RIGHT SCALE®

RightScale Grid:

Grid Computing Applications in the Cloud

A TECHNICAL WHITE PAPER

Brian Adler, Solutions Architect, RightScale, Inc.

Abstract

Grid computing is the application of multiple computational resources in a collaborative effort to perform a large or complex task. The ultimate product of this processing may be the application of a single computational algorithm or it may involve a more complex multi-phase workflow. Grid applications in classic datacenters can be costly to construct and implement and the coordination of the vast and disparate resources required can be challenging. Cloud computing, with its elasticity and virtually infinite computational and storage resources, provides an ideal infrastructure for applications of this non-continuous and resource-intensive nature.

This white paper will focus on the broad spectrum of grid and batch computing application options that the cloud provides, as well as detail the realizable benefits from moving classic datacenter-based batch processing to the cloud infrastructure. Additionally, architectural and functional details will be provided on RightScale's Grid solution, which provides a simplified framework for deploying scalable grid applications in the cloud. Particular attention will be given to the features of the RightScale Grid framework that use the elasticity of the cloud to handle the unique requirements of applications that are governed by non-structured environments such as flash mob or viral event-driven processing loads.

1 Introduction

While the majority of the web services applications that RightScale encounters are of the classical three-tier architecture, many of our customers have processing tasks that require an automated workflow encompassing one or more intermediate steps. By leveraging the capabilities of Amazon Web Services' (AWS) Simple Queue Service (SQS), Simple Storage Service (S3), and Elastic Compute Cloud (EC2), in concert with the framework provided by RightScale Grid, these grid computing applications can be implemented quickly and efficiently. The scientific, technological, and industrial applications implemented by our customers and/or RightScale Professional Services are vast, but the following is a representative sample:

- Pharmaceutical protein analysis: Several million protein compound comparisons were performed in less than a day – a task that would have taken over a week on the customer's internal resources.
- Health claims processing and analysis: Health insurance claims spanning several years were analyzed and compared for indicators of fraudulent activity. The processing was estimated to take over one month on the customer's internal datacenter, yet was accomplished in a few days via a RightScale Grid implementation.
- Media Transcoding: A viral event on a social networking site drove traffic to a customer that created videos from user-submitted photos and audio. Using the RightScale Grid framework, the number of virtual servers handling the load autoscaled from 40 to over 4,000 (and back down to 40) in the span of a few days.

Subsequent sections of this white paper will describe each of the core components of cloud-based grid computing, including the function and role of the RightScale Grid framework. Additionally, architectural details on the implementation of the RightScale Grid framework and its interaction with core web services components will be described. Finally, details will be provided on what is required for users new to cloud-based grid computing to take advantage of the benefits provided by the RightScale Grid solution.

2 Core Components

The implementation of a true cloud-based grid processing solution involves four basic components: a queuing system, a storage system, a computing platform, and a comprehensive framework to interconnect these components and ensure proper message and job flow. RightScale Grid, in concert with the Amazon Web Services solutions provided by SQS, S3, and EC2, provides the requisite components for the successful implementation of such a grid processing solution. This section will present a brief overview of each, and subsequent sections will discuss the interrelations of these components and how they can be combined to implement dynamically elastic grid or batch computing applications.

SQS: Simple Queue Service offers a reliable, highly scalable, hosted queue for the storage and retrieval of messages as they are passed from one compute process to another. These messages can be up to 8 KB in size and can remain in a queue for up to four days. Messages are enqueued and dequeued via simple API (Application Programming Interface) calls, and access control mechanisms can be enacted to provide secure storage and retrieval of these messages. Additionally, messages are stored redundantly across AWS' datacenters to enhance reliability and availability.

S3: AWS's Simple Storage Service provides a storage mechanism for any type of user or application data. Individual files are limited to 5 GB in size, but there is no hard limit on the total amount of data that

a user can store in S3. While there is obviously a practical limit, for all intents and purposes, S3 can be thought of as a limitless storage bucket (for example, some RightScale customers currently store petabytes of data in S3). As with messages in SQS, files in S3 are stored in multiple datacenters within the AWS infrastructure to ensure reliable and efficient access to the data.

