TROPIC: Transactional Resource Orchestration Platform In the Cloud

Changbin Liu†, Yun Mao*, Xu Chen*, Mary Fernandez*,
Boon Thau Loo†, Jacobus Van der Merwe*

† Penn — University of Pennsylvania

* at&t

netdb.cis.upenn.edu/dmf

Motivation

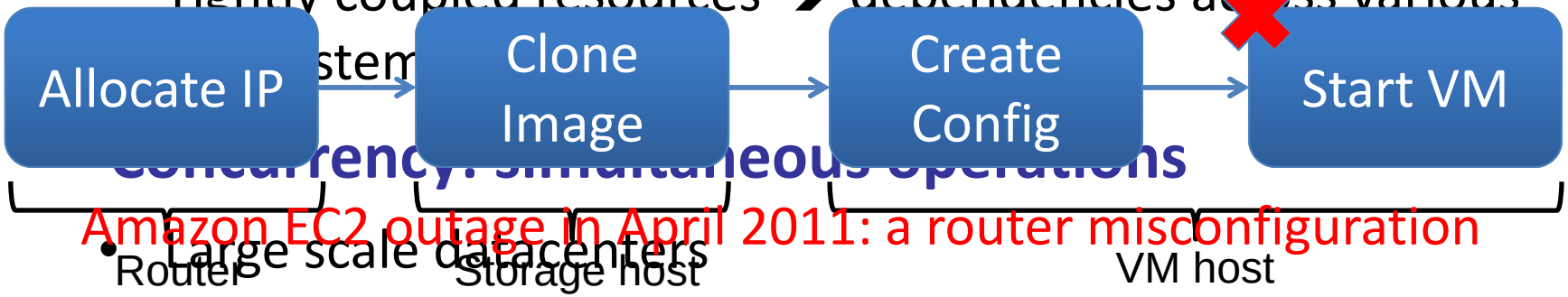
- **Infrastructure-as-a-Service (IaaS) Cloud**
  - Provide cloud infrastructure services: virtual machines (VMs), virtual block devices, VPNs
  - Widely adopted, e.g. Amazon Elastic Compute Cloud (EC2)

- **Cloud resource orchestration**
  - Provision, configure, manage, decommission
    - Compute: VM spawn / start / stop / migrate / destroy
    - Storage: replicate, deallocate
    - Network: allocate IP, create VPN

Key Challenges

- **Robustness: volatile environment**
  - Buggy software, unstable hardware, transient network disconnections, power outage
- **Safety: enforce service and engineering rules**
  - Prevent misconfigurations / misoperations
  - Tightly coupled resources ➔ dependencies across various system
- **Concurrency: simultaneous operations**
  - Large scale datacenters
  - Fast resource acquisition and release

