Building Clouds With CloudLab

Robert Ricci

May 2016

THE UNIVERSITY OF UTAH

Clouds have been transformative

Clouds are great!

… except when they're not

Lots of opportunities for research

CloudLab enables research on the future of cloud computing **architectures** and the new **applications** they enable

The Problem with Cloud Research

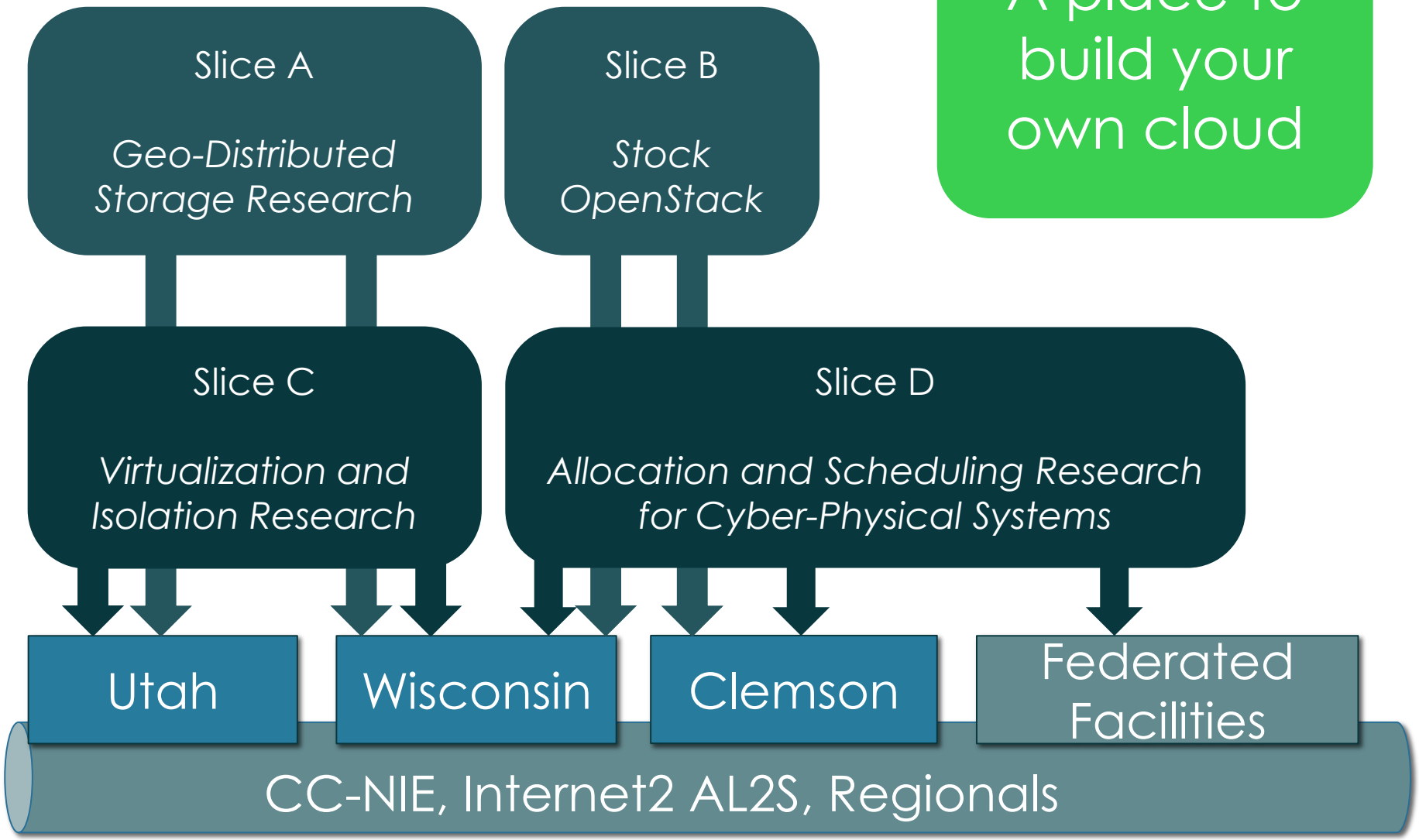
The CloudLab Vision

- A "meta-cloud" for building clouds
- Build your own cloud on our hardware resources
- Agnostic to specific cloud software
  - Run existing cloud software stacks (like OpenStack, Hadoop, etc.)
  - … or new ones built from the ground up
- Control and visibility all the way to the bare metal
- "Sliceable" for multiple, isolated experiments at once

With CloudLab, it's as easy to get *an entire cloud* as it is to get a VM *in* a cloud

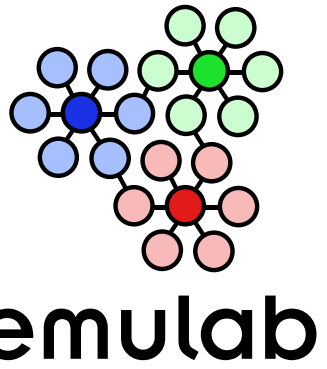
What Is CloudLab?

A place to build your own cloud



Technology Foundations

- Built on Emulab and GENI ("ProtoGENI")
- In active development at Utah since 1999
- Several thousand users (incl. GENI users)
- Provisions, then gets out of the way
  - "Run-time" services are optional
- Controllable through a web interface and GENI APIs
- *Scientific instrument* for *repeatable research*
  - Physical isolation for most resources
  - *Profiles* capture everything needed for experiments
    - Software, data, and hardware details
    - Can be shared and published (eg. in papers)



CloudLab's Hardware

One facility, one account, three locations (+ more!)

