Hogna:
A Platform for Self-Adaptive Applications in Cloud Environments

Cornel Barna, Hamoun Ghanbari,
Marin Litoiu and Mark Shtern

York University, Toronto, Canada

Agenda

- The problem and challenges
- Adaptive Systems
- Hogna
    - Overview
    - Configuration file
    - Components

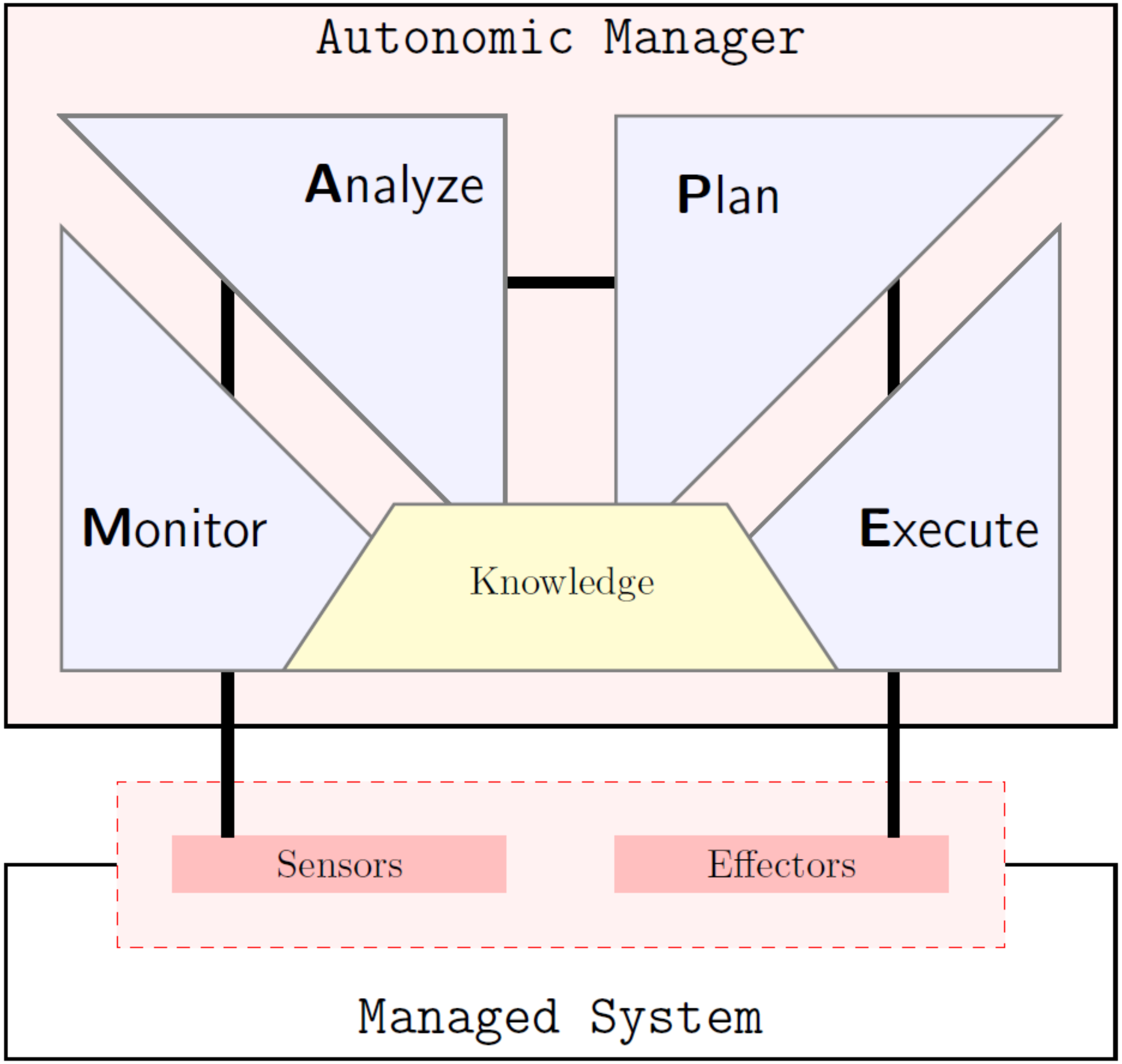
The Problem

- A researcher wants to test management algorithms/strategies for applications deployed in cloud, and explore their advantages and disadvantages.

