CloudEx: a novel cloud-based task execution framework

CloudEx: A Novel Cloud-based Task Execution Framework

Omer Dawelbeit
School of Systems Engineering,
University of Reading,
United Kingdom.
Email: o.i.o.dawelbeit@pgr.reading.ac.uk

Rachel McCrindle
School of Systems Engineering,
University of Reading,
United Kingdom.
Email: r.j.mccrindle@reading.ac.uk

***Abstract*—In recent years cloud computing has seen steady adoption due to its unique features such as elasticity, fault-tolerance and utility billing. Cloud computing Infrastructure-as-a-Service (IaaS) enables unique architectures that can dynamically scale and configure computing resources from a catalogue of available features. In addition to provisioning long running homogeneous clusters of Virtual Machines (VMs), it can also be feasible to provision ephemeral and heterogeneous per-job VMs. This is made possible due to the reduced VM startup time and per-minute billing for cloud VMs. In this paper we design and implement CloudEx, a generic and novel framework for executing jobs on public clouds by leveraging the Google Cloud Platform. CloudEx enables users to split jobs into a sequence of smaller tasks that can be distributed using Bin Packing or user-defined algorithm. Additionally, users can specify the VM specification per job or per task, CloudEx then provisions the required VMs, coordinates the job execution and terminates these VMs once the job is completed.**

I. Introduction

Public clouds, most notably Amazon Web Services (AWS) [1], Google Cloud Platform (GCP) [2] and Microsoft Azure [3] offer Infrastructure-as-a-Service (IaaS) such as Virtual Machines (VMs). Virtual Machines can be provisioned from a catalogue of predefined machine types in terms of CPU cores and memory. Additionally, with the reduced VM startup time and the per-minute billing model adopted by both GCP and Azure, it is now feasible to dynamically provision heterogeneous per-job cloud VMs which are terminated once the job is completed. This approach provides the ability to tailor the computing resources to fit the current job rather than the traditional cluster approach of tailoring jobs to fit the computing resources available.

In this paper we outline the design and implementation of CloudEx[1], an open source framework for executing jobs on public clouds by leveraging the Google Cloud Platform and in particular Google Compute Engine (GCE) [4]. CloudEx enables users to split jobs into a sequence of smaller tasks that can be distributed using Bin Packing or user-defined algorithm. Additionally, users can specify the VM specification per job or per task, CloudEx then provisions the required VMs, coordinates the job execution and terminates these VMs once the job is completed. In this paper we show that the average VM startup time on GCE is 85 seconds. Additionally we show that when acquiring VMs for short-running jobs, per-minute billing can save hundreds of hours compared to per-hour billing.

The rest of this paper is organised as follows: Section II briefly introduces Google Compute Engine, then Section III describes the high level architecture of CloudEx. Section IV explains how CloudEx jobs can be defined, followed by an explanation of the approach used for handling tasks input and output in Section V. The approach used for distributing the execution of CloudEx tasks is explained in Section VI. Subsequently, Section VII covers the implementation of CloudEx and experiments, followed by a brief review of related work in Section VIII. Finally, Section IX concludes this paper and highlights possible future work.

II. Google Compute Engine

Google Compute Engine (GCE) enables users to create and manage Virtual Machines (VMs), known as *instances*, on the Google infrastructure. VMs are defined under a project and are created in a particular zone, which specifies the region and data centre used to host the VM. GCE provides a number of machine types [5] for launching VMs, these are categorised into shared-core, standard, high-memory, high-CPU and custom categories. Once a VM is started it can query the GCE metadata server for metadata information [6] such as VM ID (instance ID), hostname and user defined metadata. GCE VMs are charged per-minute based on their CPU and memory configurations, with an initial 10 minutes charge.