- About 5,000 cores each (15,000 total)
- 8-16 cores per node
- Baseline: 4GB RAM / core
- Latest virtualization hardware
- TOR / Core switching design
- 10 Gb to nodes, SDN
- 100 Gb to Internet2 AL2S
- *Partnerships with multiple vendors*

Wisconsin

- **Storage and net.**
- Per node:
  - 128 GB RAM
  - 2x1TB Disk
  - 400 GB SSD
- Clos topology
- *Cisco and HP*

Clemson

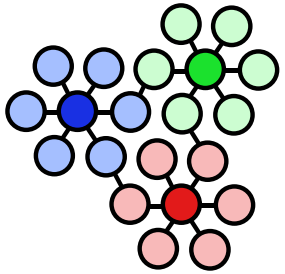
- **High-memory**
- 16 GB RAM / core
- 16 cores / node
- Bulk block store
- Net. up to 40Gb
- High capacity
- *Dell*

Utah

- **Power-efficient**
- ARM64 / x86
- Power monitors
- Flash on ARMs
- Disk on x86
- Very dense
- *HP*

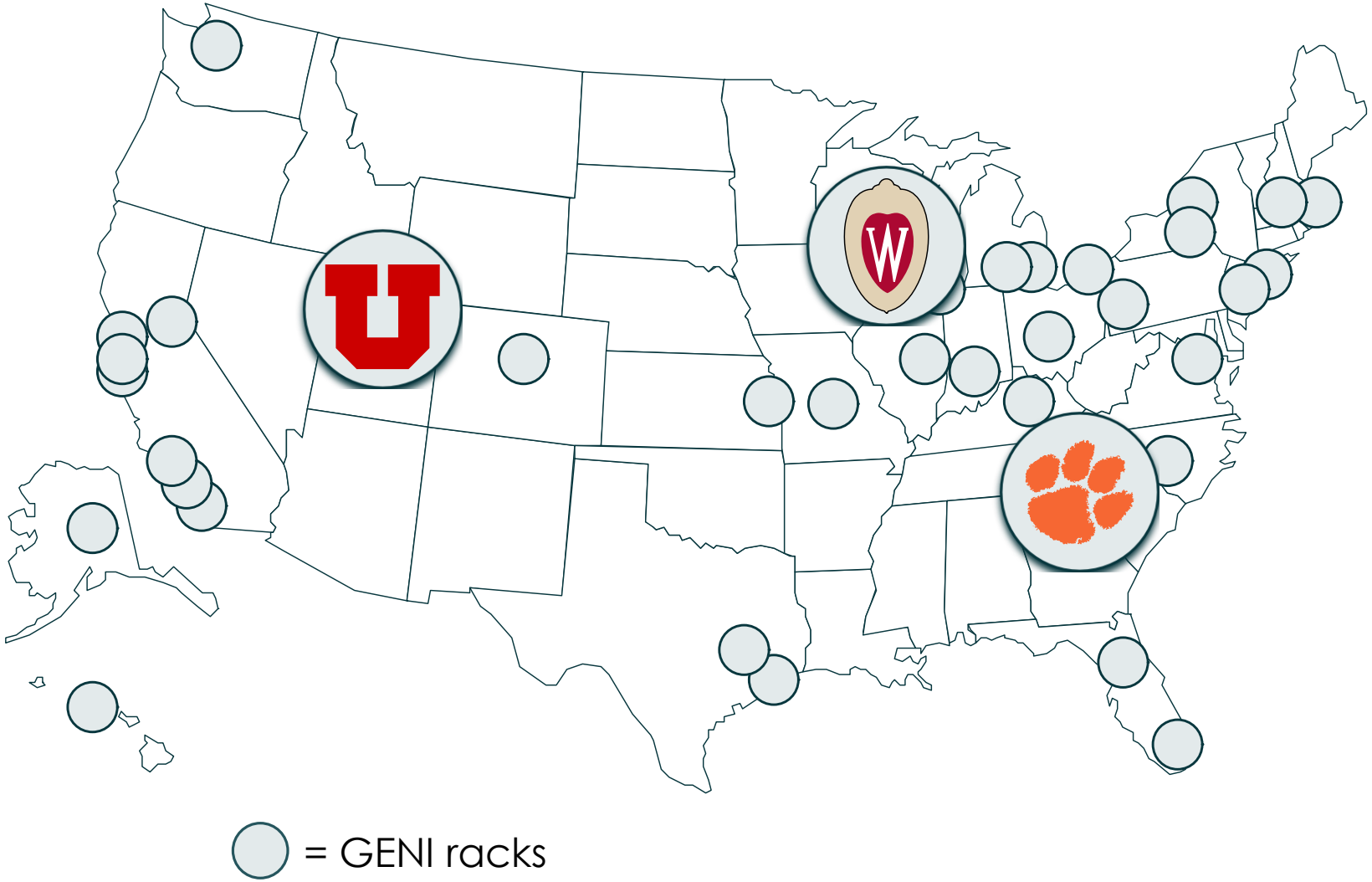
Federated with GENI

- *CloudLab can be used with a GENI account, and vice-versa*
- GENI Racks: ~ 50 small clusters around the country
- Programmable wide-area network
  - Openflow at dozens of sites
  - Connected in one layer 2 domain
- Large clusters (100s of nodes) at several sites
- Wireless and mobile
  - WiMax at 8 institutions
  - LTE / EPC testbed ("PhantomNet") at Utah
- International partners
  - Europe (FIRE), Brazil, Japan