Fail-stop ➔ unused image, config file, IP



Amazon EC2 outage in April 2011: a router misconfiguration

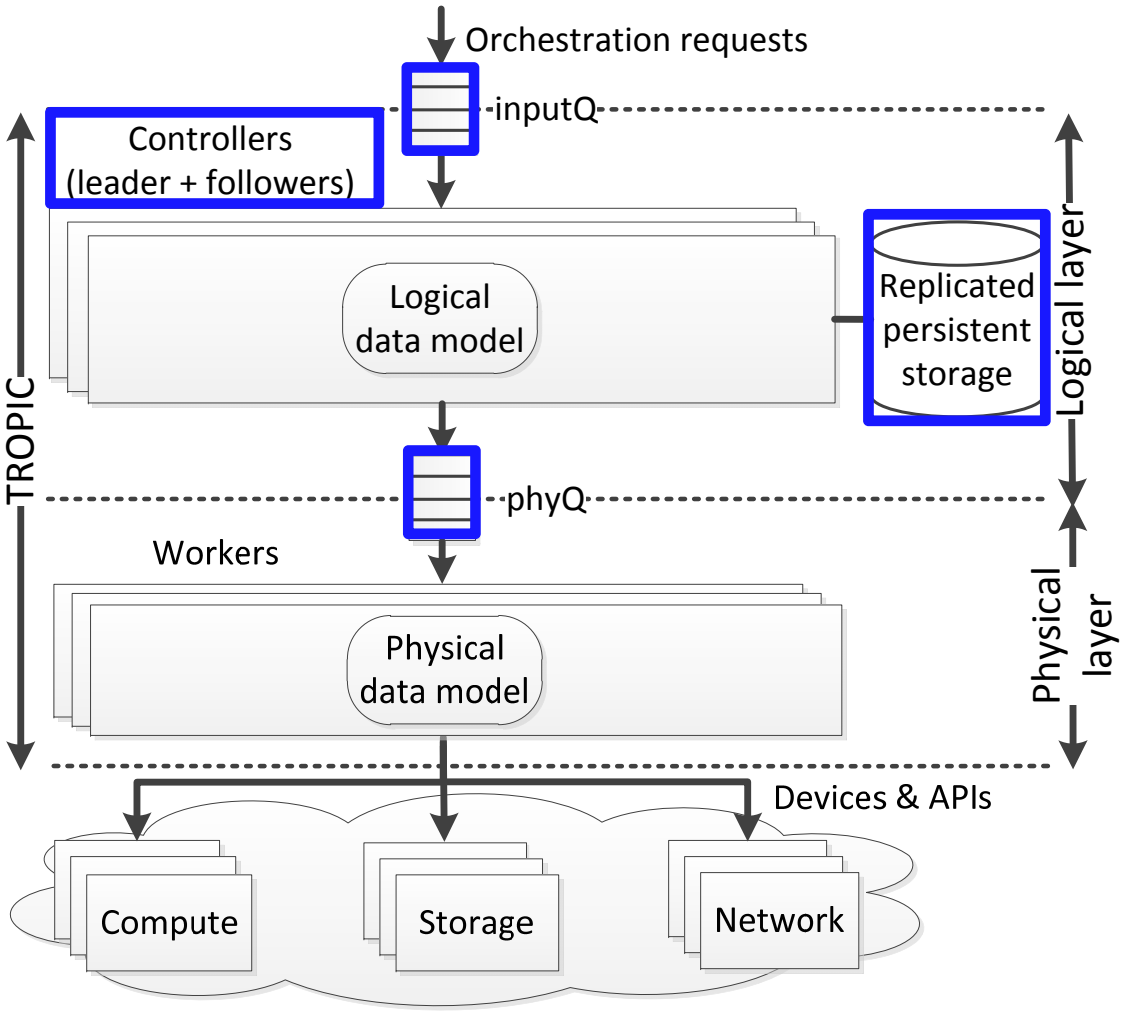
4-step VM spawning

Our Approach: TROPIC

- Key idea: Orchestration ➔ ***Transactions***
- Well-understood **ACID** properties (similar to database)
  - **A**tomicity and **D**urability for **robustness**
  - **C**onsistency for **safety**
  - **I**solation for **concurrency**

No worry about the complexities of accessing and managing underlying volatile resources!

TROPIC Architecture



Logical / physical layer

- Replica: weak, eventual consistency
- Transaction scheduling, concurrency control, simulation, reconcile cross-layer inconsistency

Replicated components

- Multiple controllers
- Distributed queues
- Persistent storage

Robustness (Atomicity, Durability)

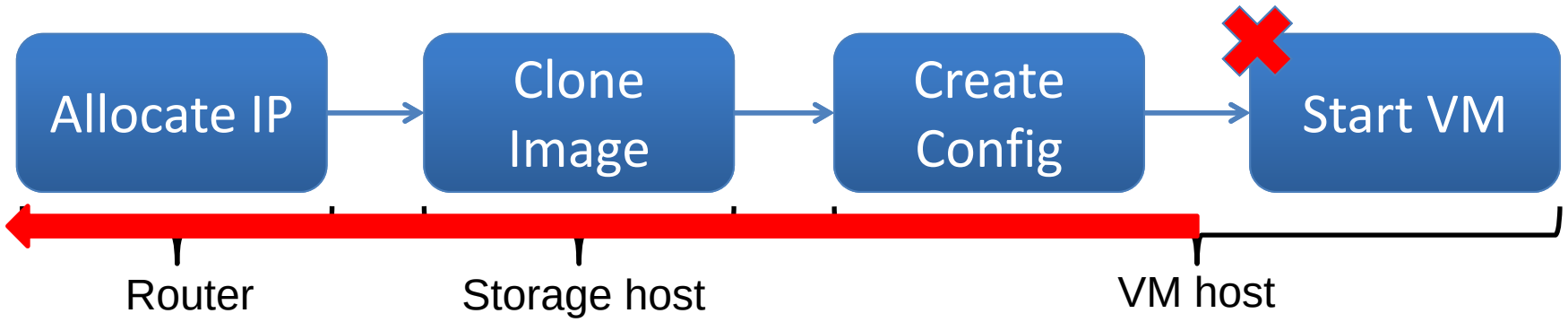
0 **Atomicity**

Txn.begin()

