ShuttleService™

Scalable Big Data Processing Utilizing Cloud Structures

A *Tick Data Custom Data Solutions Group* Case Study

Robert Fenster, Senior Engineer and AWS Certified Solutions Architect
Neal Falkenberry, President

June 2015

Tick Data Custom Data Solutions™

↳ Data Processing Services ("ShuttleService™)

ABC Manufacturing, Inc.

ABC, Inc. ("ABC") is a company that generates terabytes of real-time, asynchronous data describing inputs and outputs to its business process. Tick Data was asked by the company to parse this data and create one second summary statistics (as defined by ABC) for later in-house use in optimization of their processes. For reference to financial market data participants, the process is similar to parsing through trade and quote data to build one second summaries that include open, high(max), low(min), close, VWAP, spread, etc. In ABC's case the calculations were standard statistical measures but could have been proprietary algorithms protected in executables, dlls or other methods.

This paper will describe the methodology deployed by Tick Data in this assignment. In this case ABC did not have an existing Amazon AWS infrastructure. ABC's data was transferred to Tick Data on physical media, uploaded to Tick Data's AWS infrastructure, processed, output transferred to ABC, and original input data deleted. Field names in the original data were masked to further protect ABC's data and processes. At no point did ownership of the data transfer to Tick Data.

Turning Terabytes of Raw Data Into Gigabtyes of Information

Introducing Tick Data's ShuttleService™

ShuttleService™ refers to Tick Data's Map/Reduce process whereby data is divided into smaller blocks, distributed across Amazon EC2 instances for processing, with resulting output assembled and aggregated to a common location. The intelligence behind ShuttleService™ is the ability to distribute this processing across any number of CPUs where each CPU initializes itself, receives instructions for its portion of the processing, is monitored, and shuts down upon completion in order to minimize computing costs. The process makes effective use of Amazon spot pricing for CPU power to further manage costs.

This paper describes ShuttleService™ and documents the ABC project in detail.

Contents

Introduction

Tick Data, Inc. migrated to Amazon Web Services (AWS) from a traditional data center in 2013. As the leader in cloud data services, Amazon has created a bottom layer IT infrastructure that can be leveraged using AWS constructs to simultaneously process vast quantities of data quickly, cheaply and therefore efficiently. Tick Data performs all of its daily processing on millions of global financial market records within this environment and distributes that data to customers around the world.

ShuttleService™ evolved from the need to process and customize multi-terabyte data sets for customers. It leverages AWS's Elastic Compute Cloud (EC2), Elastic Block Store (EBS), Simple Queue Service (SQS), CloudWatch, Load Configurations, and Auto Scaling Service to help break large workloads into smaller tasks that can be independently processed across multiple EC2 instances. In short, ShuttleService™ is the commercialization of Tick Data's internal distributed processing plant made available for large data sets across many different industries.

ShuttleService™ can be tuned to deliver data quickly by expanding the size of the processing cluster, made cost effective by using spot EC2 processing power over reserved power, and many other configurable tradeoffs related to the time and cost/benefit of parsing data. In the case of ABC, the company was interested in a cost effective solution (spot over reserved) where approx. 50tb of data would be processed over 5 days.

Overview

ShuttleService™ utilizes a producer/consumer model. The producer breaks the workload into small independent tasks. The producer creates a right-sized EBS volume and attaches that volume to the producer's EC2 instance. For ABC's assignment the producer created 100 gb volumes with attention paid to edge effects around break points in the data. The producer process moves the work object onto the EBS volume. Once the 100 gb data sub-sets are stored on the EBS volume, the EBS volume is then un-mounted and detached from the producer's EC2 instance. The volume id (primary identification information) is then pushed on the SQS work queue.

An AWS CloudWatch™ alarm monitors the depth of the SQS work queue. If the queue depth exceeds a threshold, the Auto Scaling Service triggers the creation of consumer(s) to process the data. Consumers are only created when there is a workload. Hence, ABC was not paying for processing power until needed. The size of the consumer fleet is controlled by the auto scaling service to ensure it is neither too small (work sits on the queue, not being processed) nor to large (running instances with no work to process).

A consumer scans the queue and pops a message from the work queue. It attaches the volume to the consumer instance, mounts the file system and begins to process the work items.

The consumer reads the queue in such a way that the message (volume id) is not visible to other consumers. If the message is not processed in a configured time frame, the message

will return to the queue for other consumers to see. Example: if the consumer instance is terminated during processing, the message will return to the queue for continued processing.

