The Cloud Provisioning Environment (CPE) Prototype Architecture and Technologies

Matthew Arrott, Alex Clemeans, Claudia Farcaς, Emilis Farcaς, Michael Messinger, Dorion Rejmer

The Cloud Provisioning Environment prototypes a critical part of the future OOI CI Common Execution Infrastructure (CEI) the automatic elastic provisioning of compute resources on demand by a distributed system that follows the Observe-Orient/Decide-Act feedback loop. The Cloud Provisioning Environment provides the Sensor Aggregator, Scheduler and Provisioner components that interest through a reliable publish/subscribe messaging infrastructure to launch and take down Nimbus and Amazon EC2 cloud compute nodes on demand. We present details of the prototype implementation. An elastic scaling experiment: a work producer adds work messages to a message queue and several elastically provisioned workers take work off the queue. The length of the work queue is the critical input for the scheduler to decide on the need for more or fewer worker nodes. The results critically emphasize the need to separate mechanism from policy, in order to be able to flexibly adapt and fine-tune policy to given environments.

Overview and Goals

The goal of the Cloud Provisioning Environment is to provide Policy Driven Elastic Scalability for distributed applications using reliable and scalable communications. The most important design goal was to create a system that effectively separates Policy from Mechanism.

The Policy is the logic that holds rules about what resources will be needed versus what sensors inform the system receives from the environment. In real-world use cases, this is something that is enhanced through experimentation and iteration.

The Mechanism is the surrounding infrastructure that enables dynamic provisioning of cloud resources. The users should not need to know about the internal structure of the system, with the focus on creating a problem-driven, cost-effective means of resourcing their problem. Fig. 1 shows the architecture of the Cloud Provisioning Environment.

Architecture

The Sensor Aggregator receives sensor signals through various means from the environment and aggregates them into Events that are delivered via the message-passing system to any subscribed Scheduler processes. The system supports a variety of sensor inputs from different sources that may occur on different time frequencies.

The Scheduler receives event messages from the Sensor Aggregators and makes decisions about the provisioning and termination of compute resources. It is based on decisions its knowledge of currently provisioned computing resources and user provided Policy. It sends action messages to the Provisioner.

The Provisioner receives action messages from the Scheduler with instructions to launch additional or terminate existing compute instances. Its implementation is specific for each supported target execution environment, such as Amazon’s EC2 and Nimbus.

Policy

The policy is a set of self-contained logic rules that take in the goals and current state of the system, and output the next actions to be taken by the system to manipulate the environment.

The policy may be a combination of several rules, for instance compute quantity, cost, time, etc. The logic of when and where to provision or terminate computing resources lives in the Provisioner. The policy requires the existence of variousイベント sources (worker nodes) as well as in a non-optimal way. This clearly shows the importance of separating policy from mechanism. We have demonstrated the successful implementation of a powerful mechanism, elastically provisioning and terminating resources in response to changes in the environment. We leverage more sophisticated policy for future developments.

Prototype Details

The prototype implementation combines the fundamental components of the Cloud Provisioning Environment architecture as shown in Fig. 3: the Provisioner and the Scheduler, with a simple Worker-Producer-Consumer setup that also includes a Work Message Broker and a Work Consumer. For our prototype, we chose to use as cloud providers the Amazon EC2 and a Nimbus-based private cloud environment running at the University of Chicago. We chose to deploy the main components of the CEI on Nimbus VMs and have these components deployed and message nodes on EC2. To simplify the policy, we did not dynamically provision further resources on Nimbus.

The prototype setup and instantiation process is depicted in Fig. 4. The Work Broker queues work messages awaiting to be delivered and executed. The policy requires the existence of a Work Broker instance, which also comes with a Worker Agent that measures the size of the work queue. After the Work Broker comes up, the policy requires the provisioning of one Work Producer and one Work Consumer. The Sensor Aggregator continuously impacts the work queue size and sends it in event messages to the Scheduler. The Scheduler attaches a handler to these incoming events to evaluate policy (see Fig. 3). When the queue size exceeds a “high threshold” (set to 100), it provisions another Work Consumer. When the queue size drops below a “low threshold” (set to 80), it terminates a Work Consumer. The policy will never run less than 1 or more than 15 Work Consumers. Each VM instance contains a Nimbus controller that can terminate Nimbus instances in case the Nimbus VM instance is not the one that is actively managing. The Nimbus Software System itself requires a set of environment resources as well as the Nimbus VM controller instance.

Prototype Results

The prototype demonstrates our ability to combine distinct clouds providers into a single system. It also shows how it dynamically boozes a base VM image into a fully functional “role”. Our demo has 5 distinct roles that use only a single VM image per cloud. These message clients library enables the convenient implementation of asynchronous communication using the publish/subscribe messaging protocol.  Mgenet supports several message communication styles, including point-to-point, fan-out and worker queue.

Cloud Provisioning Environment (CPE)

Figure 1: Cloud Provisioning Environment Architecture

Figure 2: Scheduler Architecture within a Observe-Orient/Decide-Act Feedback Loop

Figure 3: Elastic Scaling Experiment Results

Figure 4: CPE Prototype Setup and on U.Chicago Campus

Supported by:
- NSF (INB-0953121 and INB-0740795)
- Google
- Amazon
- Nimbus
- Calit2/UCSD; Middlebury
- Operator User
- AMQP Worker
- Nimbus Service
- Amazon EC2
- Context Broker
- Nimbus Instance
- Work Message Broker
- Work Consumer