A better way to negotiate for testbed resources

Qin Yin, Timothy Roscoe
Systems Group, Department of Computer Science, ETH Zürich
{qyin, troscoe}@inf.ethz.ch

ABSTRACT

Resource allocation is an increasing challenge for distributed network testbeds as computational and network resources are involved. Testbed designers have moved to a query-based model: clients provide a declarative description of their desired resources, and the provider allocate specific resources to meet the request. In this paper, we describe an new approach to negotiate testbed resources between clients and testbed providers: the clients specify their requests as constraints, and the providers reply with resource allocations expressed also as declarative set of constraints on resources. This gives providers more flexibility in late-binding of resources to requests, and opens up a wide design space to optimize resource allocation for efficiency, cost, utilization, or other metrics. Our simple first experiments suggest that the late-binding of resources enabled by representing resource reservation as constraints achieves better network resource utilization compared to the fixed assignment solution.

1. INTRODUCTION

Networking testbeds like GENI [7] face an increasing challenge in resource allocation. As dependencies between resources (switch ports, links, virtual machines) become more constrained, resources become more diverse (specialized switches, programmable middleboxes, etc.) and testbeds scale to large numbers of clients, sites, and network elements, it becomes harder for clients to express their requests to the providers of the testbed infrastructure, and in turn for these providers to allocate resources in a way that makes efficient use of the platform.

Faced with these trends designers of testbed platforms have moved to a query-based model for resource request: clients supply a declarative description of resources they want, and the provider allocates (if possible) specific resources to satisfy this request. The request is a set of constraints on the resources to be allocated, and (optionally) some objective function to allow the resource provider to select the "best" (for the client) allocation from the available options.

In this paper, we take the idea a step further: not only do clients specify their requests as constraints, but providers reply with resource promises which are *also* expressed as sets of declarative constraints on resources. We conjecture that this allows providers much greater flexibility in late-binding of resources to requests, and opens up a wide space of techniques for optimizing testbed utilization, including statistical overbooking.

Similar challenges are faced by Infrastructure-as-a-Service (IaaS) providers and scientific Grid platforms, to a lesser, though increasing, degree. We focus on network research platforms in this paper, but we expect that as network topology becomes increasingly important for performance and as a market differentiator [10] in these areas, our ideas will have wider applicability.

In the next section we describe and motivate the specific resource allocation problem we are addressing in more detail, and discuss the background – in particular, why it has not been a problem until recently, and why we think it will become more important in the future.

In section 3 we introduce the idea of expressing resource *advertisements* and *allocations*, as well as simply requests, as declarative sets of constraints. We show this changes the resource negotiation process between clients and providers, and in turn opens up a rich design space for providers to optimize resource allocation for efficiency, cost, utilization, revenue, or other metrics.

We then describe the current status of our work to establish the effectiveness of the idea and show some preliminary results in Section 4.

2. BACKGROUND AND MOTIVATION

The basic problem we address in this paper is how a client of an infrastructure provider (e.g. a networking testbed, cloud hosting service, or grid installation) requests resources, how the provider allocates such resources, and how the allocation of such resources is returned to the client. "Resources" in this sense include virtual machines, virtual switches and routers, shares of physical network links, and the like.

At a high level, this process is quite straightforward, and existing testbeds employ simple mechanisms. In this section, we survey how it is done today, and in doing so make the argument that existing solutions to the problem will not suffice for large-scale, distributed facilities where networking resources are explicitly allocated.

2.1 Virtual machines

We start with systems that purely allocate virtual machines. Amazon EC2 [5] advertises a small (6 at time of writing) set of VM types (based on location and approximate computing power), each of which consists of a large homogeneous pool. While clients can request VMs in a specific location, EC2's scale means that these

APSys'11, July 11–12, 2011, Shanghai, China.

VMs can be allocated entirely independently of each other. Access to the VMs is granted at the time when a request is made – there is no notion of (or need for) future reservations of resources. These factors allow a simple and intuitive API to EC2 and simplify Amazon's task of provisioning physical plant.