Challenges

- Communication complexity with the cloud provider
  - Interaction using API requires: authentication, keep track of instances and their status;
- Deploying software on new instances and configuring it
- Extracting metrics from the instances
  - Customs metrics;
  - Consolidation of metrics;
- Executing custom actions as part of cloud management

Adaptive Systems



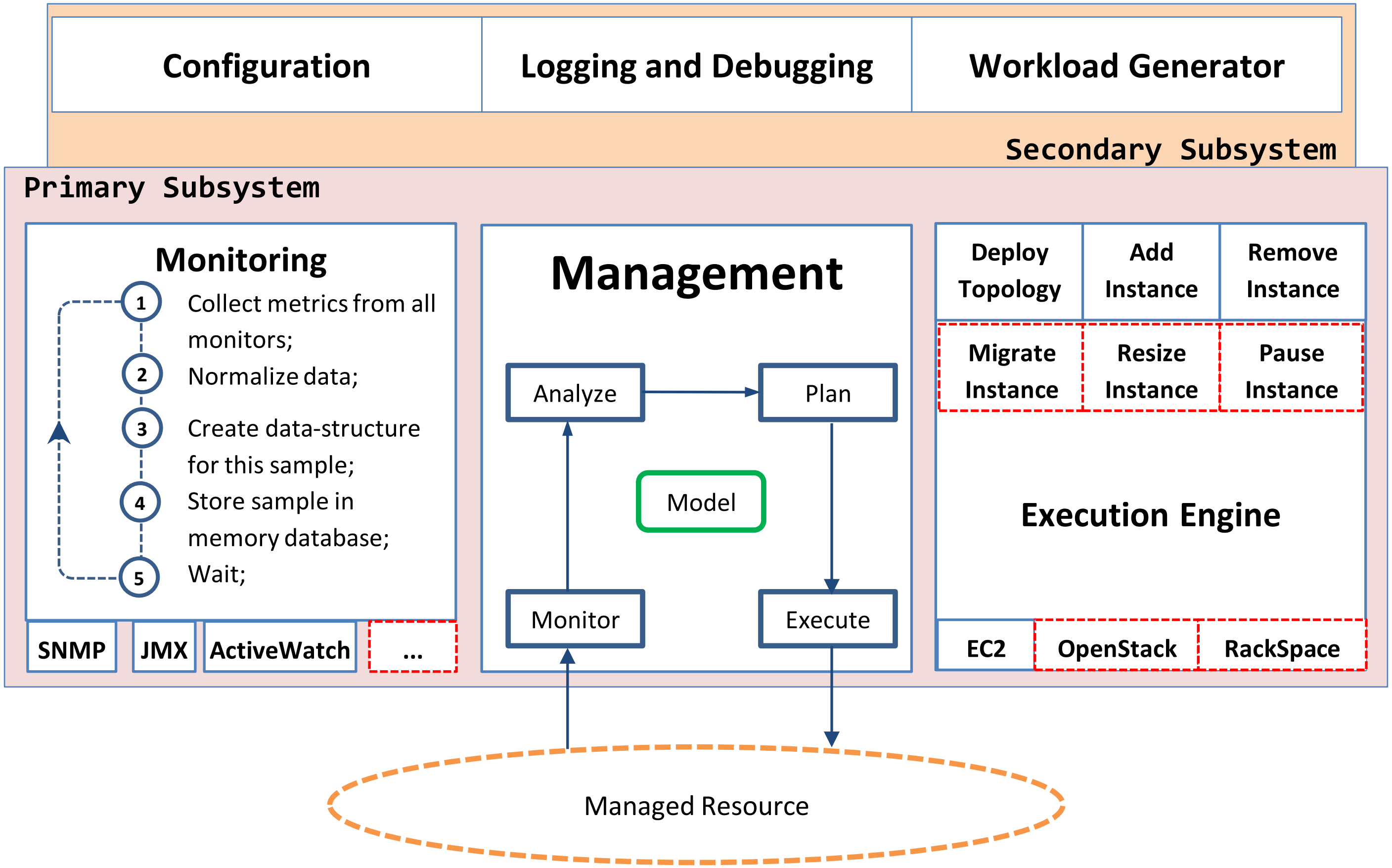
Hogna

- Platform for deploying self-adaptive applications on clouds
- Implements the MAPE-K loop
- Advantages:
  - Replace only the management logic, thus comparing the efficiency of multiple managements strategies
  - Replace the execution engine, thus using different cloud providers
  - Replace monitors to get more accurate metrics
  - Replace only the managed resource, thus comparing management strategies against different application types
  - Works on live systems, instead of simulated ones

Hogna: Features

- Automatic deployment of a topology
- Monitoring that handles metrics extraction and consolidation
- Performance model to be used during adaptation process
- Mechanism to insert logic to analyze data and plan the changes
- Extensibility: almost every aspect can be customized, and every component can be replaced

Hogna's Architecture



Hogna: Example

```
public static void main(String ... args) throws Exception
{
    ConfigurationManager.Configure("./application.config");

    HognaEngine theApp = new HognaEngine();

    SimpleDecisionEngine decEngine = new SimpleDecisionEngine();
    decEngine.AddExpandRule(  new ClusterResizeRule(
                    "/WebCluster/Average/CPUUtilization", 0.80, "WebCluster",  1));
    decEngine.AddContractRule(new ClusterResizeRule(
                    "/WebCluster/Average/CPUUtilization", 0.40, "WebCluster", -1));
    theApp.SetAnalyzer(decEngine);

    theApp.SetPlanner(decEngine);
    theApp.SetMonitorEngine(new TopologyMonitorManagerV2());