III. High Level Architecture

The high level architecture of the CloudEx framework is shown in Figure 1, this architecture is based on three major components, 1) Virtual Machines (VMs), 2) cloud services including databases and storage and 3) the CloudEx framework components. A coordinator VM is used to start, manage and terminate a number of ephemeral VMs called processors. For each job, the coordinator will start the number of required processors then distribute the workload between them and subsequently terminate these processors once the job is completed. The coordinator coordinates the processors by issuing simple

[1] http://cloudex.io

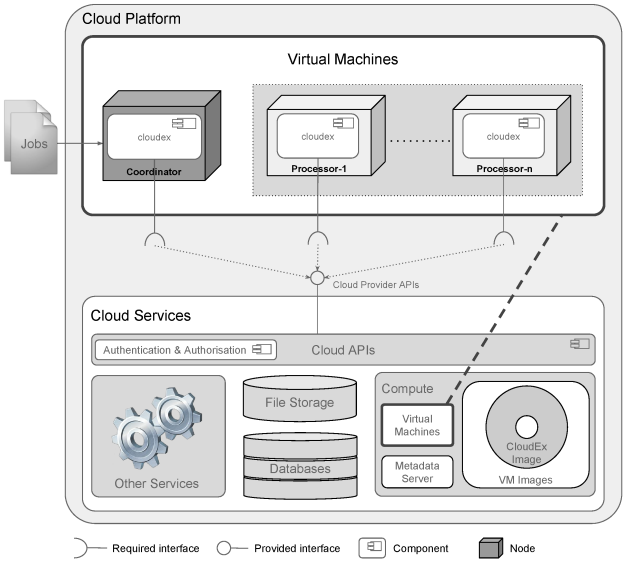

Fig. 1. CloudEx High Level Architecture.

commands in the form of key/value pairs using the *metadata* server.

A. Key CloudEx Definitions

This section introduces some of the key CloudEx definitions that will be used throughout the rest of this paper. CloudEx VMs are divided into two categories:

Definition 1: **Coordinator** (C), is a long lived VM that is required to stay on for as long as the system is running. A coordinator distribute task execution between a number of processors using a workload partitioning function f.

Definition 2: **Processors** (P), are short lived or ephemeral VMs that will be started by a coordinator to perform particular tasks and then be terminated once the tasks are done. Processors do not communicate with each other.

CloudEx work is divided into logical blocks of functionality that is implemented as a subroutine in a computer program, each logical block is called a task, and can be defined as follow:

Definition 3: **Tasks** (T), are user-defined subroutines that process a number of workload items (files, database rows, etc.) and need to be executed in either a coordinator or a number of processors.

Tasks that need to be executed by the coordinator are simply referred to as coordinator tasks. Coordinator tasks are mainly responsible for aggregation activities and are neither processing nor memory intensive. Similarly, tasks that need to be executed by processors are referred to as processor tasks. Processor tasks are either processing or memory intensive and require processors to be scaled horizontally or vertically depending on the nature of the task. Users can specify the VM requirements for each task.

A computational job can be divided into CloudEx tasks that are executed separately using a divide-conquer approach. The tasks to be executed by the CloudEx framework are grouped in a CloudEx job (j), which defines how to orchestrate and execute these tasks.

Definition 4: **Job** (j), is a definition of CloudEx tasks and how they should be executed. A job also defines some initial data in the form of key/value pairs to be used as input to the task subroutines.

During job execution, the coordinator maintains an in-memory key/value pairs store called the *job context*. Input for tasks can populated from the job context and output saved back to it. Each processor task has a workload partitioning function f which is used by the coordinator to partition the workload for the task between a number of processors and can be defined as follows:

Definition 5: **Workload partitioning function** (f), is the function ($f : W \rightarrow P$) that maps a set of workload items W to a set of processors P.

The CloudEx framework is implemented as an application component (Figure 1) that is run on both the coordinator and processor. This application can be defined as follows:

Definition 6: **CloudEx** (CLX), is an application with a number of subroutines, one for each task and runs on a coordinator or a processor. The application can read and update the processor metadata and interact with other cloud services.

