of nodes. Table 3 shows the average execution times of the Triangular application, in seconds, along with the standard deviation, minimum and maximum values, as well as the CV. It is possible to see that all sets of nodes resulted in a high CV. However, the CV was smaller for the *k*-core sets indicating that they had higher precision than the *ping* sets: the CV was 0.705 and 0.650 for the sets `core-150ms-everyround` and `core-150ms`, respectively, while the sets `ping.c3sl` and `ping.uk` had the CV equal to 0.841 and 0.916, respectively. Furthermore, the average execution times for the *k*-core sets were also smaller than for the *ping* sets.

The execution times of the Bit application were also measured. Fig. 8 shows the results obtained for all 10 series of experiments, in the same way as Fig. 7. The execution times are plotted. It is clear that the values are more dispersed for the *ping* sets than for the *k*-core sets. For instance, the `core-150ms-everyround` set presents some values far from the average in series 5 and 8, while the `ping.uk` set had several values far from the average of all series. It is also possible to notice that the confidence interval is smaller for the *k*-core sets, both for the series averages—green line—and for the global average—red line with gray shadow.

We also evaluated the selected sets of nodes by compiling metrics for the Bit application. Table 4 shows the average execution time for all 300 rounds, in seconds, for each set of nodes, along with the standard deviation, minimum and maximum values, as well as the CV. The CV for the *k*-core sets was about 27% smaller than the CV for the *ping* sets. Observing the average execution times, it is possible to see that the *k*-core sets ran the application faster than the *ping* sets.

The execution times for all experiments with the Torrent application were also observed. Fig. 9 shows the results in the same way as Figs. 7 and 8. The execution times are plotted. The *ping* sets results were much more dispersed than the *k*-core sets results which were mostly closer to the average.

We compiled metrics for the Torrent application using the execution times measured for all 300 rounds, as shown in Table 5. The



(a) core-150ms-everyround (b) core-150ms

(c) ping.c3sl (d) ping.uk

Fig. 8. Execution times for the Bit application.

Table 4
Metrics for the Bit application, in seconds.

Set of nodes	core-150ms-everyround	core-150ms	ping.c3sl	ping.uk
Average	196.86	195.02	236.39	202.82
Standard deviation	13.21	13.09	22.14	18.86
Minimum	174.89	175.31	206.37	180.95
Maximum	238.54	237.70	338.23	348.51
Coefficient of variation	0.067	0.067	0.093	0.092

Table 5
Metrics for the Torrent application, in seconds.

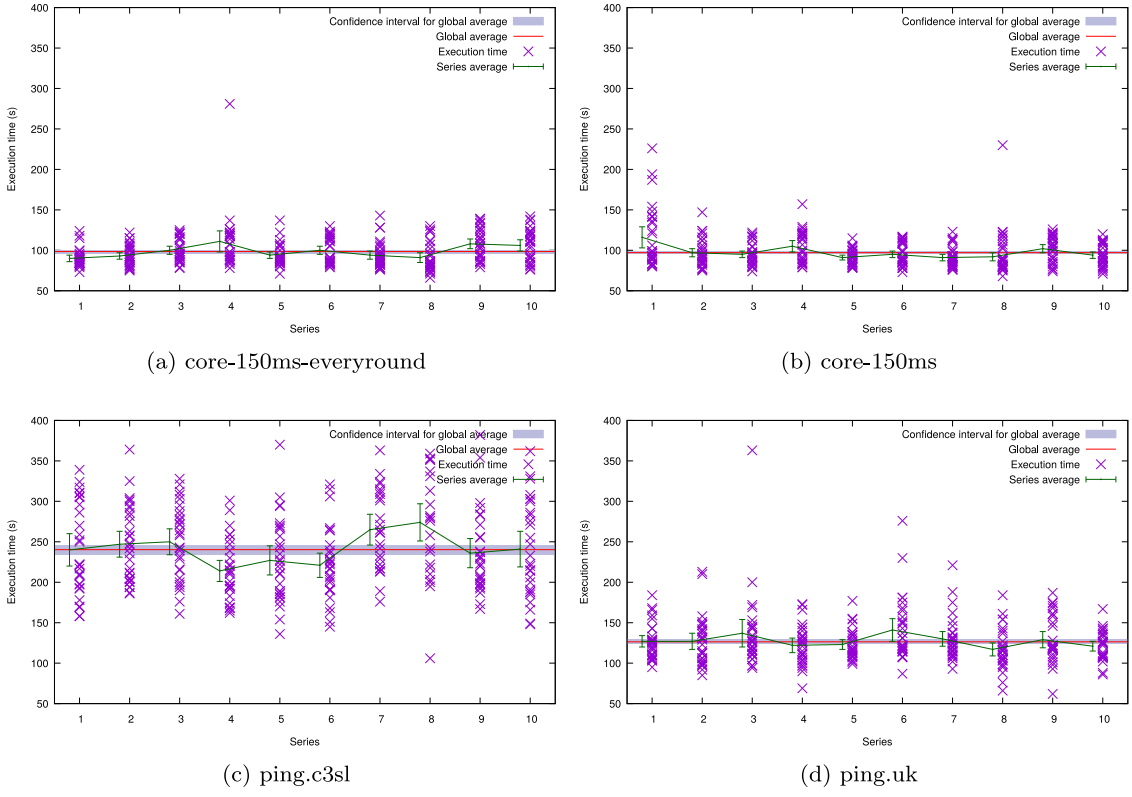
Set of nodes	core-150ms-everyround	core-150ms	ping.c3sl	ping.uk
Average	98.46	97.09	240.25	126.37
Standard deviation	15.84	15.07	47.67	21.63
Minimum	75.83	75.93	158.77	88.06
Maximum	137.97	142.30	354.43	188.35
Coefficient of variation	0.161	0.155	0.198	0.171