PlanetLab [16] is different: clients specify precisely the physical machine on which to create a VM, and all individual physical resources (close to 1000 servers) are visible to clients. Nevertheless, acquiring resources on PlanetLab is conceptually similar to the EC2 case: a request is expressed in terms of the advertised set of machines, and access to resulting "slivers" is granted in the reply. As a best-effort, community-run service, PlanetLab does not need to provide any resource guarantees or make provisioning decisions. Despite this, the diversity of resources offered by PlanetLab (albeit all resembling Linux virtual machines) has led to third-party resource managers [15, 20] which allow clients to specify requests for resources in the form of declarative queries over the available nodes.

This mirrors constraint-matching in Grid computing systems: Condor [4] allocates machines to jobs based on a match-making mechanism called ClassAds. Machine characteristics and job requirements are represented in a common framework making it possible to decide, for a particular request-resource pair, whether requirements are satisfied. RedLine [11] uses constraints to describe resource offerings and requests, interprets resource matching problem as a constraint satisfaction problem and explores constraint-solving technologies to implement matching operations.

Grid systems require more complex resource allocation schemes for two reasons: firstly, they typically allocate a large number of machines or VMs *in a single operation* for parallel compute jobs, rather than the piecemeal sliver allocation which suffices for deploying overlays on PlanetLab. Secondly, the resource requirements of Grid jobs motivate *advance reservation* of a block of machines. For example, the Haizea [19] lease manager is an OpenNebula [14] scheduling module which leases VM resources under a variety of terms, including reservations and queuing of best effort requests. The key observation is that the *dependencies between resources* motivate a richer specification for resource requests.

2.2 Network virtualization

Such inter-dependencies become more significant when network, as well as computational, resources are involved. Topological considerations tightly constrain resources: virtual machines and virtual switches must be connected by shares of physical links, for example. This introduces two challenges: firstly, how to express requests for combinations of network and compute resources, and secondly, how to efficiently allocate such resources in the provider.

Emulab [6] solves the former by using `ns2` configurations to express virtual networks, which are then embedded into its physical topology using simulated annealing graph mapping algorithms. It successfully avoids the latter challenge through its focus on centrally-controlled network emulation: Emulab exploits high-capacity network switches which approximate a physical crossbar between machines, thereby rendering the problem of embedding clients' virtual networks in Emulab's physical infrastructure tractable.

However, recent proposals for distributed network testbeds such as GENI will not be amenable to such solutions, since they presume a federated, distributed physical infrastructure over which virtual networks ("slices") are instantiated. The expected low cross-sectional bandwidth in GENI-like testbeds leads to inefficient allocation with simple greedy approaches.

A simple example (Figure 1) should make this clear. Given a physical network of 2 switches and 8 hosts, a simple strategy might allocate for `REQUEST1` 4 hosts under one switch and leave `REQUEST2` unsatisfiable. In fact, both requests can be met with the allocation solution shown on the right.

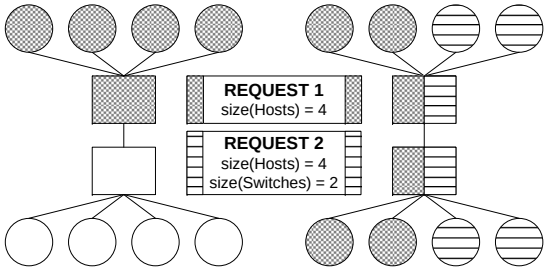

Figure 1. Virtual network allocation example

ProtoGENI [17] uses an optimistic reservation mechanism: the users make the resource availability query right before running the slice embedder, then attempt to get tickets for the components that were chosen.

The ORCA-BEN project [3] uses an ontology language called NDL-OWL to express rich resource requests as queries, including topology embedding. Like some other platforms, ORCA differentiates *tickets* from *leases*. The application requests *u* slivers of resource type *r*. If approved, a ticket is issued for *u* units from a specific pool, and the client later redeems the ticket to obtain a lease for the resources.