All processors (P) are created from the same VM image and hence contain the same software. The CloudEx application CLX is initially installed on the disk of a template VM and saved as a VM image (VMI).

Definition 7: **Image** (VMI), is a VM image used to create all the processors. This image contains a bootable root file system and the CloudEx application CLX.

When a processor is created from the image VMI, based on the metadata supplied to the processor, the CloudEx application CLX can: 1) decide which task in CLX to execute, 2) set the task input from the metadata, 3) retrieve the workload items assigned to this processor from cloud services (storage, databases, etc.).

B. The Lifecycle of the Coordinator and Processors

Initially, the coordinator is started and provided with a job to execute. The coordinator will iterate over each of the tasks defined in the job, if the task is a coordinator task then the coordinator will execute it. Otherwise for processor tasks, the coordinator will partition the workload between a number of processors using the workload allocation function f. The coordinator then, using image VMI, creates and starts the required number of processors, providing each with metadata that contains the name of the task to execute and its input data. At this point the coordinator waits for all processors to finish executing the task.

Once a processor starts up, the application CLX is run by default. Once running , CLX reads the processor's metadata, based on this metadata it chooses a task to execute and populate its input data. CLX then updates the processor metadata status as *BUSY*. When the task is run by the processor it can interact with the various cloud services such as storage

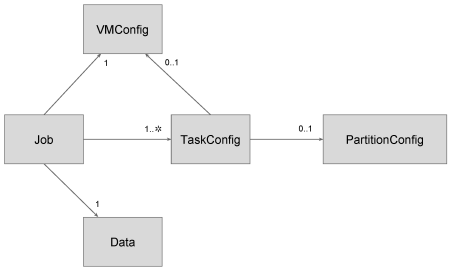

Fig. 2. CloudEx job entities.

and databases to download and process the assigned workload. Once the task is done it uploads its output, if any, to other cloud services. At this point CLX updates the processor metadata status as *FREE*. The processor then waits for further metadata updates from the coordinator.

Once all the processors have updated their metadata status to *FREE*, the coordinator checks if there are anymore tasks to execute. If all the tasks are done, the coordinator will terminate all the processors to avoid incurring unnecessary cost. If there are more tasks to execute then the coordinator will continue to process them. The processors are reused multiple times during job execution to run all the processor tasks. If a particular task requires more processors than the ones already started, or processors with different CPU and memory configurations then the coordinator will start new ones.

IV. Defining Jobs

CloudEx tasks, their input data and processing details are specified using a job definition. The job definition consists of three main parts: 1) Job Data, 2) Virtual Machine Configurations and 3) Tasks Definition. These parts are shown in Figure 2 and explained in detail in the following sections.

A. Job Data

Job data is a collection of initial arbitrary key-value pairs that can be used as input to the various tasks and partitioning functions. Values denote constants or names of cloud services data containers such as tables or buckets; these values are constraint to String and Numeric data types. When the job is executed, this initial data will be added to the job context so that it can be referenced as input to any of the tasks or partitioning functions.

B. Virtual Machine Configurations

Virtual Machine Configurations (abbreviated as VMConfig), is a collection of key-value pairs detailing the attributes required for launching the VM. Keys identify common cloud parameters such as virtual machine type, disk type, etc.... Values provide a specific identifier that relates to the cloud provider being used. The VMConfig is used as the base virtual machine settings when creating CloudEx processors. The following keys are used by CloudEx:

- **Zone**: The zone identifier in which the virtual machine should be deployed.
- **VM image**: The identifier of the virtual machine image VMI to use, which contains the CloudEx application.
- **VM type**: the type of the virtual machine, the value is a cloud provider specific identifier that corresponds to a number of CPU cores and certain amount of main memory.
- **Network**: the identifier of the networking configurations to use for the virtual machine.
- **Disk type**: the type of disk to use, covering options such as magnetic, solid state, ephemeral or persistent disk types.
- **Startup script**: a script to be executed once the virtual machine is started.