CV for the *k*-core sets was smaller than the CV for the *ping* sets. The average execution time was also smaller for the *k*-core sets. The `ping.c3sl` set had a much larger execution time in comparison to the other three, 240.25 s.

The *k*-core sets presented a smaller CV for all three applications when compared to the *ping* sets. The CV for the *k*-core sets were up to 29% smaller for the Triangular application, up to 28% smaller for the Bit application, and up to 21% smaller for the Torrent application. Also, the average execution time for the *k*-core sets were also better: on average, the *k*-core sets ran the Triangular application up to 37% faster, the Bit application up to 17% faster, and the Torrent application up to 59% faster.

The results presented in this subsection clearly indicate that our node selection strategies increase the reproducibility of experiments by increasing the precision of the results obtained. Furthermore, the maximum *k*-core strategy was able to select better sets of nodes for all three applications, despite of the different resource requirements regarding CPU usage and network traffic. Another important observation is about the similarity of the sets `ping.uk` and `core-150ms`. As stated before, they were similar to each other: about 100 nodes were present in both of them, thus they had only about 50 different nodes, which is a third of the total size. Nevertheless, the `core-150ms` set presented better results than the `ping.uk` set.

Another conclusion that can be made regarding the smaller CV presented by the *k*-core sets concerns the stability of a *k*-core as time passes and the ability of the maximum *k*-core strategy to select nodes that are faster and more consistent in terms of the range of values observed. The `core-150ms` set, which remained the same during all experiments, presented similar results during all 40 days, i.e. it remained stable during the whole period in which the experiments were executed. On the other hand,



(a) core-150ms-everyround (b) core-150ms

(c) ping.c3sl (d) ping.uk

Fig. 9. Execution times for the Torrent application.

the `core-150ms-everyround` set, which was selected at each round, also obtained similar results in all experiments, i.e. the maximum *k*-core strategy was able to select sets of nodes that presented similar results at each round.

5. Conclusion

In order to improve the performance and reproducibility of experiments executed on global-scale testbeds, we investigated several strategies to select nodes to run experiments that can be considered to be stable according to predefined criteria. These criteria are applied to monitoring data which is used to build stability graphs from which the sets of stable nodes are selected. Five strategies to select nodes in stability graphs were defined, each with a different stability pattern, ranging from cliques to sets of nodes with a minimum degree. The monitoring and node selection strategies were implemented in PlanetLab. We evaluated the proposed strategies and compared with other alternatives in order to check their performance gain and reproducibility. For this comparison, we measured the execution times and the coefficient of variation of different distributed applications, each with different resource requirements, executed both on sets of nodes selected by the defined strategies and by alternative strategies. The experimental results obtained show that nodes selected by our strategies ran the applications significantly faster and with less variation than the nodes selected by the other strategies. In other words, our strategies improved the performance and reproducibility of the experiments executed.

Future work includes the definition of node selection strategies better suited for applications with specific resource requirements (e.g. very low or very high CPU utilization). Selecting *k*-cores in a fully decentralized way is another important research direction: a distributed application can behave differently on each node depending on which *k*-core the node is currently in. Another issue for the future is to extend the classification of stability; using adaptive thresholds seems to be an attractive alternative.