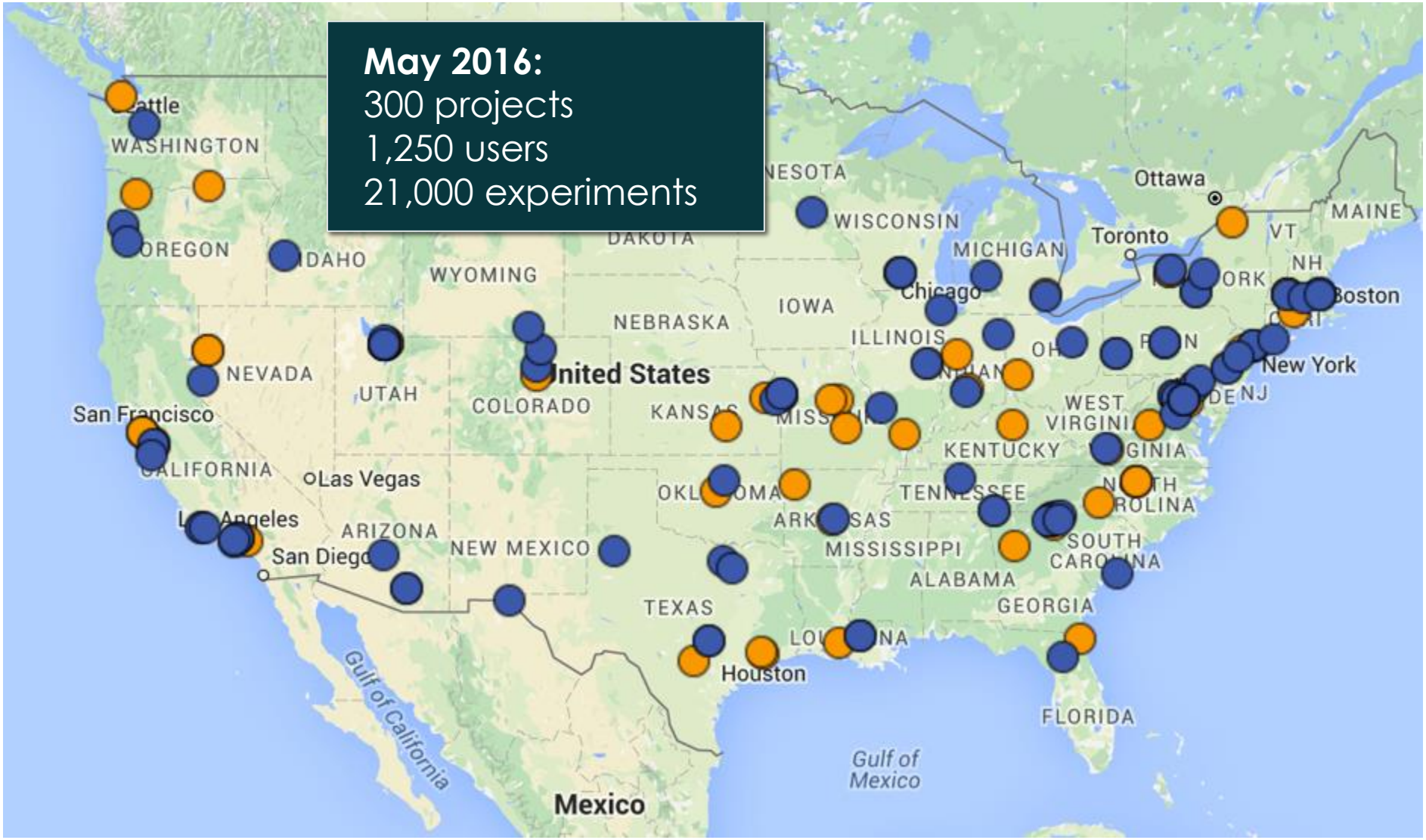


emulab

Many Sites, One Facility



CloudLab Users So Far



Subways: A Case for Redundant, Inexpensive Data Center Edge Links

Vincent Liu, Danyang Zhuo, Simon Peter, Arvind Krishnamurthy, Thomas Anderson
{vincent, danyangz, simpeter, arvind, tom}@cs.washington.edu
University of Washington

ABSTRACT

As network demand increases, data center network operators face a number of challenges including the need to add capacity to the network. Unfortunately, network upgrades can be an expensive proposition, particularly at the edge of the network where most of the network's cost lies.

This paper presents a quantitative study of alternative ways of wiring multiple server links into a data center network. In it, we propose and evaluate Subways, a new approach to wiring servers and Top-of-Rack (ToR) switches that provides an inexpensive incremental upgrade path as well as decreased network congestion, better load balancing, and improved fault tolerance. Our simulation-based results show that Subways significantly improves performance compared to alternative ways of wiring the same number of links and switches together. For example, we show that Subways offers up to 3.1× better performance on a MapReduce shuffle workload compared to an equivalent capacity network.

CCS Concepts

•**Networks** → **Data center networks;**

Keywords

Data center network; Datacenter fabric

1. INTRODUCTION

width of its data center networks by three orders of magnitude between 2004 to 2012, on average doubling every 10 months [32]. Making matters worse, the network is a large and growing portion of the total cost of the data center [17]. Because many data center applications are highly sensitive to tail latencies, networks must be configured with relatively low average link utilization, further increasing costs.

Operators often prefer an incremental approach to adding capacity while the existing network continues to carry traffic [7, 32]. While it is also possible to take down the data center and forklift in a new faster network, this process can require extensive downtime. Instead, adding multiple network links per server has become one way to support upgrades. In principle, a network operator could double capacity by doubling the amount of network hardware, wiring each server in parallel to dual Top-of-Rack (ToR) switches; those switches in turn can be wired in parallel to a replicated aggregation layer, and so forth.

In this paper, we present the counterintuitive result that it is possible to achieve better than a proportional performance improvement when upgrading a data center network for typical workloads. In other words, a doubling of network capacity can result in much better than a 2× performance improvement on the same hardware. A key insight is that nearby servers exhibit communication locality, where physically co-located servers often communicate at the same time,

Paving the Way for NFV: Simplifying Middlebox Modifications using StateAlyzr

Junaid Khalid, Aaron Gember-Jacobson, Roney Michael,
Anubhavnidhi Abhashkumar, Aditya Akella
University of Wisconsin-Madison

Abstract

Important Network Functions Virtualization (NFV) scenarios such as ensuring middlebox fault tolerance or elasticity require redistribution of internal middlebox state. While many useful frameworks exist today for migrating/cloning internal state, they require modifications to middlebox code to identify needed state. This process is tedious and manual, hindering the adoption of such frameworks. We present a framework-independent system, StateAlyzr, that embodies novel algorithms adapted from program analysis to provably and automatically identify all state that must be migrated/cloned to ensure consistent middlebox output in the face of redistribution. We find that StateAlyzr reduces man-hours required for code modification by nearly 20×. We apply StateAlyzr to four open source middleboxes and find its algorithms to be highly precise. We find that a large amount of, but not all, live state matters toward packet processing in these middleboxes. StateAlyzr's algorithms can reduce the amount of state that needs redistribution by 600-8000× compared to naive schemes.

1 Introduction

Network functions virtualization (NFV) promises to offer networks great flexibility in handling middlebox load spikes and failures by helping spin up new virtual instances and dynamically redistributing traffic among instances. The central contribution of this paper is a novel, framework-independent system that greatly reduces the effort involved in making such modifications.