EC2: The Elastic Compute Cloud service provides virtually unlimited compute capacity in the cloud, enabling users more flexibility in performing computationally-intensive applications. While the elasticity this service provides is of great benefit in implementing scalable websites that can expand and contract capacity based on dynamic traffic patterns created by viral events or other unscheduled circumstances, it is also ideally suited to the grid computing paradigm.

RightScale Grid framework: The RightScale Grid framework coordinates the automated workflow of messages and jobs as they move through the computational, storage, and retrieval processes. RightScale Grid also provides the mechanism to implement the elasticity of the grid processing solution. The input queue(s) of the system are continually monitored, and when certain criteria are met, additional worker instances are launched to handle the increased processing load. When the number of items in the input queue decreases, these servers are automatically terminated, thus taking full advantage of "utility computing" in that the user only pays for the computing resources they use (and more importantly, that they need).

3 RightScale Grid: Anatomy of a Workflow

This section will describe the end-to-end flow of a job through the RightScale Grid framework, highlighting the components supplied and automated by RightScale, as well as the components and responsibilities of the end user.

The overall architecture and workflow of a grid processing system implemented with the RightScale Grid framework is shown in Figure 1.

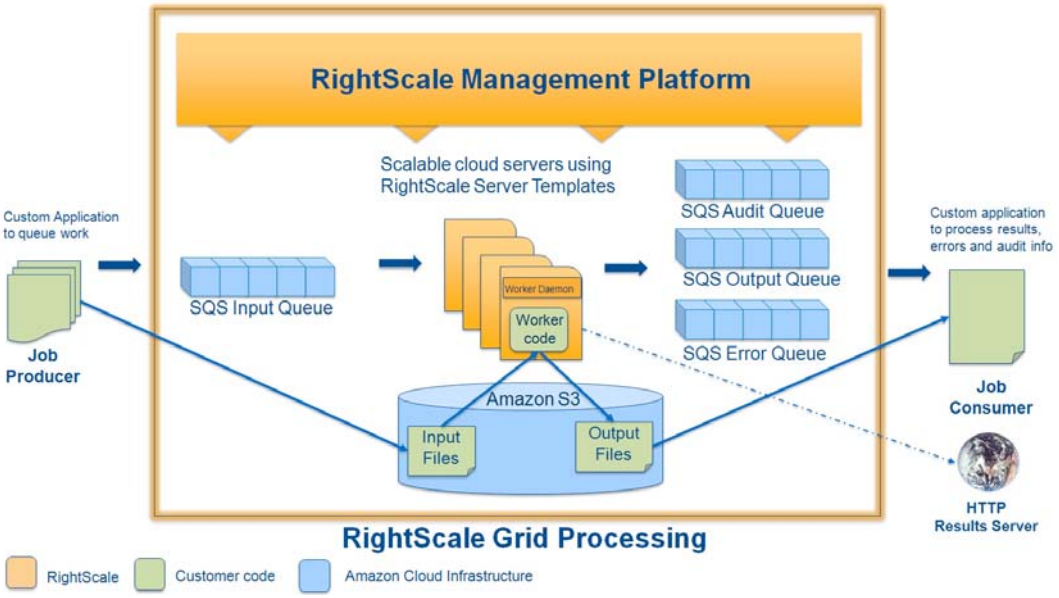

Figure 1 – RightScale Grid architecture and workflow

In this figure, everything shown within the box is under the control of the RightScale Cloud Management Platform. This includes managing the AWS infrastructure components, the worker processes, and the elasticity metrics and controls. The user-supplied components (shown in green) are limited to the Job Producer, the input files needed by a job, the worker code to execute the job, and finally an (optional) Job Consumer process that is responsible for any post-processing or output file manipulation that may be required. Although these Job Producer and Job Consumer processes are shown in Figure 1 as external to the RightScale platform, they can be (and often are) run under control of the platform as well. The result of a setup such as this is an entire end-to-end grid processing solution managed within a single interface.

