Summary Review Documentation for

“Reducing Allocation Failures in Network Testbeds”

Authors: J. Mirkovic, H. Shi, A. Hussain

Reviewer #1

Summary: This paper examines allocation failures in the DeterLab testbed, powered by Emulab. It also examines different scenarios for improving allocation success, ranging from loosening experiment specifications to changing the allocation algorithm.

Strengths: The paper provides some concrete examination of allocation failures in an Emulab-derived testbed, thereby leading to the chance that other Emulab-based environments may benefit from any lessons drawn from this system. The new allocator may also help more broadly.

Weaknesses: Various testbed software bugs that lead to failures are given too much importance, presumably to broaden the analysis of allocation failures. Similarly, a fair bit of space is devoted to relaxing constraints on allocation requirements, without really examining the impact of whether such changes would be productive, especially given the security-oriented nature of their testbed.

Comments to authors: The paper proposes a set of changes to the Emulab system based on understanding the failures of DeterLab, an Emulab-based testbed.

The changes are basically: (a) get rid of bugs that cause resource problems, (b) improve resource assignment, and (c) perform a small amount of sharing

Change (a) is easy to state, but realistically increases the cost of development fairly significantly in the way the authors describe. Yes, it’s bad that there are bugs in the resource allocation mechanism, but it’s kind of a fact of life, and for relatively niche software without a large development budget and a large user community, these things are going to happen.

Change (b) produces moderate improvements – a 20% reduction in allocation failures. In and of itself, it’s a good idea to implement because the reduction of inter-switch bandwidth seems to be significant, and it helps allocations.

Change (c) is something of a biggie. Since long-running jobs currently can’t be pre-empted on Emulab, short-running jobs can fail to get resources. By pre-empting long-running jobs, the allocation failures for short-running jobs can be mitigated. In theory, this is a good idea. However, this also changes one of Emulab’s main assumptions, and brings it closer to a system that allows scheduling of jobs and sharing of the platform. One may argue that a little pre-emption can’t hurt, and this may be true, but it is a fundamental change to the underlying model.

Overall, it’s not an overly controversial paper, but it is somewhat niche, and it’s not clear that options (a) and (c) are going to be realizable. Option (b) is nice, and will produce some improvements.

Reviewer #2

Summary: This paper presents improvements to resource allocation schemes in network testbeds, based on an 8-year dataset of activity in DeterLab as stored in the DB of the controlling node of the testbed (“boss”). The authors recap DeterLab’s design and terminology, describe the dataset (including available node types and causes for identified inconsistencies in the operational state transaction log), and describe resource allocation in DeterLab, called the “testbed mapping problem”.

They define 6 classes of resource allocation failures and present a breakdown of the frequency of failures in each class. 36% occur due to the experimenter’s constraints, 45% due to actual resource shortages.

Next, the authors discuss the selectivity of experimenter-imposed constraints in the dataset, including the percentages of topologies that can execute on fractions of the testbed, with arbitrary OSs, node types etc.

The authors then recap Emulab’s assign algorithm, and improve it to assign+ by guiding searches in the allocation space using 5 specific search strategies. The authors find their approach yields 20% allocation failures while running an order of magnitude faster.

Finally, the authors propose two strategies for dealing with resource heavy-hitters: Take-a-Break, basically forced interruption of long-running experiments, and Borrow-and-Return, a soft version of the former in which the allocator returns resources to the instances it took them from within 4 hours.

Strengths: Resource sharing in a networked environment running multiple experiments is difficult, and the authors have at their disposal a great dataset for improving the area. The authors thoroughly examine Emulab’s assign algorithm and its proposed successor.

Weaknesses: The paper has only limited relevance to network measurement. It studies DeterLab’s particular allocation problems, with DeterLab’s specific resources. What the experiments actually do in the testbed remains opaque, as does their actual network usage, about which the authors essentially only state that committing capacity to multiple experiments at the same time often works well.

Comments to authors: This paper is likely of great use to Emulab operators and also folks deploying experiments in DeterLab/Emulab. However, the underlying resource allocation problems and algorithms seem to apply to any other resource-constrained environment (such as process allocation to CPU cores), so I feel the paper isn’t a great fit for IMC. Having no relation to the material, I found the paper a bit of an abstract read. The suggestions you derive are a great start, but they remain shallow (e.g. “Testbeds
need on-demand queuing mechanisms”). I would find the paper much more interesting if I’d get a feeling of what the experiments are trying to do, in order to understand their sensitivity to reduced resources, interruptions, etc.

Suggestion 1 seems to reflect a basic requirement of a testbed that exists for many years: change. True, model-checking (and simply having a model) may help, but what do you do if the model changes over time? In my experience, this kind of evolution is hard to deal with in a systematic fashion.