When the consumer completes its task it places the completed work product back onto the EBS volume, in a completed area, un-mounts and detaches the volume. The queue message is then permanently removed from the work queue and placed on a completed queue.

Depending on configuration, the consumer will then either go back to the work queue for additional work, or self-terminate. It is this ability to self-terminate that enables ShuttleService™ to minimize computing costs.

What Data Is Being Processed?

Tick Data coded ABC's statistical routine in Java. The code was tested thoroughly and accepted by ABC prior to production. ABC could have provided its own script, an executable, or a callable library that encapsulated its IP for processing. ABC sought to have the real-time, millisecond time stamped data aggregated into one second intervals. Standard statistical measures within each second were also calculated. Field names were masked to further obfuscate the data that remained at all times the property of ABC. The resulting output contained 29 fields and appeared as (partial):

Date, Time, Field1 Avg , Field1 Max, Field 1 Min, Field 1 StDev, Field 1 Mode, ….

ABC then used this output for internal optimization work. Tick Data, Inc. was not privy to that work nor need be. In fact, we never knew the type of data that was being processed.

Additional Detail

Getting Started

The first step in the implementation was to define the producer that breaks down work items, and the consumer that processes the work items. Company ABC provided Tick Data with a basis for the smallest work item and the library to process the work items. Additionally, ABC provided estimates for how fast the library could complete a work item. Tick Data used the information provided on work item size and processing time to determine the optimal amount of data to place on each shuttle.

Selecting the Right Instance Type and Size

Care should be made to test the producer and consumer against different instance types and sizes. I/O bandwidth, EBS optimization, memory, vCPUs and cost must be considered. The instance type for producer and the consumer have different considerations.

For the producer, the focus should be Instance Type and Size that supports high I/O bandwidth and EBS optimization. When considering the AMI for the producer, pv (paravirtual) vs. hvm also need to be considered. pv instances can support up to /dev/sd[f-p][1-6] (11*16=176) volumes. hvm instances can only support /dev/sd[f-p] (11) volumes.

To best support the ShutteService™, the producer should be able to generate more than one shuttle at a time, and therefore would need 1+ devices to add attach a volume.

For the consumer, processing power and memory may be more important that high I/O bandwidth. Additionally, while there are one or maybe two producers, the number of consumers could be a very high number. Ephemeral drives (Volatile Instance Stores) may also be leveraged on the consumers to gain speed advantages. Finally, consumers could leverage spot instances to reduce overall processing costs. In this case, messages must fall back to the queue if an instance is terminated.

Select	Name	vCPU	Memory (GiB)	Instance Storage (GB)	I/O	EBS Opt.	On-Demand Hourly Cost
◉	t1.micro	1	0.6	--	Very Low	--	$0.020
○	t2.micro	1	1.0	--	Low	--	$0.013
○	t2.small	1	2.0	--	Low	--	$0.026
○	t2.medium	2	4.0	--	Low	--	$0.052
○	m3.medium	1	3.7	SSD 1 x 4	Moderate	--	$0.070
○	m3.large	2	7.5	SSD 1 x 32	Moderate	--	$0.140
○	m3.xlarge	4	15.0	SSD 2 x 40	High	Yes	$0.280
○	m3.2xlarge	8	30.0	SSD 2 x 80	High	Yes	$0.560
○	c4.large	2	3.7	--	Moderate	Yes	$0.116
○	c4.xlarge	4	7.5	--	Moderate	Yes	$0.232
○	c4.2xlarge	8	15.0	--	High	Yes	$0.464
○	c4.4xlarge	16	30.0	--	High	Yes	$0.928
○	c4.8xlarge	36	60.0	--	Very High	Yes	$1.856
○	c3.large	2	3.7	SSD 2 x 16	Moderate	--	$0.105
○	c3.xlarge	4	7.5	SSD 2 x 40	Moderate	Yes	$0.210
○	c3.2xlarge	8	15.0	SSD 2 x 80	High	Yes	$0.420
○	c3.4xlarge	16	30.0	SSD 2 x 160	High	Yes	$0.840
○	c3.8xlarge	32	60.0	SSD 2 x 320	Very High	--	$1.680
○	g2.2xlarge	8	15.0	SSD 1 x 60	High	Yes	$0.650
○	g2.8xlarge	32	60.0	SSD 2 x 120	Very High	--	$2.600
○	r3.large	2	15.2	SSD 1 x 32	Moderate	--	$0.175
○	r3.xlarge	4	30.5	SSD 1 x 80	Moderate	Yes	$0.350
○	r3.2xlarge	8	61.0	SSD 1 x 160	High	Yes	$0.700
○	r3.4xlarge	16	122.0	SSD 1 x 320	High	Yes	$1.400
○	r3.8xlarge	32	244.0	SSD 2 x 320	Very High	--	$2.800