Three factors make such modifications difficult today: (*i*) middlebox software is extremely complex, and the logic to update/create different pieces of state can be intricate; (*ii*) there may be 10s-100s of object types that correspond to state that needs explicit handling; and (*iii*) middleboxes are extremely diverse. Factors *i* and *ii* make it difficult to reason about the completeness or correctness of manual modifications. And, *iii* means manual techniques that apply to one middlebox may not extend to another. Our own experience in modifying middleboxes to work with OpenNF [16] underscores these problems. Making even a simple monitoring appliance (PRADS [6], with 10K LOC) OpenNF-compliant took over 120 man-hours. We had to iterate over multiple code changes and corresponding unit tests to ascertain completeness of our modifications; moreover, the process we used for modifying this middlebox could not be easily adapted to other more complex ones!

These difficulties significantly raise the bar for the adoption of these otherwise immensely useful state handling frameworks. To reduce manual effort and ease adoption, we develop StateAlyzr, a system that relies on *data and control-flow analysis* to automate identification of state objects that need explicit handling. Using State-

High-Performance ACID via Modular Concurrency Control

Chao Xie[1], Chunzhi Su[1], Cody Littley[1],
Lorenzo Alvisi[1], Manos Kapritsos[2] and Yang Wang[3]

[1]The University of Texas at Austin [2]Microsoft Research [3]The Ohio State University

Abstract: This paper describes the design, implementation, and evaluation of Callas, a distributed database system that offers to unmodified, transactional ACID applications the opportunity to achieve a level of performance that can currently only be reached by rewriting all or part of the application in a BASE/NoSQL style. The key to combining performance and ease of programming is to decouple the ACID abstraction—which Callas offers identically for all transactions—from the mechanism used to support it. MCC, the new Modular approach to Concurrency Control at the core of Callas, makes it possible to partition transactions in groups with the guarantee that, as long as the concurrency control mechanism within each group upholds a given isolation property, that property will also hold among transactions in different groups. Because of their limited and specialized scope, these group-specific mechanisms can be customized for concurrency with unprecedented aggressiveness. In our MySQL Cluster-based prototype, Callas yields an 8.2x throughput gain for TPC-C with no programming effort.

1 Introduction

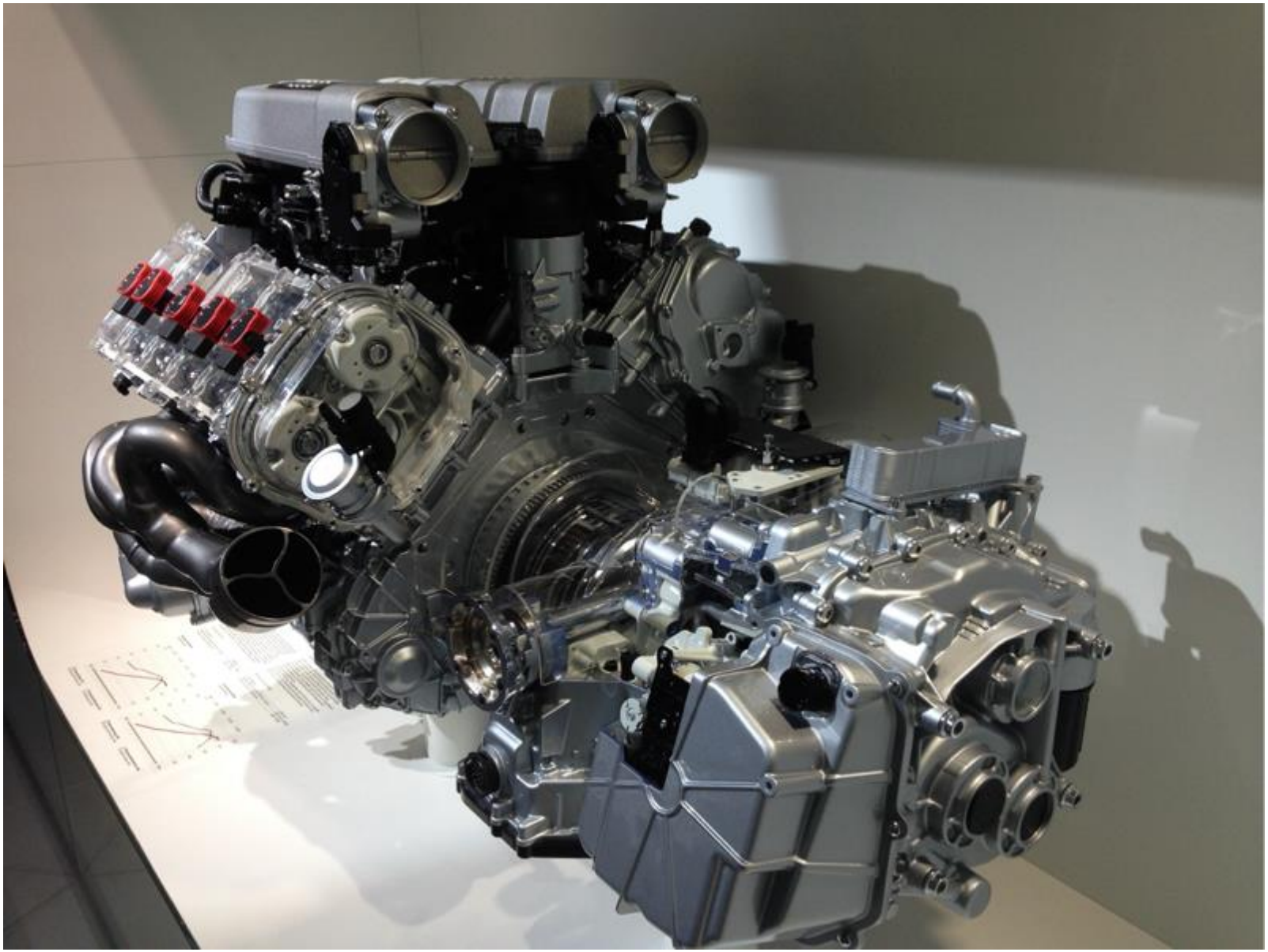
This paper describes the design, implementation, and evaluation of Callas, a distributed database system that offers to unmodified, transactional ACID applications the opportunity to achieve a level of performance that can currently only be reached by rewriting all or part of the application in a BASE/NoSQL style. The key to combining performance and ease of programming is to decouple the ACID abstraction—which Callas offers identically for all transactions—from the mechanism used to support it. MCC, the new Modular approach to Concurrency Control at the core of Callas, makes it possible to partition transactions in groups with the guarantee that, as long as the concurrency control mechanism within each group upholds a given isolation property, that property will also hold among transactions in different groups. Because of their limited and specialized scope, these group-specific mechanisms can be customized for concurrency with unprecedented aggressiveness. In our MySQL Cluster-based prototype, Callas yields an 8.2x throughput gain for TPC-C with no programming effort.