    theApp.SetActuator(new AmazonSimpleAppActuator());

    theApp.Run();
}
```

Hogna: Example with Model

```
public static void main (String ... args) throws Exception
{
    ConfigurationManager.Configure("./application.model.config");

    HognaEngine theApp = new HognaEngine();

    theApp.SetModel ("./Simple DB Operations.model.pxl");
    theApp.SetFilter("./Simple DB Operations.kalman.config");

    SimpleDecisionEngine analyzer = new SimpleDecisionEngine();
    analyzer.AddExpandRule(  new ClusterResizeRule(
                    "/WebCluster/Average/CPUUtilization", 0.80, "WebCluster", 0));
    analyzer.AddContractRule(new ClusterResizeRule(
                    "/WebCluster/Average/CPUUtilization", 0.40, "WebCluster", 0));
    theApp.SetAnalyzer(analyzer);

    theApp.SetPlanner(new SimpleModelPlanner());
    theApp.SetMonitorEngine(new TopologyMonitorManagerV2());
    theApp.SetActuator(new AmazonSimpleAppActuator());

    theApp.Run();
}
```

Input File: Topology

```
<configSections>
  <section name="topology" type="Application.Configuration.TopologyConfigurationSection" />
</configSections>

<topology>
  <cluster name="Web Cluster" id="WebCluster">
    <node name="Web Balancer" type="balancer" ami="ami-05eebb6c" size="m1.large"
security="corba" region="us-east-1d">
      <container name="Apache 2"> <service name="proxy_balancer" id="proxy_balancer" />
</container>
    </node>
    <node name="Web Host" type="worker" ami="ami-05eebb6c" size="m1.small" security="corba"
region="us-east-1d">
      <container name="Tomcat 6"> <service name="Simple Database Operations" id="webAppSDO" />
</container>
    </node>
  </cluster>
  <dependencies>
    <dependency from="proxy_balancer" to="webAppSDO" />
  </dependencies>
</topology>
```

```
TopologyConfigurationSection secTopology = (TopologyConfigurationSection)
                                  ConfigurationManager.GetSection("topology");
Topology theTopology = secTopology.GetTopology();
```