Figure 1| EC2 Instance Type and Sizes

Recovering from Failure

To support recovery, the consumer process should work to delete or move the raw data to a processed area on the shuttle once a task has been processed.

As noted, queued volumes that are being processed have a timeout set, such that they will go back to the queue if that timeout has been exceeded. The most likely situation for a timeout to be exceeded if using spot instances is that the instance gets terminated if outbid by another

AWS user. After the timeout, the volume-id will be revisable to other consumers to continue work. If the consumer moves or deletes the raw sub-work unit once completed, the next consumer to pick the volume off the queue will pick-up the work where the previous consumer left off, and not start from the start.

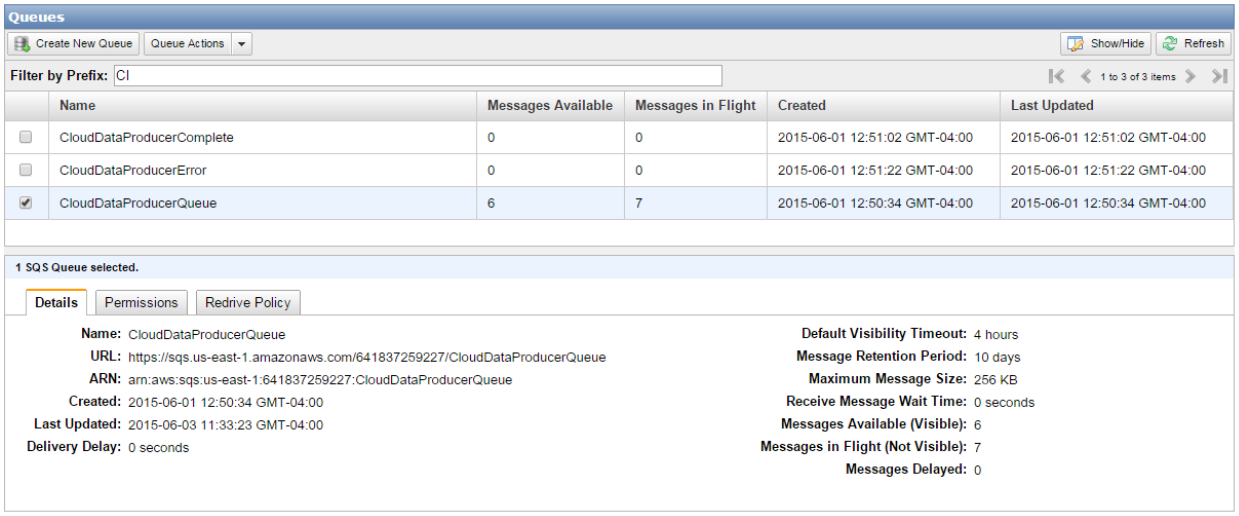

Figure 2 | SQS Window Showing Visibility Timeout

Improving Performance and Cost Efficiency

Spot instances are an exciting way to enhance the cost efficiency of the process. These instances can be found at significantly reduced prices to reserved instances.

Example: List price for a RESERVED c3.8xlarge Amazon Linux AMI: **~$1.68/hr.**

Select	Name	vCPU	Memory (GiB)	Instance Storage (GB)	I/O	EBS Opt.	On-Demand Hourly Cost
◉	c3.8xlarge	32	60.0	SSD 2 x 320	Very High	--	$1.680

Figure 3| Sample On-Demand Price c3.8xlarge

Spot pricing for the same instance type: **~0.33/hr.**

Figure 4| Sample Spot Pricing graph c3.8xlarge

As can be seen the spot price fluctuates. For a spot instance a maximum price to pay per hour is selected. If the current price exceeds the maximum chosen, the instance terminates with no warning. As such, a recovery from failure strategy is essential.