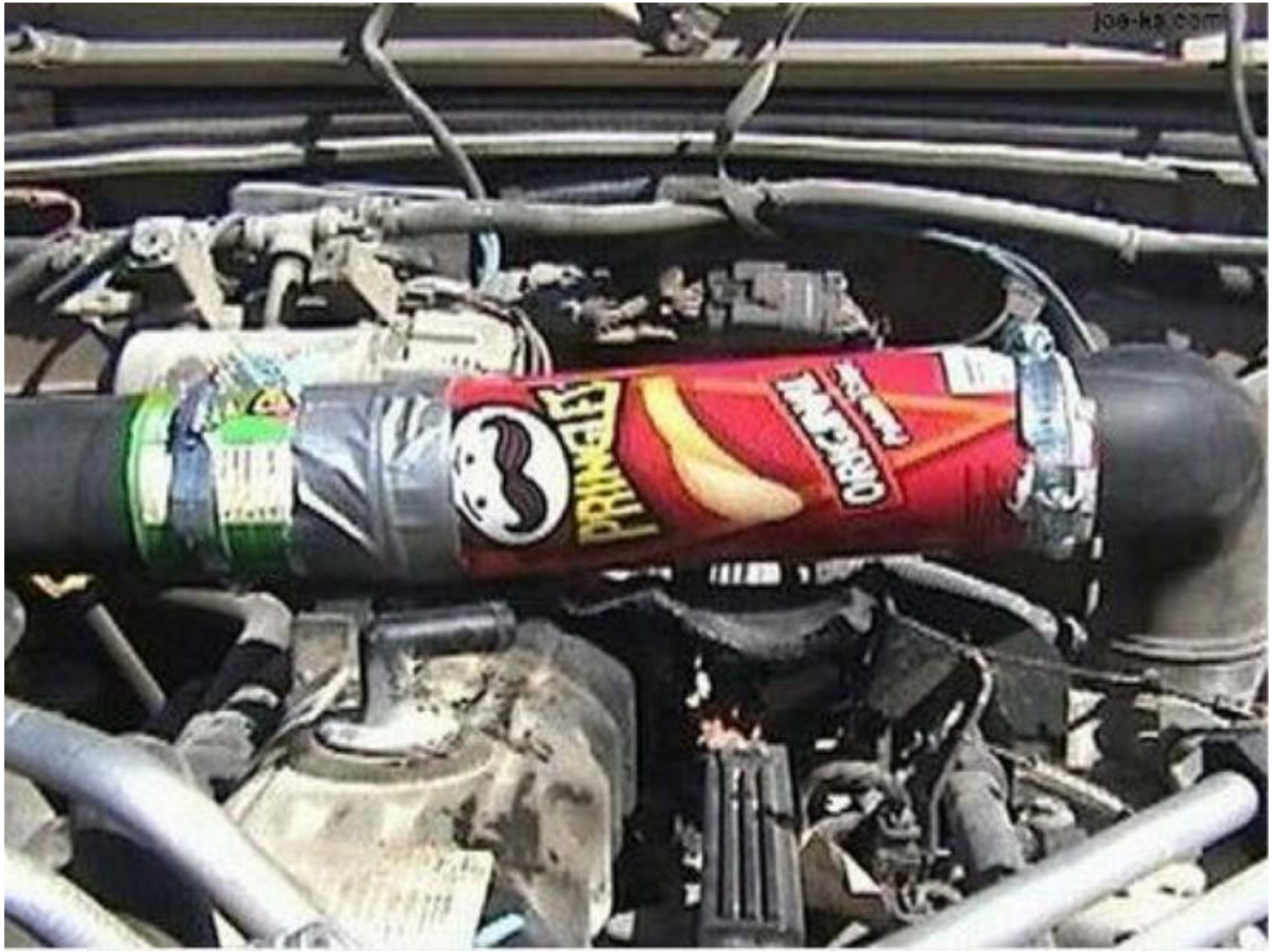
The key observation that motivates the architecture of Callas is simple. While ease of programming requests that ACID properties hold uniformly across all transactions, when it comes to the mechanisms used to enforce these properties, uniformity can actually hinder performance: a concurrency control mechanism that must work correctly for *all* possible pairs of transactions will necessarily have to make conservative assumptions, passing up opportunities for optimization.

Callas then decouples the concerns of abstraction and implementation: it offers ACID guarantees uniformly to all transactions, but uses a novel technique, *modular concurrency control* (MCC), to customize the mechanism through which these guarantees are provided.