Prior to creating a workflow, the four SQS queues indicated in Figure 1 (input, output, error, and audit) need to be created. This can easily be performed via the RightScale platform by using a predefined macro to assist in various system configuration tasks. By running the RightScale Grid macro within the RightScale dashboard, these four queues will be created automatically.

The workflow process is initiated by the Job Producer. This can be an application running within the cloud (and under management of the RightScale platform), or it can be running in a customer or hosting provider infrastructure. The first role of the Job Producer is to create the input file(s) required by the job. It then constructs the work unit data structure with details about the inputs and the work to be performed. Next, it uploads the input files to S3 and creates a message data structure which encompasses the work unit information in addition to details on the input file location. Finally, the Job Producer inserts this new message into the SQS Input Queue. This process is repeated for every new job that is to be introduced into the workflow. The Job Producer application can take virtually any form: a PHP script, a compiled application, a Ruby script, etc. The RightScale Grid macro mentioned previously generates a Job Producer Ruby script that can be modified and used in new applications, or can be used as a model or template for developing a new Job Producer. The RightScale Grid macro creates a single Job Producer for simplicity of illustration, but multiple Job Producers can be run simultaneously feeding the same input queue (or even multiple input queues in more complex scenarios).

The RightScale platform contains an Elasticity Daemon whose function is to monitor the input queues, and to launch worker instances to process jobs in the queue. Different scaling metrics can be used to determine the number of worker instances to launch and when to launch these instances, with the most common metric being the number of jobs in the queue. Within the RightScale dashboard, the user can specify that for every N jobs in the input queue, a worker instance should be launched. The second metric that can be used is related to the length of time that jobs have been in the queue. This is useful in situations where particular Service Level Agreements (SLA) need to be met. Using this time-based metric, the user may specify that when the average time a job spends in the queue exceeds a certain threshold, a worker instance should be launched. Similarly, when the maximum time that any job has been in the queue exceeds a specified value, a worker instance is launched to assist in the processing load. (These averages and maximums are calculated using the previous 10 jobs that were processed from the input queue.)

Once the Elasticity Daemon has determined that additional worker instances are required, a call is made within the RightScale platform to initiate the allocation of these server resources (illustrated by the orange blocks in Figure 1 containing the Worker Daemon and Worker Code). These worker instances are launched in a server array, which can be controlled via configuration settings in the RightScale dashboard. These settings allow the minimum and maximum number of worker instances to be specified. This provides a mechanism to ensure there are always one or more instances available to process any jobs that are inserted into the queue, and to limit the maximum capital expenditure in the case of unplanned or out-of-control input growth. Another feature of the RightScale platform that plays

a major role in this phase of the workflow is the ServerTemplate. Prior to launching a grid computing application, a ServerTemplate for the worker instances is created indicating the instance-specific details, such as the size of the instance, the image to use as the base operating system, the region and availability zone in which the instance should be launched, along with other configuration information. This ServerTemplate can be created manually or automatically by calling the RightScale Grid macro. Another key aspect of the ServerTemplate is that it specifies the technology stack required on the instance (Ruby, Perl, PHP, etc.) and performs the installation of these tools as well as the installation of the worker code. This worker code is downloaded from an SVN repository, and installed on the worker instance as specified in a script run at the end of the instance's boot cycle. Optionally, the worker code can be included as an attachment to the ServerTemplate, but this method is not as flexible as downloading from a repository, and is not considered a best practice.

In addition to the user-specified tools and worker code, the RightScale Grid Worker Daemon is also installed on the worker instance. This worker daemon is responsible for managing the workflow on the instance, executing the worker code, and uploading output files to S3 to be processed by the Job Consumer. Numerous functions are performed by the RightScale Grid Worker Daemon and the worker code, but they can be generally categorized as follows:

1) The Worker Daemon retrieves a message from the input queue. (It does not delete this message, but instead makes a local copy, while rendering the actual message in the queue invisible to other processes. This is done in case there is a failure in the processing of the job. If a failure occurs, the message will again become visible after a configurable timeout period and will be picked up by another worker daemon, thus guaranteeing no jobs are lost.)
2) After message retrieval, the Worker Daemon processes the message and downloads any required files from S3 to an input directory on the local filesystem. (See Figure 2. Individual steps are indicated in square brackets [ ].)