Such concepts are not totally new: open signaling [2] and systems like Tempest [13] and the Genesis Kernel [8] explored how to virtualize networks efficiently to support multiple, custom control planes over a single physical infrastructure. More recently, OpenFlow [12] has applied similar ideas to IP- and Ethernet-based networks. Based on this, systems like FlowVisor [18] act as transparent proxies between OpenFlow switches and controllers to provide slices of a physical network. In Internet architecture research, so-called "pluralists" even view support for co-existing virtual networks as the key feature of the architecture [1].

Recently, problem has grown beyond academic interest. Topology-Aware Resource Allocation [10] in IaaS-based cloud systems aims to better support data-intensive workloads by making providers more aware of hosted application requirements and giving users fine-grained visibility into, or control over, the infrastructure.

3. A (POSSIBLY) BETTER WAY

We are building a resource allocator for testbeds to investigate a different approach: we use constraints to describe not only resource requests, but also resource promises made by the provider.

A request is a list of constraints on:

- when the request should be satisfied (start, end time)
- types of computational and networking resources: characteristics of these resources (CPU, memory of the compute units, table entries of the switches, latency or bandwidth of the links) as well as aggregated properties of the resource set;
- connectivity: topological properties of the virtual network, maximum fanout, network diameter, etc.

A very simple example request may look like this:

```
Time :: 8..12,       % request from 8am for 4 hours
size(Hosts) = 3      % three hosts
sum(Hosts, cpu) > 8 % total CPU units larger than 8
size(Switches) >1    % more than one switch
di(Topology) = 3     % network diameter is 3
```

We retain the distinction between tickets and leases: a lease grants access to specific, named resource components, whereas a ticket simply describes (in more or less detail) a set of resources which the client may gain a lease to in the future.

In previous systems, a ticket, signed by the provider, serves as a reservation of a set of concrete resources – a promise by the provider to bind the resources to the client and grant access to them in the future.

In contrast with those systems, the ticket need not bind a specific set of resources, and indeed need not correspond at all with the request that generated it. Instead, the ticket is simply another set of constraints. These might refer to specific switches or nodes, but are more likely to be "unbound": they simply describe elements in the virtual network are not yet mapped to a physical component.

Only leases will always refer to specific resources. Exchanging a ticket for a lease grants access to specific concrete components over a definite interval of time.

3.1 Client-Provider interaction

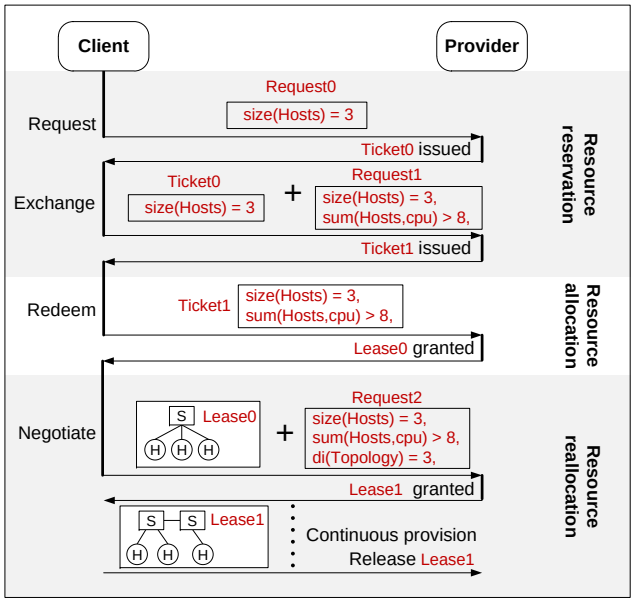

Figure 2. Resource negotiation

Figure 2 depicts how client and provider negotiate testbed resources. This is similar to the protocols used in other systems, but the significant differences are, firstly, that tickets generally do not specify definite resources (or even definite numbers in some cases), and secondly, that resource negotiation is ongoing: a client can at any time return a ticket (or a lease) and exchange it atomically for some other ticket or lease.