C. Tasks Definition

Tasks definition is a list of task configuration (abbreviated as TaskConfig) for the individual tasks to be executed by CloudEx. Each TaskConfig provides details on how to initialise and process the task. A TaskConfig provide details that are applicable to either processor tasks, coordinator tasks or both, these details are summarised as follows:

- **Task subroutine (both)**: a reference to the task subroutine to be executed for the task.
- **Input (both)**: the input data for the task specified as a collection of key-value pairs. A value can either be a constant or a reference that will be resolved from the job context.
- **Target (both)**: indicate which component, coordinator or processor that executes the task.
- **Error action (both)**: an action to be taken if the task subroutine exits due to an error. Two error actions are considered, to ignore the error and continue executing other tasks in the job, or to terminate the job execution.
- **Output (coordinator)**: the names of the output of the task subroutine. The output of the task will be added to the job context using this name as the key and the actual output as the value.
- **VMConfig (processor)**: optionally for processor tasks, a virtual machine configuration can be specified for the execution of the task. This configuration can be statically specified in the task definition or dynamically specified during the job execution. If omitted the VMConfig provided as part of the job definition will be used for processor tasks.
- **Partitioning configuration (processor)**: the TaskConfig for all processor tasks is required to have a partitioning configuration, abbreviated as PartitionConfig.

The PartitionConfig is required for all processor tasks and contains a reference to the subroutine of partitioning function f and its input data, which can be resolved from the job context as will be explained in the next section. The PartitionConfig also includes output keys to be used when adding the output of f to the job context. This PartitionConfig describes how the workload for the task can be distributed between a number of processors.

V. Dealing with Tasks Input and Output

When executing a job, the coordinator maintains an in-memory collection of key-value pairs called the *job context*. The job context is used to share data between the various tasks and partitioning functions. The coordinator tasks can directly accept input from and provide output to the job context. Processor tasks can also accept input from the job context, however, this input is sent remotely to the processors through the metadata server. Partitioning functions for processor tasks are executed by the coordinator and hence have direct access to the job context. Consequently, these functions can directly accept input from and provide output to the job context. When defining tasks, values in the job context can be referenced as input or output for the tasks, by simply referring to them by their keys.

A. Input and Output Resolution

When defining a CloudEx task, its input is defined as either constants or variable names. These variables' names are used as keys to lookup a value from the job context. For example, if a task is defined to have the following key-value pairs as input {`threshold:50, domain:example.com, table:#key1, file:#key2`}. The coordinator will treat all the values as constants, except those prefixed with #, those are treated as variables. The coordinator will remove # prefix and will then lookup `key1` and `key2` in the job context.

The same approach is applied to the input of partitioning functions attached to processor tasks. Additionally, this approach is also used to resolve the input for processor tasks. However, this input is not directly populated by the coordinator, instead it is sent to the processor with the task metadata through the metadata server. Output of coordinator tasks and partitioning functions can also be specified when defining tasks. Once a task or a partitioning function is executed, the coordinator uses these keys to save its output back to the job context. The approach provides the ability to explicitly reference output values in the job definition.

VI. Partitioning the Workload

The CloudEx framework provides two mechanisms for users to specify how computationally intensive tasks can be distributed between a number of processors. The number of processors to use for task execution can be determined by using one of these mechanisms: 1) a built-in partitioning function f based on a variation of the Bin Packing algorithm [7], [8] or 2) a user-defined partitioning function. The built-in partitioning function f is explained in detail in the following section.

A. Bin Packing Partitioning

CloudEx provides a built-in workload partitioning function f based on a variation of the Bin Packing algorithm. This function is used to partition the task execution between a number of processors or bins based on the sizes of the workload items W. Workload items W can be a collection of items which can be resolved from the job context, for example a collection of file names with their sizes. To use the built-in function f, one of the following parameters must be provided:

- the number of bins (processors) N to use or
- the bin capacity C and optionally δ (where $0 \leq \delta < 1$) which is the maximum fraction of a bin that needs to be rounded up to a full bin.