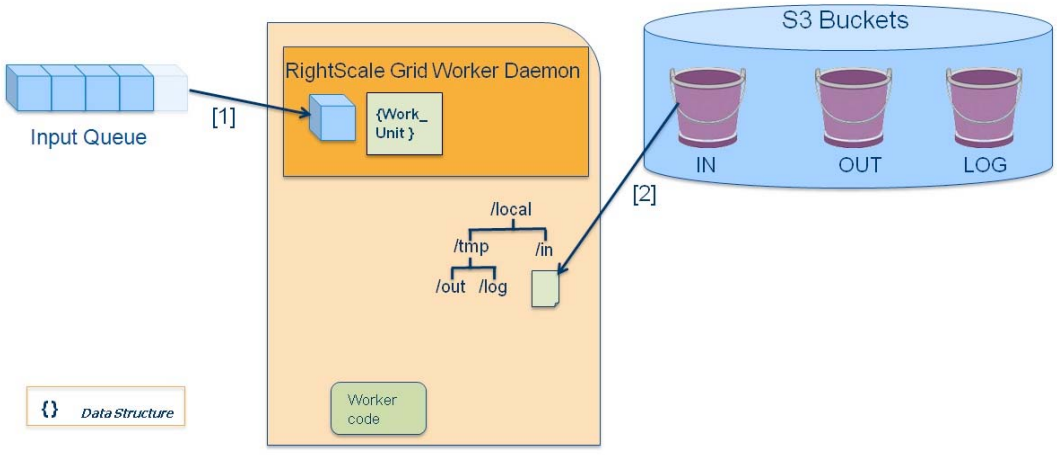

Figure 2 – Message and input file retrieval by the Worker Daemon. The first message in the queue is copied by the Worker Daemon and rendered invisible in the Input Queue.

3) The Worker Daemon then creates the necessary data structures from the message and passes the required inputs to the worker code (See Figure 3.)

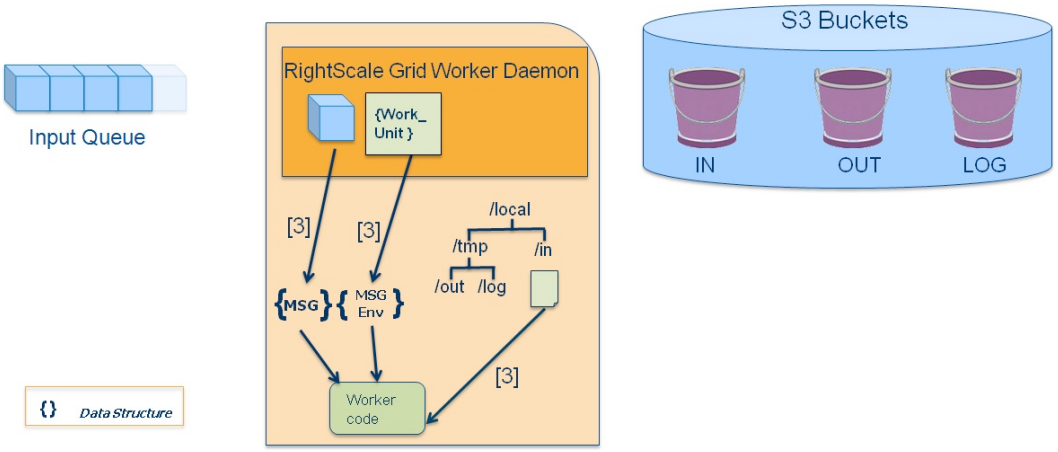

Figure 3 – Worker Daemon passes necessary inputs to the worker code on the instance

4) The worker code processes the job, and places any result files in an output directory on the local filesystem, while any log files are placed in a local log directory.