Scaling Up

To support the producer/consumer model, **creating consumers on demand to handle load is essential to the fast processing of data.** Care must be taken in the configuration of the Auto Scaling Service to maximize performance and throughput. When CloudWatch triggers an alarm ("Too many messages in the queue"), the Auto Scaling Service will execute its scale up policy. If the scale up policy deploys "On Demand" instances*, the instances are fulfilled within moments. The instances will start quickly and begin to handle the work load. However, if the scale up policy is set to use "Spot" instances, the policy will request a spot instance. Spot instance requests are then placed in a queue and vetted. If the request is successfully vetted, i.e. the spot price is below the current spot price, the instance will be placed in another queue to be fulfilled. The time between a spot request and a spot instance coming online can be as much as 15 minutes. Because of this delay, it is generally beneficial to request 2 or more instances at a time during a scale up event.

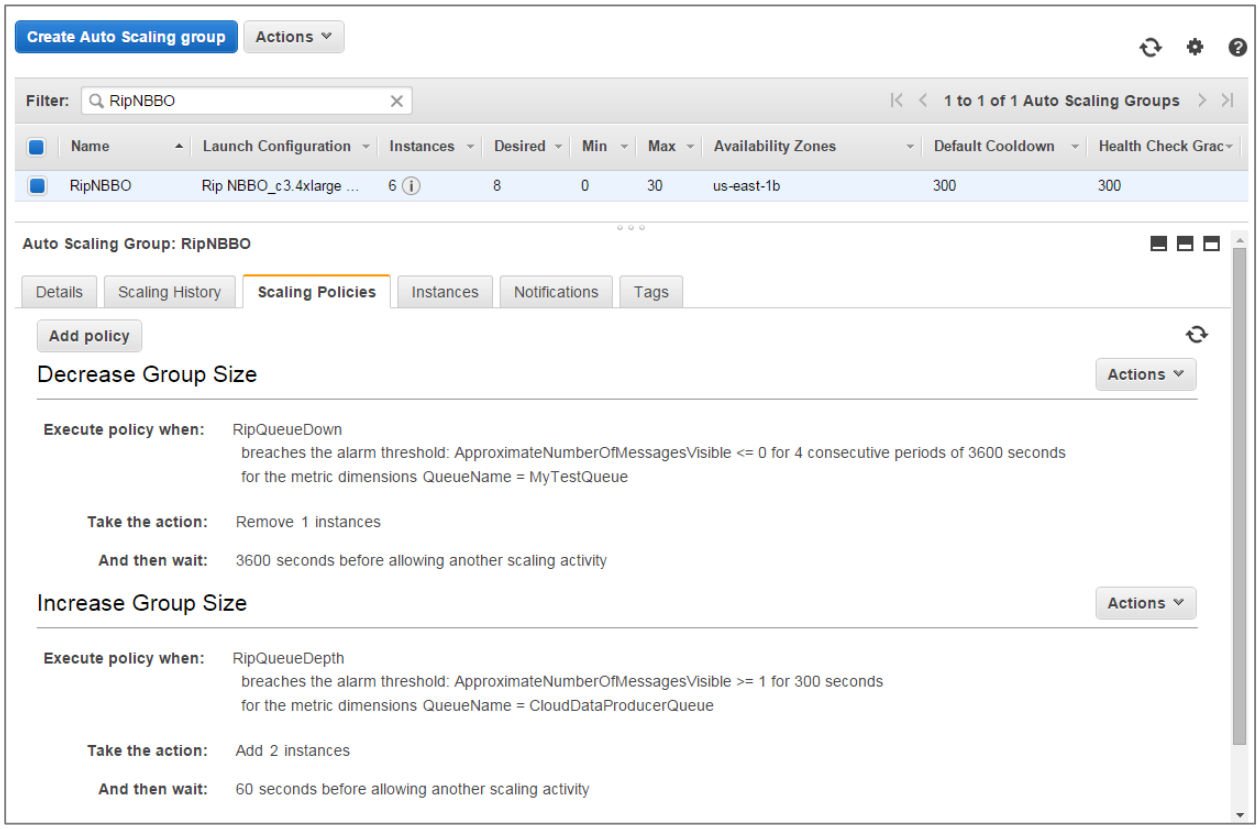

Figure 5| Auto Scaling Service - Policies

Scaling down is also essential. Consumer tasks must be stopped when there is not work load, or they will continue to run. In the above case, when there are no available messages for a 4hr period, the group will begin to scale down. It is proper practice to scale up quickly (in the above situation, start 2 instances after 5 min breach) and scale down slowly. Scaling down too quickly can lead to a bottleneck if there is any gap delay in the producer sending data.