In 5, I was thinking that standard provisioning practices as used by ISPs should help, as would observation of an experiment over time, to estimate its resource usage. How long do experiments on DeterLab typically run for?

Where do the 6 categories in Section 6 come from? How do you know they’re exhaustive?

In 6 you say you “removed virtual topologies to remove fixed allocations, because they hurt allocations.” Presumably often experimenters have good reason for picking fixed allocations? If not, you should say so; if so, you should elaborate.

Reviewer #3

Summary: This paper describes a measurement study on network testbed allocation errors, and propose a new allocation algorithm that increases the chances of successful allocation of the testbed resources to researchers.

Strengths: This paper contains a lot of detailed data collection and error classification. It provides a thorough look at what researchers do when allocation error occurs, what can happen if long running jobs are stopped, etc. It proposes a new, non-deterministic, allocation algorithm and few additional tricks/suggestion to ensure successful allocation of testbed resources to users.

Weaknesses: This reads more an experience paper. The presentation is muddled, and the scope feels limited.

Comments to authors: The presentation is muddled and there are too many things going on. First, there is a list of suggestion/best practices for network testbed administrators, then there is a section on studying why testbed failures occur, then there is a section about relaxing user constraint to allow better allocation, then one on an algorithm that improves resource allocation, and finally another set of algorithms on how resource allocation can be used for fair sharing. It is unclear what the focus of the paper is.

Further, most of the sections sound more like an experience paper with either very little new or little that is well evaluated. The “suggestions” section is definitely an experience section.

But even the “relaxing user constraint” section talks about how users should relax their constraints on the resources they want (for example, get an upgraded machine if the requested machine is not available). There is no algorithm or procedure presented, and the users already seem to relax their constraints on their own.

The Assign+ is a newer algorithm that replaces the existing Assign algorithm. Assign+ uses a deterministic search instead of simulated annealing. There is limited discussion on why a deterministic technique will be better than a non-deterministic technique. For example, is it possible to just do the simulated annealing, but with constraint relaxation added to it?

Further, in the results (Figure 6), the authors remove all instances when both Assign and Assign+ could not find a match. This makes it hard to evaluate how well Assign+ is really doing. What is the percentage of the remaining instances?

Many of the techniques: Take-a-Break, Borrow-and-return, relaxing-user-constraint, all do not take into account (or have any way of taking into account) what the experiment really needs. For example, the Take-a-break system breaks long running experiments that may holding a resource. This can be dangerously disruptive and may cause more harm. Similarly, automatically assigning resources by relaxing user constraint can be disruptive if it does not take into account experiment requirements.

This paper would have been more interesting if it provided a set of APIs/Programming tools that will let experimenters specify a fine-grained constraint matrix, to automatically allow the resource engine to allocate resources within the constraint.

This work is also limited in scope. It provides a set of suggestions to improve the network testbed allocation of a specific kind of research platform (emulab).

Reviewer #4

Summary: This paper analyzes logs of allocation events from the DETER testbed to identify potential causes of resource allocation failures for Emulab-style experiments and suggests some improvements and user recommendations to improve the overall allocation rate.

Strengths: This is a very cool dataset and one of the first studies of this kind. This is a problem that many networking researchers can relate to and will find a lot of immediate impact if the findings are valid.

Weaknesses: The paper could be better aligned with the measurement venue by presenting more in-depth analysis of the workloads rather than only focus on the strategies for improvement.

There is a rich literature in cluster/datacenter scheduling that is missing in the related work in the proposed strategies (e.g., borrow, take-a-break).

I also have a meta-concern whether the proposed measures are practical given the nature of the testbed workload – live migration, saving swap state, or relaunching on a different set of nodes.

Comments to authors: As someone who has experienced first hand the troubles of getting adequate nodes on such testbeds close to conference deadlines and thus having to manually try out different topology configurations and often sacrificing the intended scale of the experiments, I see tremendous value in this study and I do hope that your recommendations can be validated by other testbed data and implemented in practice! Thanks for this valuable undertaking!

I have mostly minor clarification questions aside from the three perceived weaknesses above:

Section 3 – how typical is it for a specific experiment to be modified? My own experience is that I often create a new experiment name for a new setup rather than modify an existing one. (cost of creating a new exp is zero and this might be easier for management?)
Section 4 – the recommendation at the end of this section is pretty generic and applies to any software, is there anything that is testbed specific you want to emphasize here?