5) The worker code returns a data structure containing information about the results of the job to the RightScale Grid Worker Daemon. (See Figure 4.)

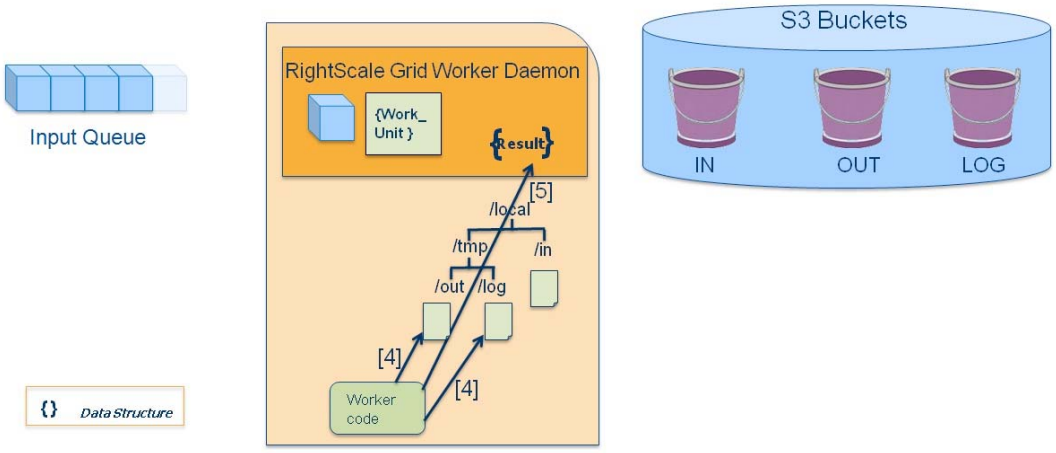

Figure 4 – Worker code has completed job, created output files, and passed results to Worker Daemon

At this point in the workflow, the job has been processed by the worker code, and the resulting data structure has been passed to the RightScale Worker Daemon, which is the daemon's indication that the results-processing portion of the workflow can now commence. At this point, the Worker Daemon performs the following steps:

6) The output files are uploaded to the appropriate S3 bucket.

7) Any log files created during the processing of the job are uploaded to S3.

8) Input and temporary files are deleted from the local filesystem (input files can be left on the local filesystem via a configuration file option to the RightScale Grid Worker Daemon). (See Figure 5.)

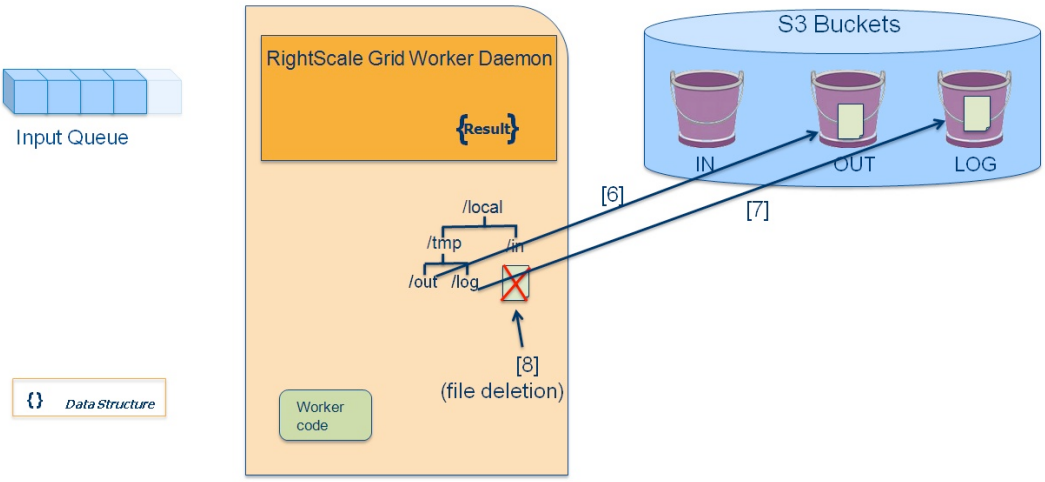

Figure 5 – Worker Daemon processes files in local filesystem for completed job

9) If the job completed successfully, a results message is placed in the SQS Output Queue, and (optionally) results information is posted to a user-specified web server. If the job resulted in an error, a message is placed in the SQS Error Queue.