Acknowledgments

This work was partially supported by the Brazilian National Research Council (CNPq), project 309143/2012-8. Thiago Garrett has a Ph.D. scholarship from the Brazilian Education Ministry CAPES. We thank Lauro L. Costa for conducting the execution of the reproducibility experiments.

References

[1] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, PlanetLab: an overlay testbed for broad-coverage services, SIGCOMM Comput. Commun. Rev. 33 (3) (2003).
[2] S. Fdida, T. Friedman, T. Parmentelat, OneLab: an open federated facility for experimentally driven future internet research, New Network Architectures, 297, Springer, Berlin, Heidelberg, 2010.
[3] M. Berman, J.S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, I. Seskar, GENI: a federated testbed for innovative network experiments, Comput. Networks 61 (2014) 5–23.
[4] E. Duarte, T. Garrett, L. Bona, R. Carmo, A. Züge, Finding stable cliques of PlanetLab nodes, in: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2010.
[5] N. Spring, L. Peterson, A. Bavier, V. Pai, Using PlanetLab for network research: myths, realities, and best practices, SIGOPS Operating Syst. Rev. 40 (1) (2006) 17–24.
[6] P.E. Veríssimo, Travelling through wormholes: a new look at distributed systems models, SIGACT News 37 (1) (2006) 66–81.
[7] L. Tang, Y. Chen, F. Li, H. Zhang, J. Li, Empirical study on the evolution of PlanetLab, in: International Conference on Networking, 2007.
[8] K. Park, V.S. Pai, CoMon: a mostly-scalable monitoring system for PlanetLab, SIGOPS Operating Syst. Rev. 40 (1) (2006) 65–74.
[9] J. Londono, A. Bestavros, netEmbed: a network resource mapping service for distributed applications, IEEE International Symposium on Parallel and Distributed Processing, 2008.
[10] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: a decentralized network coordinate system, SIGCOMM Comput. Commun. Rev. 34 (4) (2004) 15–26.
[11] M.L. Massie, B.N. Chun, D.E. Culler, The ganglia distributed monitoring system: design, implementation and experience, Parallel Comput. 30 (7) (2004) 817–840.