1) Calculating The Bin Capacity: Assuming there is a total of M workload items $(w_1, w_2, \cdots, w_M)$ and each has a size of $(s_1, s_2, \cdots, s_M)$ with the largest item having a size of s_K such that $(0 < s_i \leq s_K)$ these items are divided into N bins where the capacity of each bin is C. Two cases are dealt with, the first if the user specifies N, then C is determined as $C = \frac{1}{N}\sum_{i=1}^{M} s_i$, rounding up the results if necessary. In this case the number of bins must be fixed, so each bin is filled up to the maximum capacity C. After all the bins are filled any remaining items are equally spread between the bins. This is achieved by sorting the bins in ascending order and the items in descending order then placing the largest items into the smallest bins, this continues until there are no further items to add.

2) Calculating The Number of Bins: The second case is when the user provides both C and δ, in this case N is determined by first calculating the lower bound number of bins N_L as $N_L = \frac{1}{C}\sum_{i=1}^{M} s_i$. If N_L has a fraction that is greater than δ then it is rounded up to the nearest integer value. However, if this fraction is smaller than δ then it is discarded and the remaining items after filling up all the bins are spread between the bins as explained previously. If the user does not provide a value for C, or if the provided value is less than the largest item s_K, then the size of the largest item s_K is used.

VII. Implementation and Experiments

An implementation of the CloudEx framework was created as a reusable library using the Java programming language. This implementation is open source and is publicly available[2]. The current implementation consists of the high level components shown in Figure 3 and summarised as follows:

- **cloudex-core**: is the generic implementation of the processor and coordinator algorithms and provides a *Coordinator* and *Processor* sub components. Additionally, this component includes the built-in Bin Packing function f explained in Section VI.
- **cloudex-google**: is an implementation that is specific to the Google Cloud Platform (GCE) [2].
- **user-defined-tasks**: is a component of all the user defined tasks subroutines that the CloudEx framework needs to execute.
- **user-defined-partitioning-functions**: is a component of all the user defined partitioning function subroutines that the user can reference when defining processor tasks.

We have used the CloudEx framework to execute, on the Google Cloud Platform, a range of short-running jobs for

[2]https://cloudex.io/

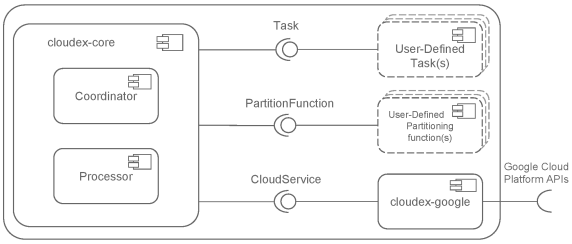

Fig. 3. CloudEx high level components.

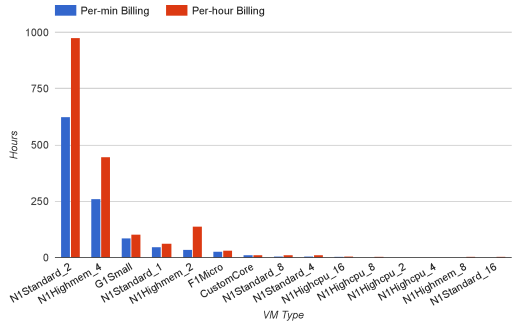

Fig. 4. Comparison of per-min and per-hour billing vs. VM Type.

processing structured data as part of the ECARF project[3]. Over the duration of the project, CloudEx has acquired 1,086 VMs at a total cost of $290.29. The CloudEx framework successfully acquired and sent instructions to processors using the Google Compute Engine APIs. The number of acquired processors ranged from 1 to 16 with various memory and CPU cores configurations. The average startup time taken to acquire CloudEx processors is 85 seconds. This is the time taken for the processor to be created, started and runs the CloudEx application.