10) Statistics gathered during job execution are posted to the SQS Audit Queue.

11) The original message is now deleted from the SQS Input Queue. (See Figure 6.)

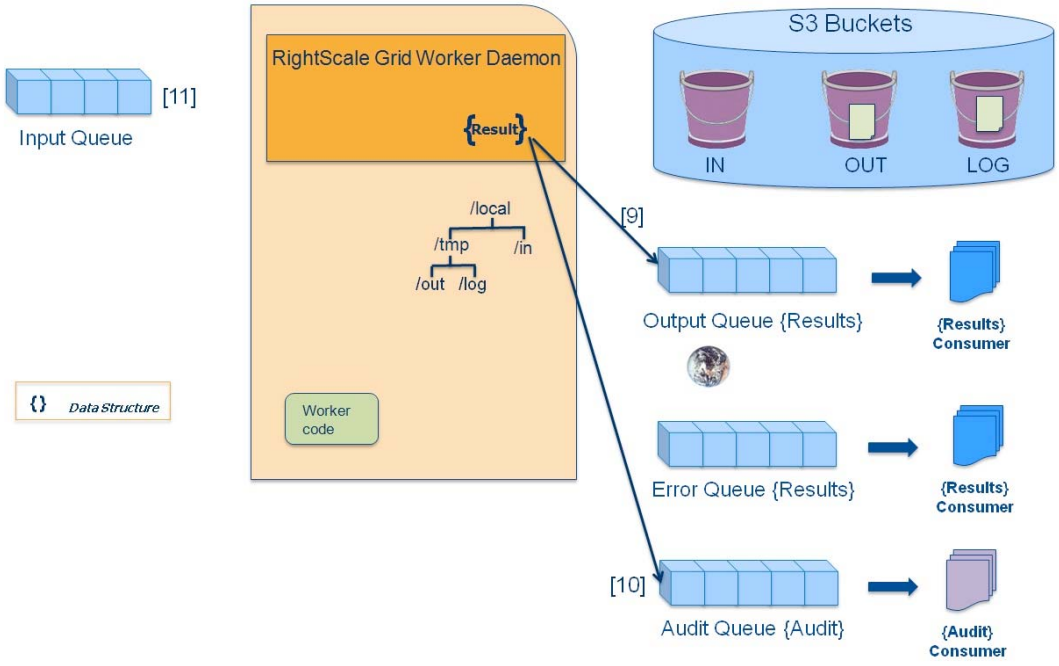

Figure 6 – Worker Daemon has placed results in appropriate queues, and removed completed job from Input Queue.

As mentioned in step 1 above, the original message is not deleted from the input queue (step 11) until the entire process completes successfully. The visibility of a message can be changed via a setting in the RightScale Grid Worker Daemon configuration file. For complex jobs that may take a long time to process, this setting should be configured to a value much greater than the anticipated maximum job processing time. Currently, the maximum setting for the visibility timeout is 12 hours.

Shown on the right side of Figure 1 is the Job Consumer. This is an optional component of the workflow, but one which is typically implemented. The Job Consumer is not necessary in configurations that utilize grid chaining, for in these cases the output queue of one workflow functions as the input queue of a separate workflow. In the deployment created via the RightScale Grid macro, the Job Consumer is implemented via three separate applications: a Result Consumer that processes the SQS Output Queue, an Audit Consumer that processes messages in the SQS Audit Queue and compiles these statistics into a .csv file for later analysis, and an Error Consumer that processes the SQS Error Queue. These consumer processes are basic examples of the types of tasks that the Job Consumer can perform, and are provided as an instructional sample and model for production tasks. As with the Job Producer, multiple Job Consumer processes can run simultaneously and process the same queues.