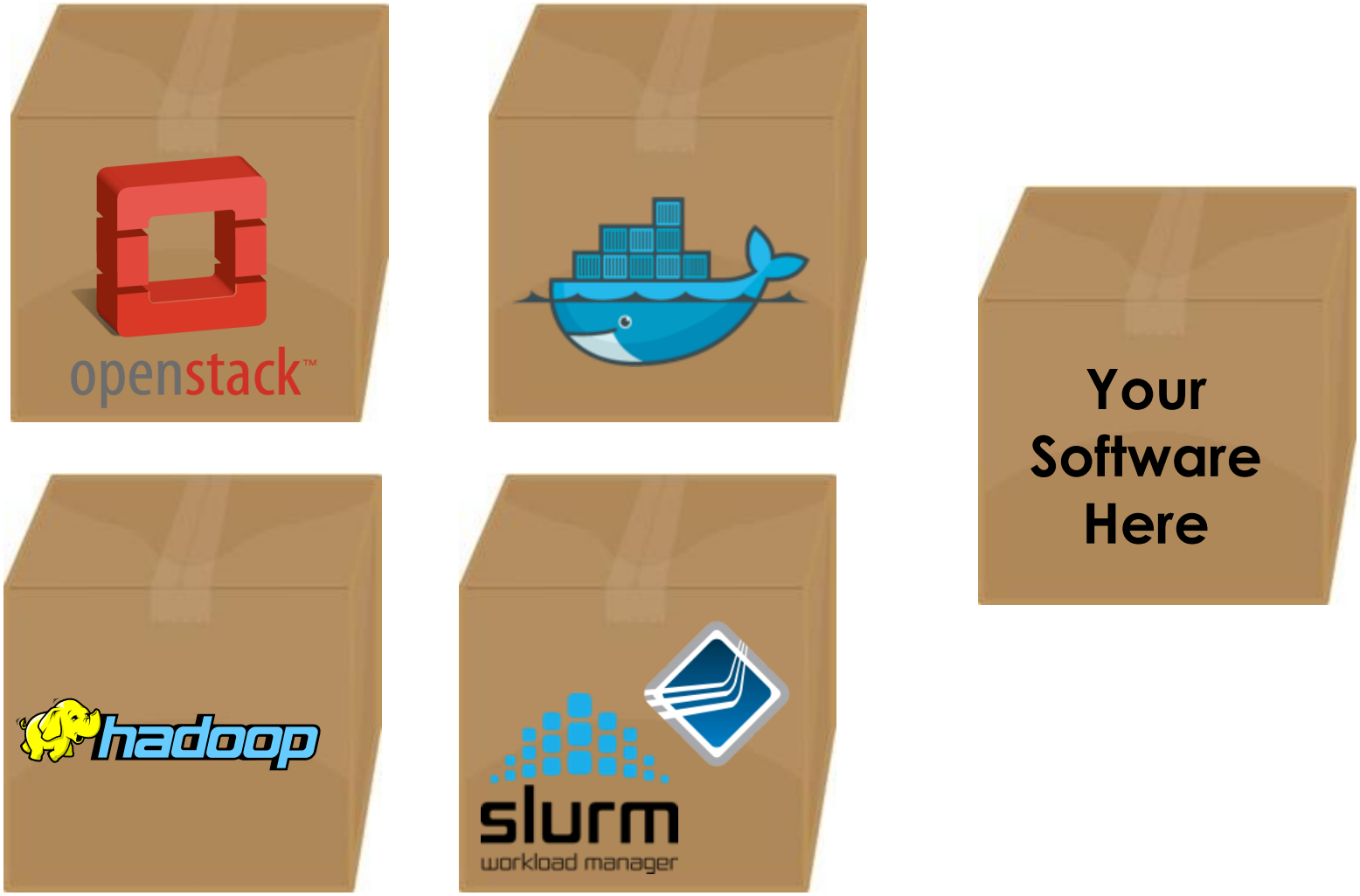
MCC makes it possible to think modularly about the enforcement of any given isolation property I. It enables Callas to partition transactions in separate groups, and it ensures that as long as I holds within each group, it will also hold among transactions in different groups. Separating concerns frees Callas to use within each group concurrency control mechanisms optimized for that group's transactions. Thus, Callas can find opportunities for increased concurrency where