In the example, the client sends $Request_0$ for 3 hosts. The provider accepts this request and issues $Ticket_0$ for 3 (non-specific) hosts. Later, the client refines its request, returns $Ticket_0$ and requests in exchange 3 hosts with at least 8 CPU units. This request is again satisfied and $Ticket_1$ returned. In this resource reservation phrase, tickets are "unbound", and the client is not subject to failures of specific nodes. Therefor, the client does not have to frequently check the availability of the reserved "unbound" nodes.

Eventually, the client presents the ticket and requests a lease for the resources. Only then, the provider will allocate specific resources and grant the client a lease after which the client gets the control over the resources.

Even after resource allocation (the ticket is redeemed), the client is still able to negotiate with the provider by sending a new refined request and returning the granted lease. In the example, the client adds an additional constraint on network diameter. For the provider, to minimize the overhead of resource reallocation, it's preferrable to keep as much current allocation possible (two hosts under one switch), release the last host under this switch, and alloate another switch with one host connected to it.

More general operations such as splitting and merging of tickets are also possible, in line with existing testbed proposals.

Of more interest, however, than the protocol itself is the state maintained by both sides and the actions taken on receipt of a message, in particular, in the provider.

The provider maintains an up-to-date list of all physical resources available, and their current condition, together with a list of all valid tickets and leases that it has issued. A provider will typically also have some kind of objective function it will seek to maximize – mostly likely utilization in the case of a networking testbed, but in commercial settings this will probably be some function of yield or revenue.

Conceptually, when it receives a request for a ticket, the provider will try to generate (or, strictly speaking, *prove the existence of*) an assignment of physical resources to a new ticket which optimizes its object function subject to the set of constraints imposed by:

1. the availability and topology of physical resources
2. the set of already-issued tickets and leases, *minus* those being returned in the request

If the request is for a lease, the provider will return this concrete assignment in the lease. This operation may, of course, fail: changes in testbed since the ticket was issued may result in the provider being unable to satisfy the request. However, if the request is for a ticket, the provider has considerable freedom in deciding whether to issue a ticket, and for what.

The straightforward procedure above which takes all allocations into account can give optimal use of the infrastructure (without resorting to revoking leases), and therefore extracts maximum benefit from the fact that tickets do not imply definite assignments. Solving this problem includes embedding many virtual networks into a physical network, and is known to be NP-hard. However, there may be approaches which produce near-optimal solutions cheaply.

Another approach is to assign resources as we go: the provider retains the previous concrete assignment and assigns resources for a new request only from previous unassigned ones. This incremental approach is the behavior of current testbeds.

Between these extremes there is a trade-off: the more existing reservations our system can reconsider, the greater the complexity of allocation but the higher the potential efficiency of the result. We show preliminary investigations of this trade-off in the next section.

Our point is not to identify an optimal algorithm at this stage, but rather to show that there is a wide space of possible solutions. Some may be appropriate for academic testbeds which follow PlanetLab's community model, while others may employ sophisticated yield-management techniques (as in, for example, the travel industry), including the use of overbooking and variable price models, to maximise commercial revenue.