Google Compute Engine charges per-minute, consequently, we have also compared the total cost incurred by the various VM types on per-minute versus per-hour billing as shown in Figure 4. With CloudEx processors utilised for less than an hour, the per-minute billing model has saved 697 hours compared to per-hour billing. This is because in per-hour billing if a VM is terminated before reaching a full hour of operation, the usage is rounded up to the nearest hour.

VIII. Related Work

A number of approaches have focused on improving and extending existing cluster management and job scheduling frameworks to enable migration and interoperability with the cloud [9], [10], [11], [12]. These approaches are mainly focused on extending and integrating with existing cluster resource managers such as Torque[4] and job schedulers such as Moab[5] to be able to create cloud based elastic compute clusters.

A large scale elastic environment for scientific computing with recontextualization was presented by Marshall et al. [12]. This approach integrates with Torque and utilises an elastic resource manager, which consists of three major components, a component to read submitted jobs from a queue, a decision engine and a provisioner that interacts with the Cloud provider APIs. An Elastic Cloud Computing Cluster (EC3) tool is developed by Caballer et al. [10]. EC3 creates elastic virtual clusters on top of a number of cloud providers (Amazon EC2, OpenStack and OpenNebula) and integrates with existing resource management systems. The clusters in EC3 are self managed with the ability to scale up or down depending on a predefined policy.

IX. Conclusion and Future Work

In this paper we have presented the architecture of CloudEx, a novel and generic task execution framework that can be implemented on any cloud provider IaaS. Additionally, we have presented a workload partitioning approach based on the bin-packing algorithm to distribute the processing of tasks between a number of processors. As future work we plan to improve the CloudEx framework by providing resilience for the coordinator and adding the capability to autoscale the processors based on particular cost and deadline constraints. Additionally we plan to provide implementations for other prominent cloud providers such as Amazon Web Services and Microsoft Azure.

[3]http://ecarf.io

[4]http://www.adaptivecomputing.com/products/open-source/torque/

[5]http://www.adaptivecomputing.com/products/cloud-products/moab-cloud-suite/

References

[1] I. Amazon Web Services, "Amazon web services (aws) - cloud computing services," https://aws.amazon.com/, 2015, [Online; accessed 13-December-2015].

[2] G. Developers, "Google cloud computing, hosting services & apis," https://cloud.google.com/, 2015, [Online; accessed 13-December-2015].

[3] Azure.microsoft.com, "Microsoft azure: Cloud computing platform & services," https://azure.microsoft.com/, 2015, [Online; accessed 13-December-2015].

[4] G. Developers, "Compute engine - iaas," https://cloud.google.com/products/compute-engine/, 2015, [Online; accessed 13-December-2015].

[5] G. Developers, "Machine Types - Compute Engine – Google Cloud Platform," https://cloud.google.com/compute/docs/machine-types, 2015, [Online; accessed 13-December-2015].

[6] G. Developers, "Storing and retrieving instance metadata," https://cloud.google.com/compute/docs/metadata#waitforchange, 2015, [Online; accessed 13-December-2015].

[7] E. G. J. Coffman et al., "Approximation Algorithms for Bin Packing: A Survey," *Approximation Algorithms*, pp. 1–53, 1996.

[8] E. G. Coffman, Jr. et al., "An Application of Bin-Packing to Multiprocessor Scheduling," *SIAM Journal on Computing*, vol. 7, no. 1, pp. 1–17, 1978.

[9] C. D. Alfonso et al., "Infrastructure Deployment Over the Cloud," *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pp. 517–521, 2011.

[10] M. Caballer et al., "EC3: Elastic Cloud Computing Cluster," *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1341–1351, dec 2013.

[11] M. Caballer et al., "Dynamic Management of Virtual Infrastructures," *Journal of Grid Computing*, pp. 53–70, 2014.

[12] P. Marshall et al., "Architecting a Large-scale Elastic Environment-Recontextualization and Adaptive Cloud Services for Scientific Computing." *ICSOFT*, 2012.