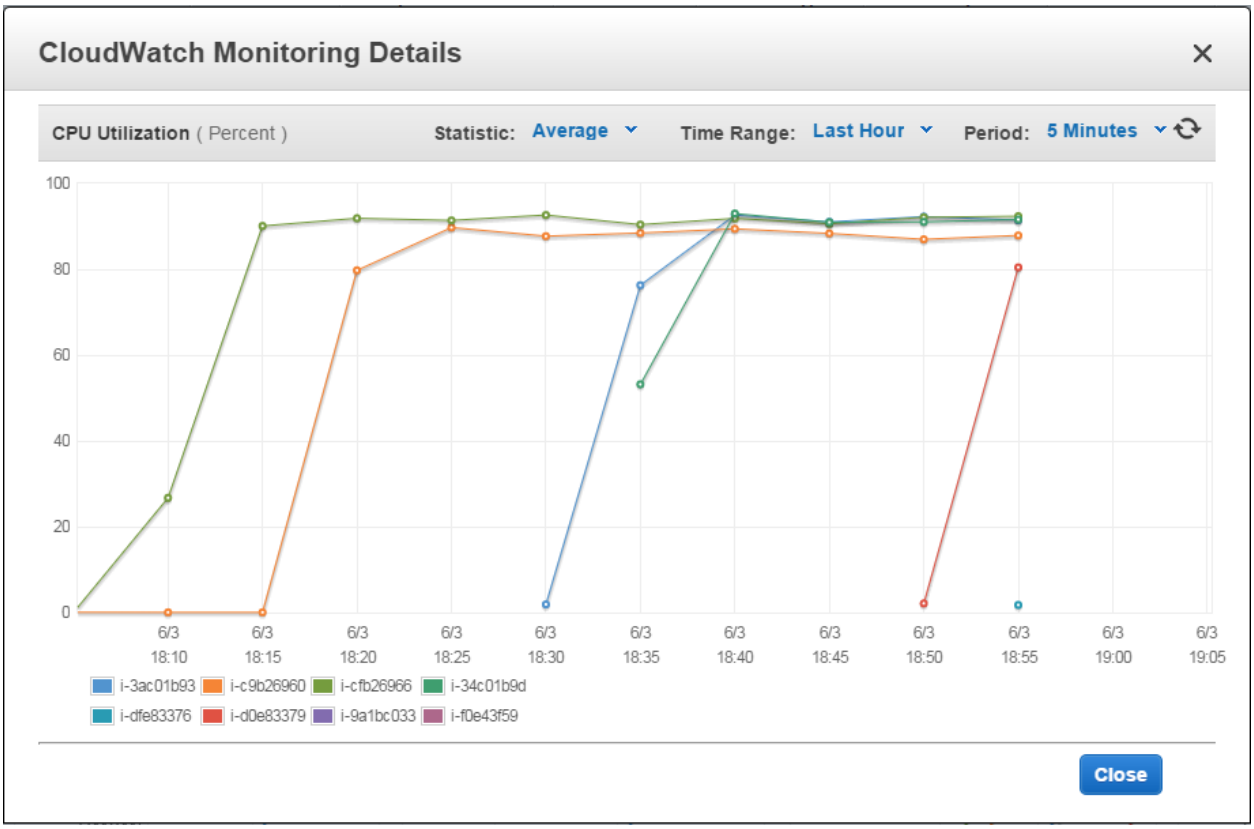

Figure 6| Scaling up for load

Reducing the Risk of Change

Deploying AMI instances for consumers would be the fastest way to get the consumers up and running. The AMI would be a pre-configured instance with all software pre-installed and would start and begin executing quickly. However, the downside of an AMI would be to rework / rebuild and re-test the AMI anytime a change is required. That change could consist of an operating system patch, software change, no matter how small, or other configuration change. To reduce the risks involved with change, the use of user-data in the Launch Configuration to deploy software from S3 onto the instance can mitigate impact of change.

Having the executable code stored in S3, with configuration scripts as well as a deployment script, any changes to any of these modules can be contained to just the module and testing can be done quicker. The Launch Configuration can be setup with a simple user-data script.

