# Overview

Amazon EC2 offers two pricing schemes for its virtual machines. The first, which Condor (via the EC2 GAHP) presently supports, are for “dedicated instances”, generally intended to provide infrastructure (“as-a-service”). These virtual machines, once active, remain so for as long as you can pay for them. The second pricing scheme, for “spot instances,” allows customers to bid (in cents per hour) for whatever resources happen to be available at the moment. Unlike dedicated instances, submitting a spot request – a bid – does not ensure that an instance starts; neither, once started, is a spot instance guaranteed to remain active. It may be terminated (by Amazon) at any time. However, the hourly cost under the second scheme tends to be much lower, therefore making it worthwhile for users whose applications can deal with unreliable computational resources. The parallels to Condor’s opportunistic computing model need not be further elaborated.

However, while the difference between dedicated and spot instances do not require any changes to Condor, the difference between spot instance requests (SIRs) and dedicated instance requests are substantial and important. Many of these differences can be avoided by restricting the kind of SIRs Condor supports, but the fact that cancelling an SIR doesn’t terminate the corresponding instance (and vice-versa) means that Condor must track two distinct remote resources for each SIR. However, we currently have no mechanism for a (grid) job to insert a (new, remote) job. Since the job is Condor’s atomic unit of control, how would the user control the SIR if the job represented the instance? Or the SIR, if the job represented the instance?

# Architecture

Our plan is to restrict SIRs in Condor so that they start at most one instance. (The API call may generate more than one SIR; each SIR, if persistent, could start an arbitrarily large number of instances, albeit one at a time.) Rather than worry about how to make one job represent both an SIR and an instance, we will, internally to the grid manager, convert a SIR job to an instance job when the former spawns the latter. If we cancel the SIR during the transition, we don’t have to worry about remembering to do so after the instance terminates – we don’t have to change anything about the existing and working code for handling an instance.

Instead, we add a subgraph to the grid manager’s EC2 state machine. This subgraph, rooted at GM\_SPOT\_START, is entered from GM\_SAVE\_CLIENT\_TOKEN if the job includes a spot price attribute. It is otherwise unreachable; this isolation helps simplify design and recovery. If the request successfully spawns an instance, it returns to the main graph of the state-machine via GM\_SAVE\_INSTANCE\_ID. (To avoid code duplication, the latter state is also modified, and transfers to GM\_SPOT\_CANCEL if it handles an SIR; control then continues in GM\_SUBMITTED.) Otherwise – on errors, faults, and failures – it generally enters GM\_HOLD to report the problem to the user.

The GM\_SPOT\_START state issues (requests the GAHP issue) the SIR and transitions to SPOT\_SUBMITTED. SPOT\_SUBMITTED and SPOT\_QUERY correspond to existing GM\_SUBMITTED / GM\_PROBE pair, and regularly poll the EC2 service to determine the status of the SIR. When the SIR spawns an instance, the transition to GM\_SAVE\_INSTANCE\_ID occurs. The GM\_SPOT\_CANCEL state cleans up the SIR.

The grid manager enters the fourth new state, GM\_SPOT\_CHECK, during recovery. It’s used to determine if we crashed between submitting an SIR and recording the corresponding ID; it checks each SIR the job credentials permit it to see on the EC2 service for the client token, as saved in GM\_SAVE\_CLIENT\_TOKEN. Because SIRs don’t presently have the ClientToken attribute, we use LaunchGroup instead. (If we only permit single SIRs, we don’t need LaunchGroup for its intended purpose.)

# Development Plan

[Jaime] Code review.

[ToddM] Testing.