Configuring Instances

```xml
<ec2>
  <configHelpers>
    <helper serviceId="proxy_balancer"
          type="Framework.Cloud.EC2.ConfigHelperLoadBalancerWithProxy" />
  </configHelpers>
</ec2>
```

```java
public class ConfigHelperLoadBalancerWithProxy implements IConfigHelper {
  public void Configure (Node node) {
          String configScript = "sudo service apache2 start; exit 0;";
          SshClient.ExecuteCommand(node.GetPublicIp(), configScript);
  }

  public void AddDependency (Node depFrom, List<Node> depTo) {
          // create/load script that modifies "proxy-balancer.conf"
          // adding as workers all nodes in "depTo"
  }

  public void RemoveDependency (Node depFrom, List<Node> depTo) {
          // create/load script that modifies "proxy-balancer.conf"
          // removing all workers specified in "depTo"
  }
}
```

Input File: Monitors

```
<configSections>
  <section name="monitoring" type="Application.Configuration.MonitorConfigurationSection" />
</configSections>

<monitoring>
  <loaders>
    <loader type="cloud watch" value="Framework.Cloud.EC2.CloudWatchMonitorLoader" />
    <loader type="snmp"        value="Framework.Monitoring.SnmpMonitorLoader" />
  </loaders>
  <monitors>
    <monitor name="CPUUtilization" type="cloud watch">
      <description>
          Gets the CPU utilization of an instance.
      </description>
      <credentials file="./config/AwsCredentials.properties" />
    </monitor>

    <monitor name="SnmpCPU" type="snmp">
      <description> ... </description>
      <connection host="127.0.0.1" port="1610" timeout="5000" retries="2" />
      <object oid=".1.3.6.1.2.1.25.3.3.1.2.768" community="public" />
    </monitor>
  </monitors>
</monitoring>
```

Monitoring subsystem

- Automatically loads monitors from the configuration file
- Maintains a list of monitors
  - JMX, SNMP and EC2 ActiveWatch
  - New types of monitors can be added
- The list is dynamically updated when instances are added/removed
- Each monitor extracts a single value
- Works independently from other components (at any moment there is `MetricValues` object available)

Analyzer and Planner

- Must be implemented together: the output from analyzer must be understood by planner
- Analyzer
  - Implements the `IAnalyzer` interface
  - Receive the measured metrics (an object of type `MetricsValues`)
  - Evaluates system's health
  - The results are stored in an object of type `AnalyzerResults`, passed to the planner
- Planner
  - Implements the `IPlanner` interface
  - Has access to the topology, metrics, the results of the analyzer, and a performance model
  - Must to be able to interpret the analyzer's results
  - Creates a set of actions that must be executed to fix the problems identified by the analyzer