In the case of grid chaining, the RightScale Grid framework provides the ability to implement independent scaling events for each of the automated workflows.

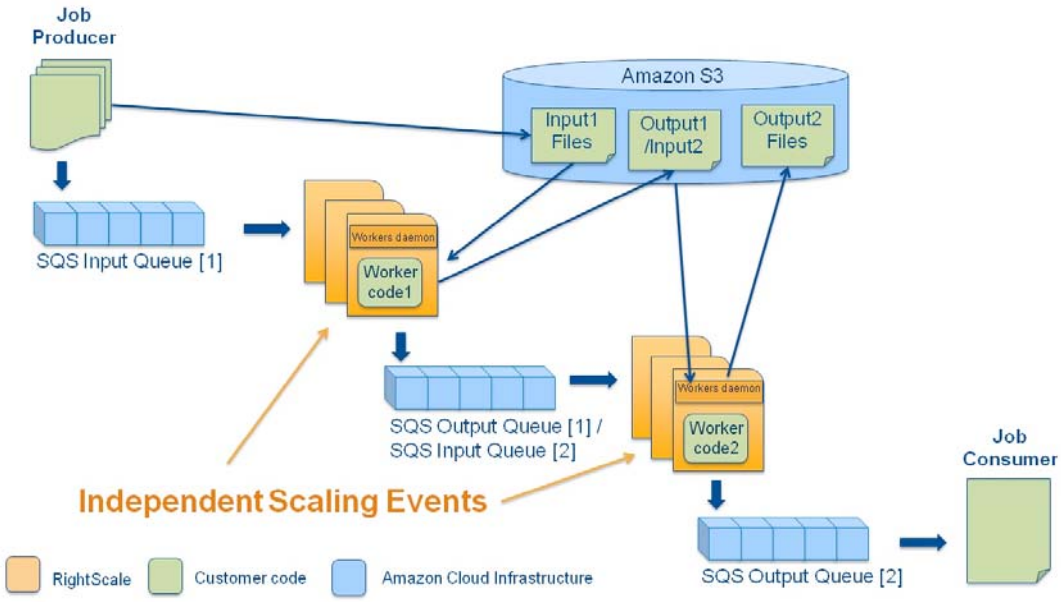

Figure 7 – Simple Chaining of Grid Computing Processes

As shown in Figure 7, the worker instances associated with "Worker code 1" can autoscale independently of the worker instances running "Worker code 2". This can be very useful in situations in which specific processes in multiple workflows require additional compute cycles and thus take longer to complete. To prevent jobs from backing up in one queue while a second input queue sits idle, different scaling metrics can be associated with different queues. This allows additional instances to be utilized for the more time-consuming jobs, thus keeping the output flowing at a rate reasonable with respect to the input of the next workflow in the chain.

4 Summary and Conclusions

With the ongoing maturation of the cloud, and with tools such as RightScale Grid available to take advantage of the elasticity and transient nature of servers in the cloud, the opportunities and possibilities of grid computing and batch processing solutions continue to thrive. The wide array of grid computing options provide solutions for a vast range of applications, as several of the use cases outlined previously have shown. There are grid processing opportunities for the smallest Web 2.0 customer that has a new offering that goes viral, all the way to the large enterprise-scale organizations that have batch jobs requiring massive processing power to handle vast data loads. The RightScale Grid framework provides an easy on-road for customers with applications that lend themselves to the grid processing paradigm. The automation inherent in RightScale Grid can enable a new grid processing application to be up and running in the cloud quickly and efficiently. While this paper has presented an overview of the features and functions of the RightScale Grid framework, the best way to understand and appreciate the simplicity that it brings to implementing a grid computing solution is through individual use and experimentation. The aforementioned RightScale Grid macro is an ideal way to quickly build a functioning grid processing deployment and obtain access to all the moving parts necessary to get a more complete understanding of the framework, and how it can meet your grid computing needs. To see a demonstration of RightScale Grid in action, please contact the RightScale Sales team at sales@rightscale.com.