[12] J. Liang, S.Y. Ko, I. Gupta, K. Nahrstedt, MON: management overlay networks for distributed systems, ACM Symposium on Operating Systems Principles, ACM, 2005.
[13] J. Albrecht, D. Oppenheimer, A. Vahdat, D.A. Patterson, Design and implementation trade-offs for wide-area resource discovery, ACM Trans. Internet Technol. 8 (4) (2008) 113–124.
[14] H. Ortiz, A. Casimiro, P. Veríssimo, Architecture and implementation of an embedded wormhole, International Symposium on Industrial Embedded Systems, 2007.
[15] S. Gorender, R. Macedo, M. Raynal, An adaptive programming model for fault--tolerant distributed computing, IEEE Trans. Dependable Secure Comput. 4 (1) (2007).
[16] R. De Araujo Macedo, S. Gorender, Perfect failure detection in the partitioned synchronous distributed system model, in: International Conference on Availability, Reliability and Security, 2009.
[17] M. Huang, A. Bavier, L. Peterson, PlanetFlow: maintaining accountability for network services, SIGOPS Operating Syst. Rev. 40 (1) (2006) 89–94.
[18] Y. Fu, Y. Wang, E. Biersack, A general scalable and accurate decentralized level monitoring method for large-scale dynamic service provision in hybrid clouds, Future Gener. Comput. Syst. 29 (5) (2013) 1235–1253.
[19] Y. Liao, W. Du, P. Geurts, G. Leduc, Decentralized prediction of end-to-end network performance classes, in: International Conference on Emerging Networking Experiments and Technologies, ACM, 2011.
[20] V. Lo, D. Zhou, Y. Liu, C. GauthierDickey, J. Li, Scalable supernode selection in peer-to-peer overlay networks, International Workshop on Hot Topics in Peer–to-Peer Systems, 2005.
[21] Y. Wang, A. Carzaniga, A.L. Wolf, Four enhancements to automated distributed system experimentation methods, in: International Conference on Software Engineering, ACM, 2008.
[22] T. Buchert, C. Ruiz, L. Nussbaum, O. Richard, A survey of general-purpose experiment management tools for distributed systems, Future Generation Comput. Syst. 45 (C) (2015) 1–12.
[23] S. Edwards, X. Liu, N. Riga, Creating repeatable computer science and networking experiments on shared, public testbeds, SIGOPS Operating Syst. Rev. 49 (1) (2015) 90–99.
[24] M. Santos, S. Fernandes, C. Kamienski, Conducting network research in large-scale platforms: avoiding pitfalls in PlanetLab, in: 2014 IEEE 28th International Conference on Advanced Information Networking and Applications, 2014, pp. 525–532.
[25] S.B. Seidman, Network structure and minimum degree, Social Networks 5 (3) (1983) 269–287.
[26] S.N. Dorogovtsev, A.V. Goltsev, J.F.F. Mendes, *k*-core organization of complex networks, Phys. Rev. Lett. 96 (2006) 040601.
[27] S.P. Borgatti, M.G. Everett, Models of core/periphery structures, Social Networks 21 (4) (2000) 375–395.
[28] P. Holme, Core-periphery organization of complex networks, Phys. Rev. E 72 (2005) 046111.
[29] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, E. Shir, A model of internet topology using k-shell decomposition, Proc. Natl. Acad. Sci. 104 (27) (2007) 11150–11154.
[30] Z. He, Y.H. Cao, X.H. Huang, Y. Ma, Overlay node deployment based on node coreness, J. China Univ. Posts Telecommun. 17 (Suppl. 2) (2010) 99–103.
[31] M. Kitsak, L.K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H.E. Stanley, H.A. Makse, Identification of influential spreaders in complex networks, Nat. Phys. 6 (2010) 888–893.
[32] J.-G. Liu, Z.-M. Ren, Q. Guo, Ranking the spreading influence in complex networks, Physica A: Stat. Mech. Its Appl. 392 (18) (2013) 4154–4159.
[33] D. Mills, Internet time synchronization: the network time protocol, IEEE Trans. Commun. 39 (10) (1991) 1482–1493.
[34] L.C.E. Bona, E.P. Duarte Jr., T. Garrett, Monitoring pairwise interactions to discover stable wormholes in highly unstable networks, in: International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 44, Springer, Berlin, Heidelberg, 2012.
[35] R. Diestel, Graph Theory, Springer, 2006.
[36] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., 1990.
[37] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments, Theor. Comput. Sci. 363 (1) (2006) 28–42.
[38] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[39] M. Bhandarkar, MapReduce programming with Apache Hadoop, IEEE International Symposium on Parallel Distributed Processing, 2010.
[40] H. Wainer, Robust statistics: a survey and some prescriptions, J. Educ. Stat. 1 (4) (1976) 285–312.
[41] D. Eastlake 3rd, T. Hansen, US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF), 2011, (RFC 6234).

Thiago Garrett is a Ph.D. student at Federal University of Parana, Curitiba, Brazil, from where he also received the M.Sc. and B.Sc. degrees in Computer Science, in 2011 and 2008, respectively. His research interests include Computer Networks and Distributed Systems, Parallel Computing, and Embedded Systems. He is currently a student member of the Computer Networks and Distributed Systems Lab (LaRSis), Curitiba, Brazil.

Luis C. E. Bona is an Associate Professor at Federal University of Parana, Curitiba, Brazil, where he is a member of the Computer Networks and Distributed Systems Lab (LaRSis). He obtained a Ph.D. degree in Electric Engineering at Federal University of Technology—Parana, 2006, and carried out his post-doctoral studies at the Barcelona Supercomputing Center (BSC), 2013. His research interests include Operating Systems, Computer Networks and Distributed Systems. He acted as coordinator of several research technological development projects, for both national and international. He also served as chair of the Department of Computer Science of Federal University of Parana from 2008 to 2012.

Elias P. Duarte Jr. is a Full Professor at Federal University of Parana, Curitiba, Brazil, where he is the leader of the Computer Networks and Distributed Systems Lab (LaRSis). His research interests include Computer Networks and Distributed Systems, their Dependability Management, and Algorithms. He has published more than 150 peer-reviewer papers and has chaired several conferences and workshops. He received a Ph.D. degree in Computer Science from Tokyo Institute of Technology, Japan, 1997, the M.Sc. degree in Telecommunications from the Polytechnical University of Madrid, Spain, 1991, and the B.Sc. and M.Sc. degrees in Computer Science from Federal University of Minas Gerais, Brazil, 1987 and 1991, respectively. He chaired the Special Interest Group on Fault Tolerant Computing of the Brazilian Computing Society (2005–2007); the Graduate Program in Computer Science of UFPR (2006–2008); and the Brazilian National Laboratory on Computer Networks (from 2012). He is a member of the Brazilian Computer Society and a Senior Member of the IEEE.