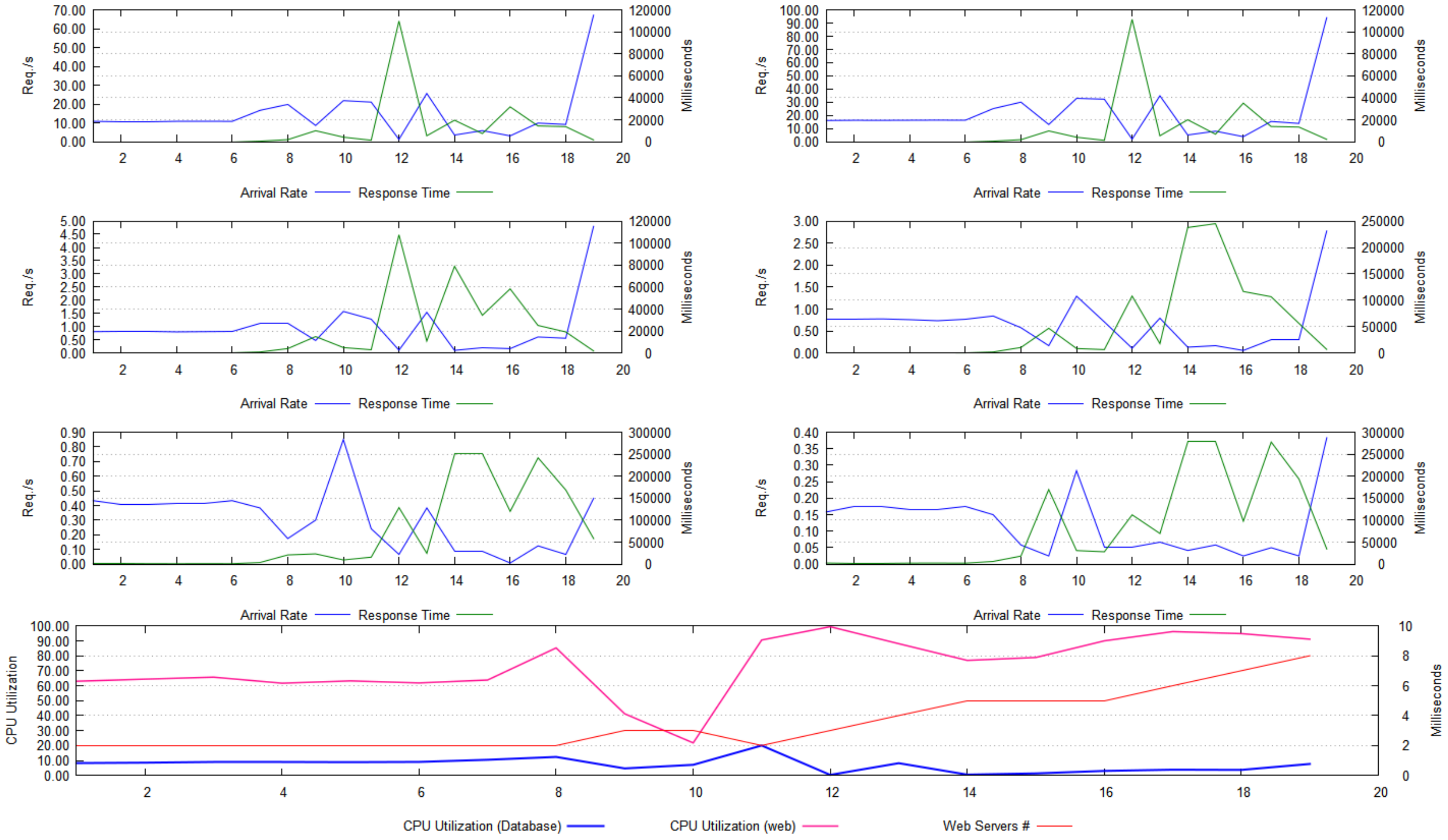
Performance Model

- OPERA is available for modeling the web applications:
  - Uses Layered Queuing Networks
  - Can be used to evaluate the impact of changes, before they are deployed
- The model's parameters are tuned automatically using Kalman Filter