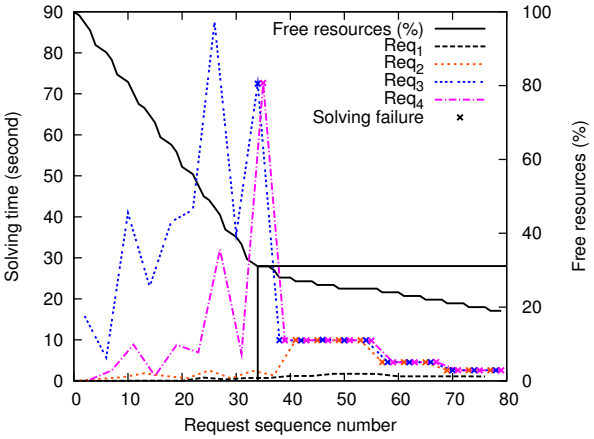

Figure 3. Sequential solving

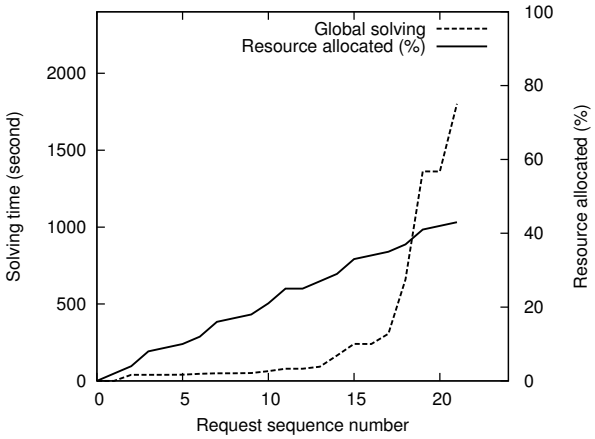

Figure 4. Global solving

3.2 Further advantages

As well as opening up a wide space of allocation strategies, issuing tickets as constraints offers many other potential advantages for network testbeds. Firstly, late-binding of resources accommodates changes in physical resource availability. Failures can be masked if resources can be reallocated before they are required.

Secondly, resources can be specified with different levels of generality. Very general requests such as "4 machines for 2 hours in the morning" are both easy to express and fit naturally into the provider's framework. Furthermore, in the case of network elements with varying capabilities, we avoid committing a machine early to a task which does not fully exercise it. Finally, multi-stage negotiation is possible with constraint-based tickets. The client is able to refine the resource request by adding or modifying constraints.

4. INITIAL RESULTS

We now briefly present some initial results aimed at establishing the feasibility of our approach. In particular, we are interested in the *size of space* for optimization that is opened up by late-binding resource requests. A thorough, real-world evaluation is beyond the scope of this paper and a topic of our ongoing work.

We used Mininet [9] to generate a physical tree network (depth 3, fanout 6) with 216 hosts and 43 switches, and randomly annotated the nodes with different capabilities. Our test workload is a round-robin sequence of 4 pre-defined requests, Req_1 and Req_2 are simple requests for networks of 2 and 5 nodes respectively, while Req_3 and Req_4 are more complex requests for larger networks (7 and 11 nodes) with specific topologies. For different allocation strategies, we are interested both in time to perform successive allocations, and the total proportion of physical resources that can be allocated.

In our first experiment we allocate resources to each request when it arrives, and never reallocates resources, roughly following the behavior of current systems. As Figure 3 shows, after the first 34 requests are satisfied (70% utilization), only small requests can be met. Note also that solving time (whether successful or not) decreases as available resources are reduced.

Second, we try full resource reallocation: as each request arrives, we globally allocate all resources to the complete set of requests so far with no a priori allocations. This problem is NP-complete, and so as Figure 4 shows, execution time increases exponentially and after several hours, our solver fails to allocate even half of the available network.

Finally, we pick a design point between these two: allocate sequentially as in the first experiment case, but when allocation fails retry by remapping, in one go, the current request and all existing requests. These requests are ordered by their complexity, and solved independently as we did in the first experiment. Intuitively, this represents a compromise between the exponential solving time of reconsidering all prior requests altogether, and the severely constrained approach of fixing previous allocations. Remapping the more complex requests first might be expected to result in more freedom to find space for new requests.

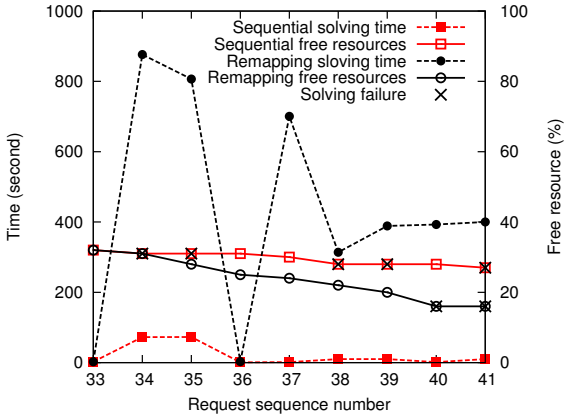

Figure 5. Constraint remapping

Figure 5 shows the results. Versus the simple sequential approach, runtime is much higher (though not prohibitive, even with our unoptimized solver). We satisfy more requests (we now fail first at request 40), resulting in greater utilization (16.7% remaining free).