- Execute physical orchestration operations
- If error: abort and rollback via logging and undo actions

Txn.commit()

1



4-step VM spawn

Safety (Consistency)

Txn.begin()

- Logical layer simulation to check for constraint violations
- Execute physical orchestration operations
- If failure: abort and rollback via logging and undo actions

Txn.commit()

– **Constraint examples**

- Aggregate VM memory <= Host memory capacity
- EC2: next hops of primary routers can not be backup routers

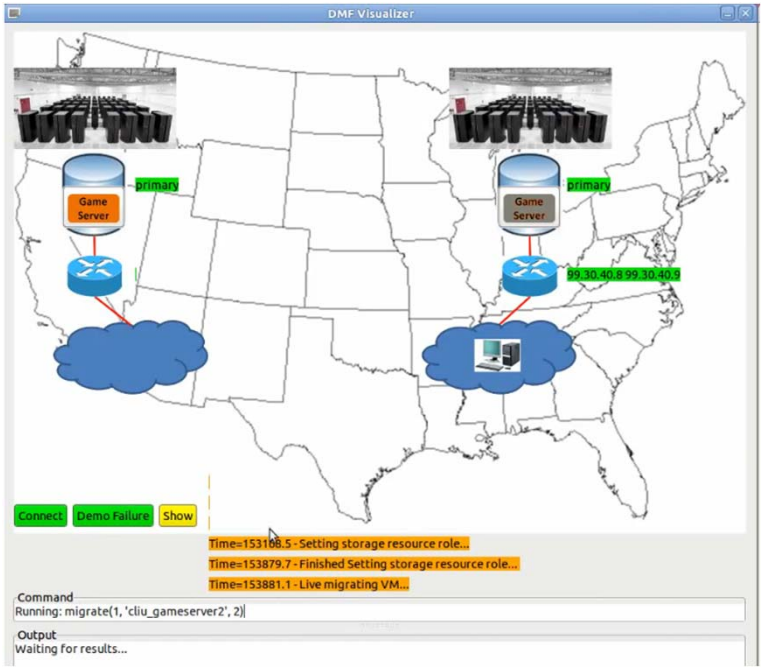
```
@constraint
def nextHopConstraint(self):
    if self.nextHop in backupRouters:
        yield ("Amazon EC2 outage!", self)
```

Implementation

- **System prototype**
  - Implemented in 11K LOC Python
  - 1K LOC as core transaction logic
- **Cloud resources modeling**
  - Compute: Xen VMs, libvirt APIs
  - Storage: DRBD, GNBD, LVM
  - Network: Juniper routers
- **ZooKeeper for coordination and storage**
  - Distributed queues, leader election, key-value store

Cloud Services

- **Adaptive Cloud**
  - A mini Amazon EC2, plus VM migration in LAN
  - Deployment on 18 hosts in 3 data centers (CA, IL, TX)
  - Manage 1,000+ CentOS VMs
- **Follow-the-Sun**
  - Wide-area live VM migration across data centers



Evaluation

- **Scalability**
  - Scales up to 5X Amazon EC2 traces (collected in US-east region in July 2011), median transaction latency < 1s
  - Manage up to 225K servers, 1.8 million VMs
  - Constant transaction throughput (55-85 /s) as load scales up
- **Low overhead**
  - Robustness: transaction rollback upon failure
  - Safety: enforcing constraints
  - < 0.01s per transaction
- **High availability**
  - Transactional semantics are always guaranteed even during controller failure and recovery

Summary

- **TROPIC: transactional orchestration**
  - Robustness, safety, concurrency, high availability
  - System prototype with testbed deployment
  - Rapidly build reliable and sophisticated IaaS cloud services
- **Open-source code release**
  - Integration with OpenStack
- **Automated cloud orchestration** [SOCC'11, VLDB'12]
  - Provider operational objectives
  - Customer service level agreements
  - Declarative constraint optimization

Thank you

netdb.cis.upenn.edu/dmf