Building on Each Others' Work

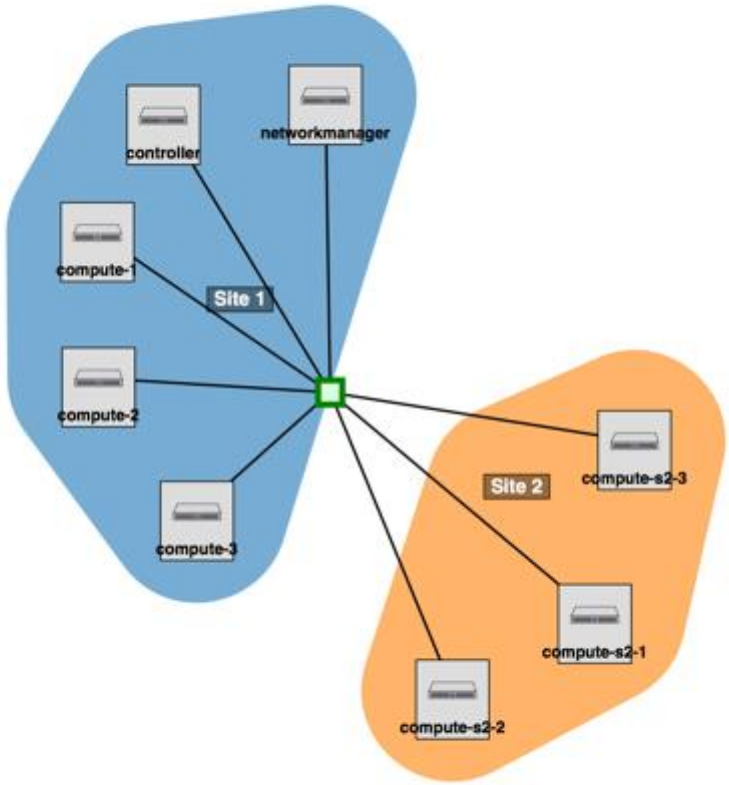



joe-ks.com
PRIMER
PAINT

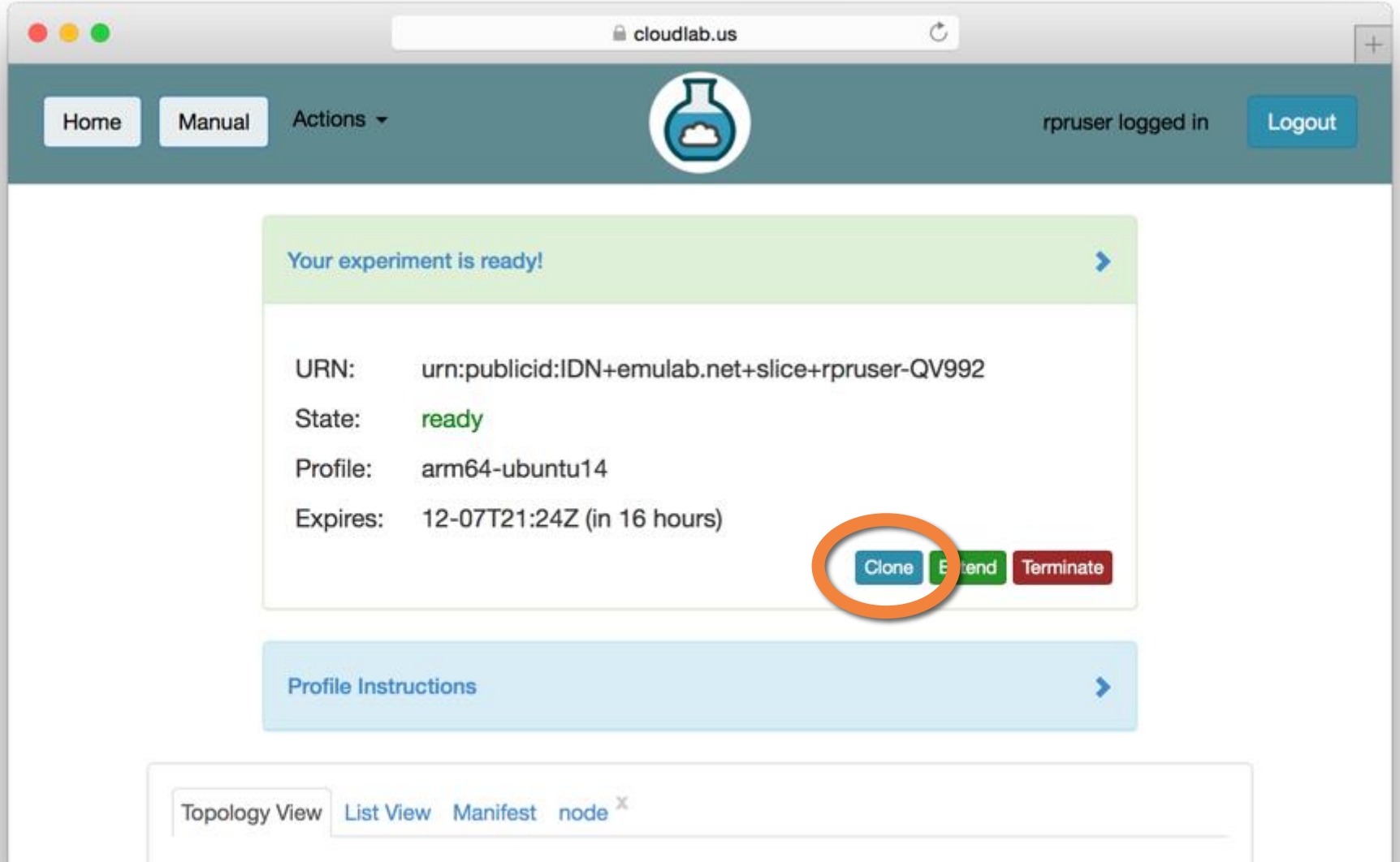
Profiles: Packaged environments



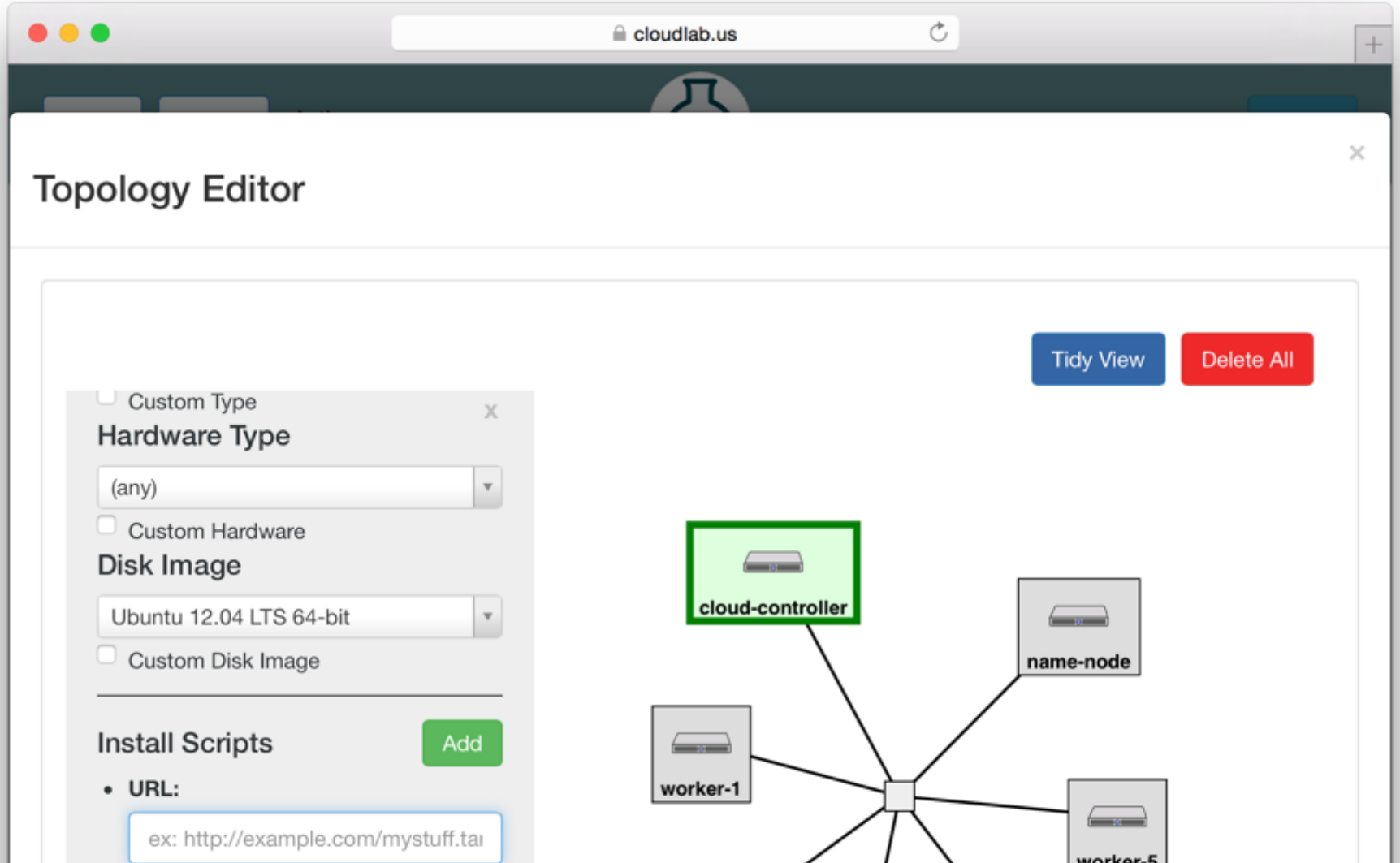
What a Profile Contains



Copy an Existing Profile



Use a GUI


cloudlab.us
Topology Editor
Tidy View
Delete All
Custom Type
Hardware Type
(any)
Custom Hardware
Disk Image
Ubuntu 12.04 LTS 64-bit
Custom Disk Image
Install Scripts
Add
URL:
ex: http://example.com/mystuff.tar
cloud-controller
name-node
worker-1