5. DISCUSSION

There is much scope for improving the run time of our solver for remapping resources (better algorithms such as simulated annealing, a more sophisticated solving engine, etc.). We conjecture that there are also better heuristics for resource remapping than the sim-

ple one we evaluate here: the field of Operations Research has a wealth of results on this kind of problem.

We are also only beginning to explore the possibilities for overbooking and online resource renegotiation permitted by this approach. However, even our simple first experiments suggest that the late-binding of resources enabled by representing resource tickets as constraints opens up a significant space for optimization of resource usage by platform providers.

References

[1] ANDERSON, T., PETERSON, L., SHENKER, S., AND TURNER, J. Overcoming the internet impasse through virtualization. *Computer 38* (April 2005), 34–41.

[2] CAMPBELL, A. T., DE MEER, H. G., KOUNAVIS, M. E., MIKI, K., VICENTE, J. B., AND VILLELA, D. A survey of programmable networks. *SIGCOMM Comput. Commun. Rev. 29* (April 1999), 7–23.

[3] CHASE, J. ORCA control framework architecture and internals. Technical report, Duke University, September 2009.

[4] Condor high throughput computing. http://www.cs.wisc.edu/condor/.

[5] Amazon elastic compute cloud (amazon EC2). http://aws.amazon.com/ec2/.

[6] Emulab - Network Emulation Testbed. http://www.emulab.net/.

[7] Global environment for network innovations (GENI). http://www.geni.net/.

[8] KOUNAVIS, M. E., CAMPBELL, A. T., CHOU, S., MODOUX, F., VICENTE, J., AND ZHUANG, H. The genesis kernel: A programming system for spawning network architectures. *IEEE Journal on Selected Areas in Communications 19* (2001), 511–526.

[9] LANTZ, B., HELLER, B., AND MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks* (New York, NY, USA, 2010), Hotnets '10, ACM, pp. 19:1–19:6.

[10] LEE, G., TOLIA, N., RANGANATHAN, P., AND KATZ, R. H. Topology-aware resource allocation for data-intensive workloads. *SIGCOMM Comput. Commun. Rev. 41* (2011), 120–124.

[11] LIU, C., AND FOSTER, I. A constraint language approach to matchmaking. In *Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)* (Washington, DC, USA, 2004), RIDE '04, IEEE Computer Society, pp. 7–14.

[12] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38* (March 2008), 69–74.

[13] MERWE, J. E. V. D., ROONEY, S., LESLIE, I. M., AND CROSBY, S. A. The tempest - a practical framework for network programmability. *IEEE Network 12* (1997), 20–28.

[14] OpenNebula: The Open Source Toolkit for Cloud Computing. http://opennebula.org/.

[15] OPPENHEIMER, D., ALBRECHT, J., PATTERSON, D., AND VAHDAT, A. Distributed resource discovery on PlanetLab with SWORD. In *WORLDS'04* (Dec. 2004).

[16] PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org/.

[17] ProtoGENI. http://www.protogeni.net/trac/protogeni.

[18] SHERWOOD, R., GIBB, G., YAP, K.-K., APPENZELLER, G., CASADO, M., MCKEOWN, N., AND PARULKAR, G. Can the production network be the testbed? In *Proceedings of the 9th USENIX conference on Operating systems design and implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 1–6.

[19] SOTOMAYOR, B., MONTERO, R. S., LLORENTE, I. M., AND FOSTER, I. Resource leasing and the art of suspending virtual machines. In *Proceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 59–68.

[20] YIN, Q., SCHÜPBACH, A., CAPPOS, J., BAUMANN, A., AND ROSCOE, T. Rhizoma: a runtime for self-deploying, self-managing overlays. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware* (New York, NY, USA, 2009), Middleware '09, Springer-Verlag New York, Inc., pp. 10:1–10:20.