Using Reinforcement Learning for Controlling an Elastic Web Application Hosting Platform

Han Li      Srikumar Venugopal
School of Computer Science and Engineering
University of New South Wales, Sydney, Australia
E: hli, srikumary@cse.unsw.edu.au
The Promise of Cloud Computing
However..

• **Side-effects of elasticity**
  – Exceptions
  – Shifting bottlenecks
  – Diminishing returns
  – Uncontrolled spending

• **Plus**
  – Unpredictable demand
  – Choices..Choices
Initial Conditions

Instance 1
- App Server 1
  - app1
  - app2

Instance 2
- App Server 2
  - app3
  - app4

IaaS Provider
A Critical Event

Instance1

App Server1

app1
app2

Instance2

App Server2

app3
app4

IaaS Provider
Placement 1

IaaS Provider

Instance 1

App Server 1

app2

Instance 2

App Server 2

app3
app4
app1
Placement 2

Instance 1
App Server 1
app1

Instance 2
App Server 2
app3
app4
app2

IaaS Provider
Placement 3

IaaS Provider

Instance 1

App Server 1

app2

Instance 2

App Server 2

app3

app4

Instance 3

App Server 3

app1

$$
Placements 4 & 5

IaaS Provider

Instance1

App Server1

app1
app2

Instance2

App Server2

app3
app4
app1

Instance3

App Server3

app1
Challenges of App Placement

• Load shifts are dynamic

• Multiple applications may go critical simultaneously

• Instance provisioning should be controlled

• Application QoS must be maintained
Twin Objectives

• Provisioning
  – Smallest number of servers required to satisfy resource requirements of all the applications

• Dynamic Placement
  – Distribute applications so as to maximise utilisation yet meet each app’s response time and availability requirements
Co-ordinated Control of Elasticity

- Instances control their own utilisation
  - Monitoring, management and feedback
- Local controllers are learning agents
  - Reinforcement Learning
- Controllers learn from each other
  - Share their knowledge and update their own
- Servers are linked by a DHT
  - Agility, Flexibility, Co-ordination
Software Architecture for Elastic Scaling
Basic Actions

**Instance**

- create (-3.5)
- terminate (3.5)
- find (3.5)

**Application**

- move (0.5)
- duplicate (0.5)
- merge (0.5)
Co-ordination using *find*

- Server looks up other servers with the least load
  - DHT lookup
- Sends a *move* message to the selected server
- Replies with *accept* or *reject*
  - *accept* has a +ve reward
Shrinking

- The controller is always reward maximising
  - Highest Reward is for merge+terminate
- A controller initiates its own shutdown
  - Low load on its applications
- Gets exclusive lock on termination
  - Only one instance can terminate at a time
- Transfers state before shutdown
Experiments

• IaaS provider: Amazon EC2
  – small instances and high CPU instance
• Load-tester: Apache Jmeter
• Application server: Tomcat 6.0
  – JVM with 1 GB RAM
• Server thresholds: 60% and 85%
Experiments

- Six web applications
  - Test Application: Hotel Management
  - Search → Book → Confirm
- Five were subjected to a background load
  - Uniform Random
- One was subjected to the test load
- Application thresholds: 200 and 500 ms
- Average Response Time, Drop Rate, Servers
Results: Peaking Load

Graphs showing the number of requests per minute, average response time, number of servers, and percentage of requests dropped over time.
Results: Normal Distribution
Conclusion

• Demonstrated a co-ordination architecture
  – elastic provisioning and placement of web applications

• Each server is independent
  – Co-ordination via rewards

• The system is managed by set of simple states and actions

• Distributed reinforcement learning
Future Work

- Extension to SOA
- Standby servers
- Distributed reinforcement learning
- Scaling the database
Thank you!

Questions?

srikumarv@cse.unsw.edu.au