Write Code

```
#!/usr/bin/env python
"""An example of constructing a profile with a single Xen VM. Instructions: Wait
for the profile instance to start, and then log in to the VM via the ssh port
specified below. (Note that in this case, you will need to access the VM through a
high port on the physical host, since we have not requested a public IP address
for the VM itself.)
"""

# Import the Portal object.
import geni.portal as portal
# Import the ProtoGENI library.
import geni.rspec.pg as pg

# Create the Portal context.
pc = portal.Context()
# Create a Request object to start building the RSpec.
rspec = pg.Request()

# Create a XenVM and add it to the RSpec.
node = pg.XenVM("node") rspec.addResource(node)

# Print the RSpec to the enclosing page. pc.printRequestRSpec(rspec)
```

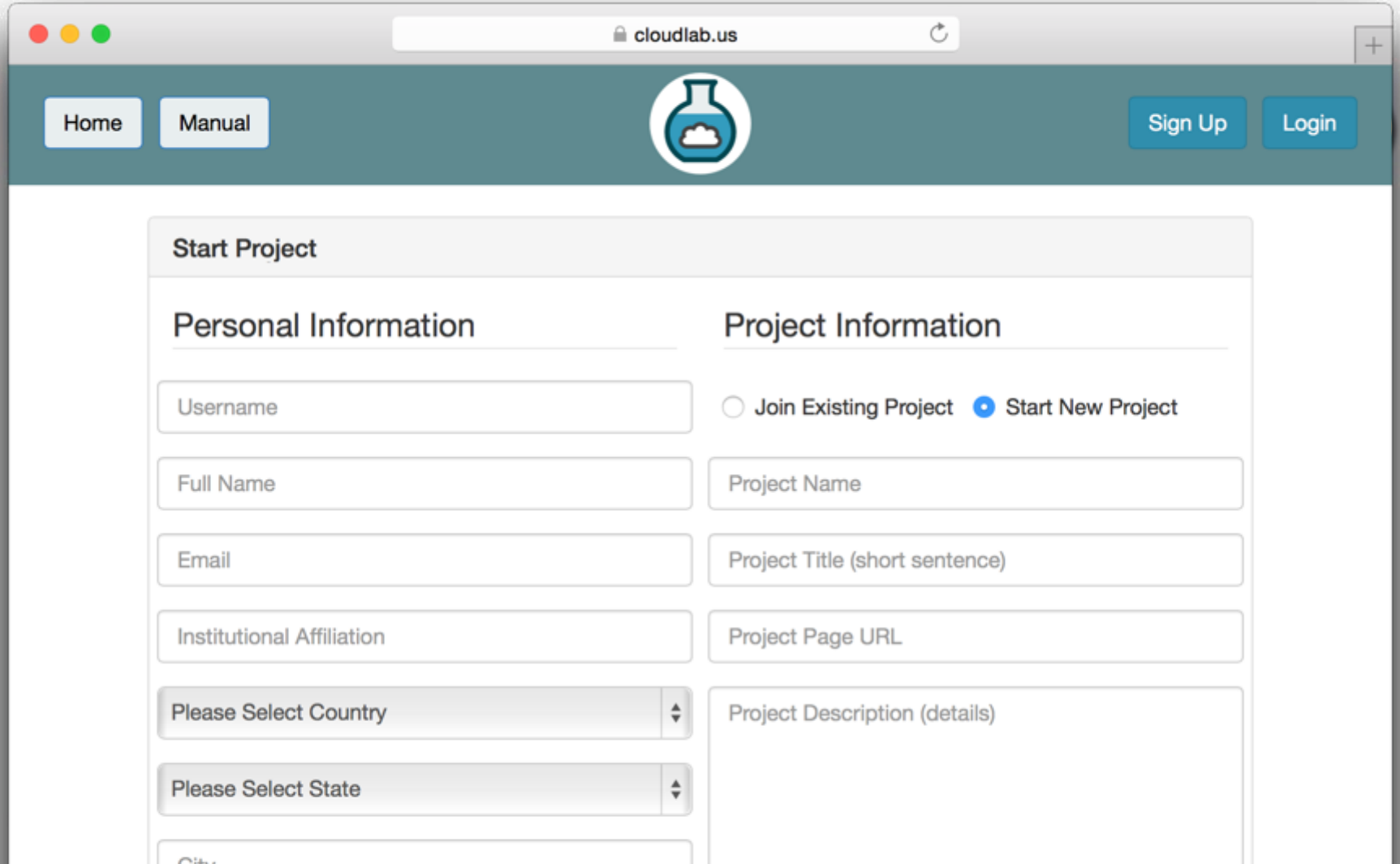
A More Complex Profile

```
# Describe the parameter(s) this profile script can accept.
pc.defineParameter( "n", "Number of VMs", portal.ParameterType.INTEGER, 1)

# Retrieve the values the user specifies during instantiation.
params = pc.bindParameters()

# Check parameter validity.
if params.n < 1 or params.n > 8:
    pc.reportError(
        portal.ParameterError(
            "You must choose at least 1 and no more than 8 VMs."))

for i in range( params.n ):
    # Create a XenVM and add it to the RSpec.
    node = pg.XenVM( "node" + str( i ) ) rspec.addResource( node )
```

Demo / Tutorial

`http://cloudlab.us/tutorial`

Sign Up At CloudLab.us



The CloudLab Team

THE UNIVERSITY OF UTAH®

Robert Ricci (PI)
Eric Eide
Steve Corbató
Kobus Van der Merwe

WISCONSIN
UNIVERSITY OF WISCONSIN-MADISON

Aditya Akella (co-PI)
Remzi Arpaci-Dusseau
Miron Livny

CLEMSON UNIVERSITY

KC Wang (co-PI)
Jim Bottum
Jim Pepin

Raytheon BBN Technologies

Chip Elliott (co-PI)
Larry Landweber

UMASS AMHERST

Mike Zink (co-PI)
David Irwin

usignite

Glenn Ricart (co-PI)



Learn more, sign up, share your research:

www.CloudLab.us

This material is based upon work supported by the National Science Foundation under Grant No. 1419199. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.