Section 5 – how typical is it for experiments to specify “fixed mappings” – again drawing on my own experience we usually fix the node type not the specific physical node

Also, is the pcvm unique to DETER, I do not recall seeing this in the main Emulab at Utah

Section 6 – this is a nice decomposition of the failure scenarios, but I did initially find (6) to be a bit odd (it became clearer later on when you mentioned that the assign algorithm uses simulated annealing)

It might be useful to expand on this section to also analyze when the failures occur – e.g., are they all clustered during deadlines which is likely a reflection of oversubscription or are they pretty random? Also, in this section I didn’t get a sense of what fraction of total requests end up failing – is it significant in the common case?

Section 7 – this note about the users flexibility is a bit odd – “322 cases a user has reduced the topology size” – this may be an undesirable outcome where papers get submitted with less than ideal experiments! it may not reflect true flexibility. I also find the heavy-tail with year-long values very very suspicious; this suggests some bug in your cleaning and correlating scripts.

Section 8.2 – I found 8.2 a bit hard to follow – at no earlier point did you mention that bandwidth becomes a bottleneck, yet your optimization via partitioning is focusing on the bandwidth. Second, its not clear what exactly is the partitioning that your heuristic is achieving – is it actually partitioning the graph or is the only important outcome of the algorithm making the graph “directed” to indicate parent-child relationship for future steps?

Section 8.5 – the mixed use of absolute and relative improvements here was a bit hard to follow. maybe it would be better to explicitly mention whether the % you are quoting is absolute or relative?

Also, in terms of positioning the work, I felt that it might be better to introduce the more generic mechanisms like migration, queuing etc before the specific algorithmic improvements (or justify why you chose this order of presentation)

Section 9 – the CSFQ reference here seems a bit strange and out of place. You should probably discuss the literature on cluster and data center scheduling here – e.g., DRF, Mesos from NSDI or the work from the Condor cluster management software? On a related note, I am sure that many of these strategies have been tried or implemented in the cluster scheduling world; it might be useful for you to either draw parallels or contrasts depending on how different the workload/recommendations are.

Reviewers:

**Strengths:** I liked this paper. A careful study of testbed practices is important for our community that has become very dependent upon them as a key mechanism of conducting research. The results herein could be broadly useful because they focus on Emulab software (which is broader than the DeterLab testbed). Each section ends with a suggestion for better practices. Revisiting a multiyear history of failed or blocked experiments is pretty neat. The paper is also very well written and thorough.

**Weaknesses:** There were a few places where I felt I would have liked to see more details on how a particular statistic was computed. But this paper has a fair bit of content as is, so I accept the space constraint. Perhaps the authors could publish a tech report with the extra details.

**Comments to authors:** Your paper is very well written. I liked the way you used the example in Figure 3 to illustrate a whole bunch of different things. I would have liked to know more details about how the calculations in section 6 were done. What exactly is in each “testbed state snapshot”? Is a snapshot generated every time a state transition occurs, or is there some aggregation that occurs? In step 3 on page 6, you use the word “infer” a lot. I don’t think you are referring to data mining here, but simply a process of elimination referring to data mining here, but simply a process of elimination based upon the logs. Is that right? You say you “infer” which physical nodes were available on the testbed at the time of the TEMP error. Is this method 100% accurate? Is there a notion of a false positive here, or not?

Also on page 6, column 2, you conclude that 80% of the TEMP failures could have been avoided. In order to arrive at this number, did you actually rerun an assignment algorithm on the “free” resources you discovered in the logs? Or is this 80% coming from the tests you did in the rest of the paper with new proposed alternate methods? If so, you should make a forward reference here, because otherwise one wonders (after only having read 6 pages) how you can claim that 80% of requests COULD HAVE BEEN satisfied.

The authors are quite thoughtful to the needs of the research community, and their proposed fairness attempts to balance the need for small jobs to get resources while not penalizing big jobs. Nice touch.

**Response from the Authors**

We thank reviewers for their comments, which helped us clarify and improve our paper. In the “Changing Allocation Policy” section we indeed propose a radical change to testbed policies, that also require sophisticated state-saving, migration and idleness detection mechanisms that are not available today. We believe that, as testbed popularity grows but their physical growth remains limited by financial and hosting aspects, policy changes are necessary to meet user needs such as differing priorities, differing use models and fairness. Our changes are very gentle, minimizing impact on users but improving their chances of allocation success. Future research in this direction is needed.

We have further clarified the ideas behind the design of assign+ that contribute to its good performance. These can be summa-
rized as “deterministic search”, “allocation of groups of nodes in one pass”, “expert knowledge of testbed architectures used to minimize interswitch bandwidth use” and “separate optimization of interswitch bandwidth use and unwanted features”. These both improve allocation success and shorten the runtime.