Resources experience changing demand
Purchase resources for an estimated peak demand
Over time, demand often increases
  - Resources may become insufficient during times of peak demand

How do we accommodate?
Queue wait times increase
Storage runs out
Resource crashes
Buy More

- Expensive
  - Hardware
  - System administration time
  - Power

- Time-consuming
Assumption: control over the remote resource is with the site, not the user
- Will they let you run a service?
- Deploy a custom database?
- Give you root?

Resources are physical: difficult to expand and contract dynamically
IaaS Cloud Computing

- Infrastructure-as-a-Service (IaaS)
  - Nimbus, Eucalyptus, Amazon EC2
  - Not Platform- or Software-as-a-Service

- Virtualization
  - Give control of the resource to the user
  - Easy to expand and contract dynamically

- Lease it when you need it
Elastic Site

Demand

Static Site

Compute Resource

Monitor Demand

Elastic Site Resource Manager

Launch or Terminate Resources

Cloud Resources

Cloud Resources

Cloud Resources
Initial Target Workloads

- High Throughput Computing
- Many Task Computing
- Embarrassingly Parallel Workloads
  - BLAST
  - Rendering (computer graphics)
  - Brute-force key searching
Initial Assumptions

- IaaS clouds
- No communication between cloud nodes
  - Individual cloud compute nodes
- No shared file system for the cloud nodes

- We’ll come back to the last two assumptions later…
Elastic Site Implementation

Demand

Static Site
- Torque / PBS Queue
- Physical Compute Nodes
- Storage Resource
- Monitor Demand

Python Elastic Site Resource Manager
- iptables
- Launch or Terminate Resources

Elastic Site
- Nimbus Clouds
- Amazon EC2
Elastic Site Architecture

- Elastic Site Manager
  - Main Driver
  - Execute Policy
  - Manage Cluster

Cloud Infrastructure (Nimbus or EC2)
- Start/Stop VM
- Status
- Node Joins Cluster
- Dispatch Job
- Job Completes

Cluster Queue (Torque)
- Status
- Query, Update
Integrating the Cloud Nodes

- Cloud nodes boot a basic Linux image running pbs_mom

- Host-based authentication between cluster head node and cloud node (preconfigured in the image)

- New SSH key generated for a cloud node at boot
  - Public key needs to be added to cluster head node
  - Nimbus context broker
Policies

- Policies decide when to launch or terminate a cloud node
- Implemented as individual Python modules
- Static: select one and stick with it
- Estimated waste time for a particular cloud = estimated node startup time + estimated node shutdown time
Initial Policies

- Initial set of policies:
  - On Demand
    - One VM for each job
  - Steady Stream
    - 1 VM always running
    - Launch: (Total queued time) > (5 * Estimated waste time)
    - Terminate: (Total queued time) < (3 * Estimated waste time)
  - Bursts
    - Always boot at least 1 VM
    - Launch num. nodes = (Total queued time) / (Estimated waste * 2)
  - Dedicated (evaluation)
    - 10 VMs always running
More on Policies

- These policies are an initial starting point
- Will tweak in the future
- Create other policies for other scenarios

- Eventually we want to make intelligent scheduling decisions
  - Build a layer on top of policies
  - Dynamically select policies
  - Use the policy that makes the most sense given the current conditions and what you expect to see in the future
Evaluation

- Compare Clouds
  - Startup and shutdown time

- Examining Waste
  - VM Startup Time
  - Idle Nodes
  - VM Shutdown Time

- Examining Reactiveness of Policies
The Head Node

- Located at the University of Colorado at Boulder
  - Running Torque and the Elastic Site Manager
  - Launch and terminates nodes with the Nimbus cloud client and EC2 Java command-line tools

- Two 2.4GHz Xeon processors with hyperthreading

- 6GB of RAM
We are not comparing the underlying hardware, network, or hypervisors
  - See references

We are examining the 3 policies
  - Which policies minimize waste?
  - Do the policies respond quickly (and appropriately) to changes in demand?
  - Does it scale?
Comparing Clouds
UChicago vs. Indiana: Startup 1 VM

Seconds

May 18, 2010
UChicago vs. Indiana: Startup 5 VMs
Amazon is a different story...
we’ll come back to it.
Examining Waste
Which policies minimize waste?
The Workloads

- **Workload 1**
  - “A burst of short jobs”
  - 20 60-second jobs submitted immediately

- **Workload 2**
  - “Steady stream of small jobs”
  - 10 60-second jobs each 30 seconds apart
  - Sleep 5 minutes
  - 10 120-second jobs each 10 seconds apart
Elapsed Time of the Workload

OD = On Demand
SS = Steady Stream
BT = Bursts
DD = Dedicated

OD = On Demand
SS = Steady Stream
BT = Bursts
DD = Dedicated
Overhead is the amount of time VMs are active but not running any jobs for all VMs. This includes startup and shutdown time as well as idle running time.
Reactiveness

Do the policies respond quickly (and appropriately) to changes in demand?
On Demand

- **VMs Running**
- **Total Jobs (Queued or Running)**

May 18, 2010
Steady Stream

![Diagram showing VMs Running and Total Jobs (Queued or Running)]

- **(1–10 minute jobs)**
- **(1 minute jobs)**

**Count**

- VMs Running
- Total Jobs (Queued or Running)

May 18, 2010
Scalability

Does it scale?
Bursts on Amazon EC2 (no maximum)

VMs Running

Total Jobs (Queued or Running)
Does it scale?

- It ramps up to process all of the jobs
- ...but it’s not very reactive
What’s the problem?

- We wrapped the EC2 Java command line tools
- The head node experienced very high load
  - We had to reduce the number of launch threads down to 6 (from 10) ... JavaVMs were eating up the system
- We used a gateway service at UChicago instead of connecting directly to Amazon
- EC2 startup times varied between 74 seconds and 205 seconds
Future Work

- Integrate directly with APIs
  - Avoid wrapping command-line tools
- Dynamic selection of policies
- Support for multiple clouds simultaneously
- Economic considerations of policies
  - You’re charged for an entire hour when you boot an EC2 machine
  - Meta-scheduling: select the cheapest / fastest cloud
More Future Work

- We made two assumptions initially
  - No communication between cloud nodes
  - No shared file system for the cloud nodes

- Communication: secure network overlay
  - VPNCubed
  - Amazon Virtual Private Cloud

- Shared file system
  - sshfs?
  - Where do we put it?
    - In the cloud we’re deploying in?
    - At the site we’re deploying from?
    - Another cloud?
IaaS cloud resources can be used to dynamically extend existing static physical resources

Running jobs in the cloud can be done in a reasonably transparent manner for the user

Running jobs in the cloud can help to minimize “waste” in certain scenarios
Conclusions

- There are still a lot of issues to address
  - Security and privacy
  - Communication: MPI
  - Network latency and throughput
  - Large file I/O
Questions?

paul.marshall@colorado.edu