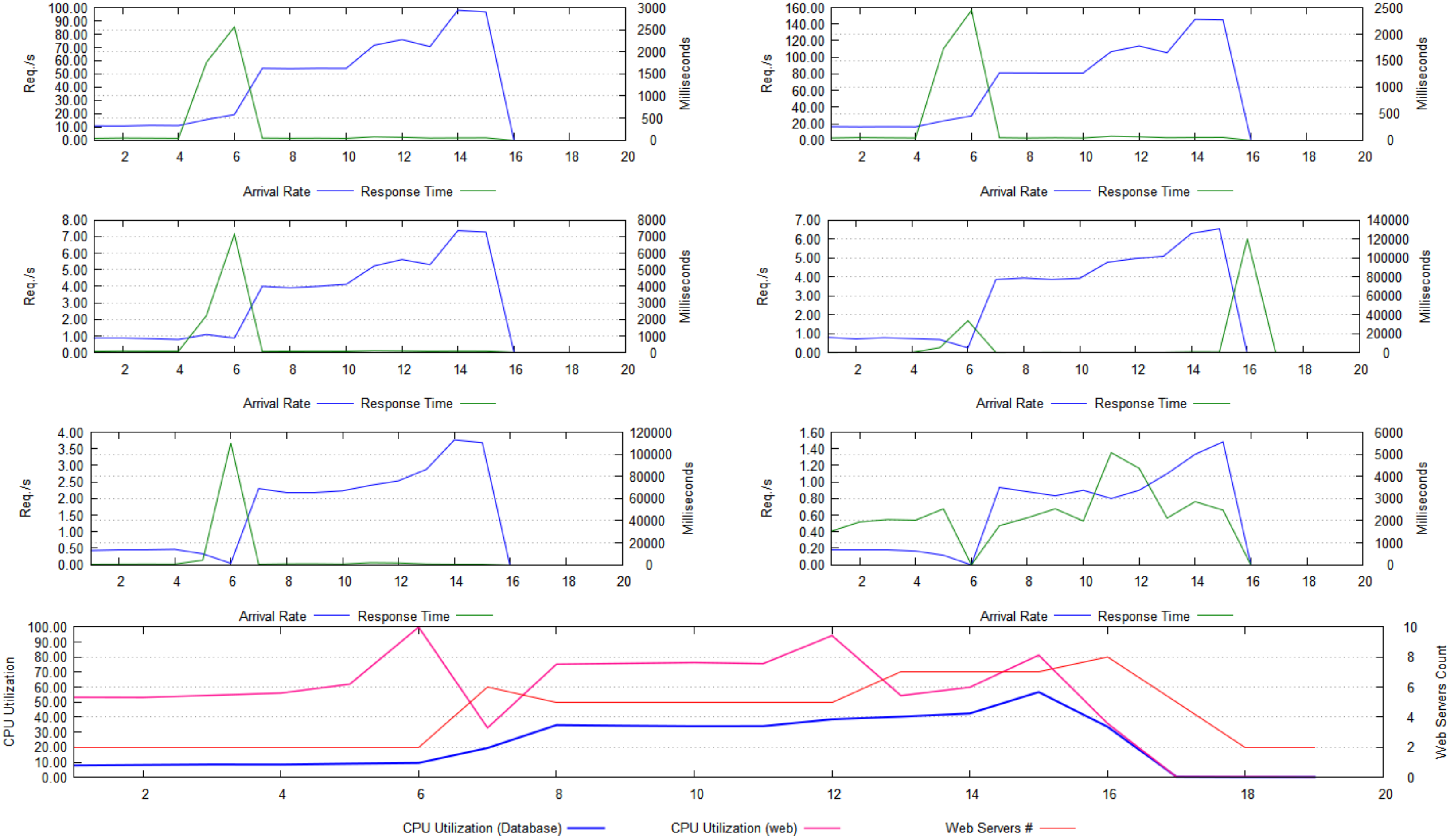
Executor

- Implements the action plan
- Handles communication with the cloud manager
- Out-of-the-box executor for Amazon EC2
  - adds/removes instances
  - deploys a topology
  - adding more actions requires customizing the executor

Case Study:
Elasticity using Thresholds



Case Study: Elasticity using OPERA



Questions?

- Hogna is available at:

http://www.ceraslabs.com/hogna

- OPERA description

http://www.ceraslabs.com/technologies/opera