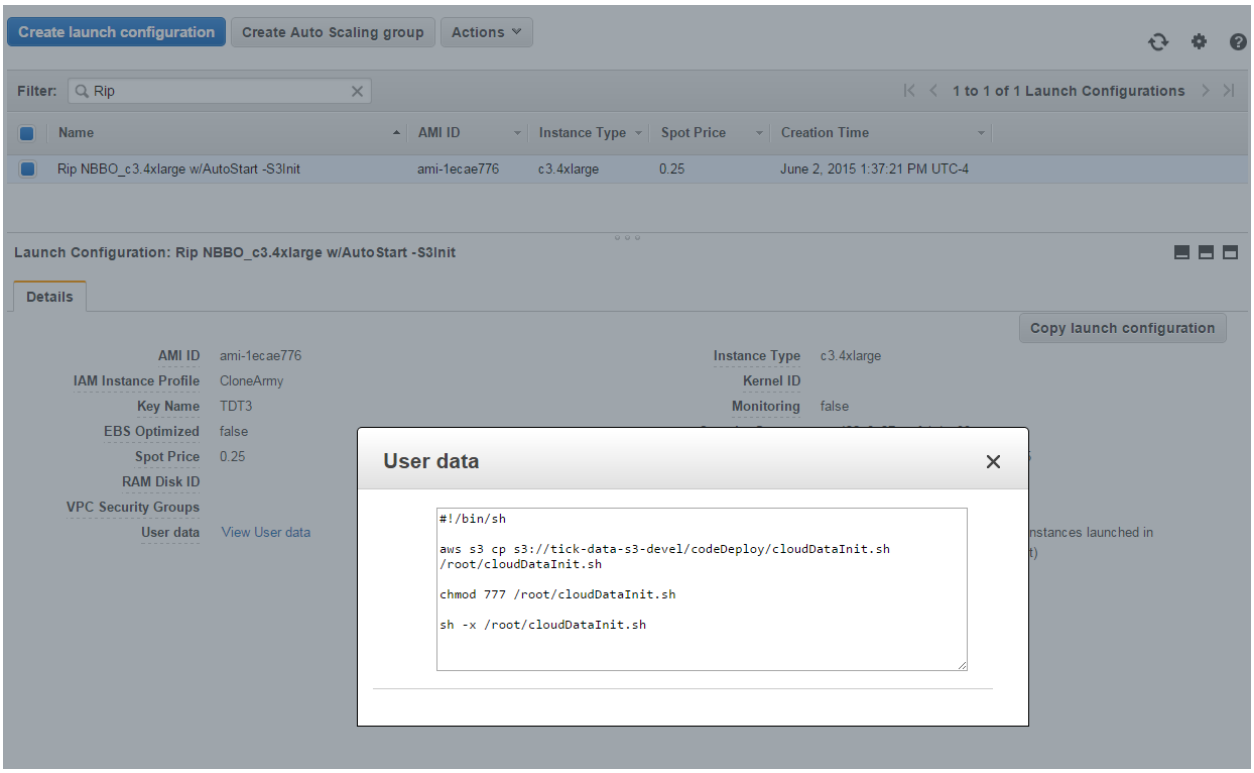

Figure 7|Launch Configuration with User data

Utilizing this strategy, changes only need to be made to the contents of the cloudDataInit.sh script, rather than the AMI and / or the Launch Configuration.

Responsive Design

Tick Data harnessed the power and flexibility of its proprietary task-based architecture. Tasks are individual work items that can be easily chained into profiles. This approach allows for tasks to be individually written and introduced into the process. Tasks additionally support sub-tasks and proper exception handing to allow for clean software development. Simple configuration files allow for complete control of tasks, sub-tasks and variables to be used during execution of tasks.

Monitoring

During execution, Cloud Watch allows for monitoring tools of the solution. Monitoring should be done for both CPU capacities of the consumers as well as disk bandwidth for the volumes.

To ensure the most efficient performance, CPU should be targeted at 90+%. As new consumers come online, they should ramp up to the 90% level quickly and maintain. If the CPU is falling below 70%, it is likely that either the wrong instance size has been

considered, or the consumer task has not been tuned to utilize the processors available. In an example, utilizing c3.4xlarge instances (16 core), consumer CPU usage should reflect figure 8. As the producer places workload onto the queue, the consumers grab it and spin to ~90% immediately.

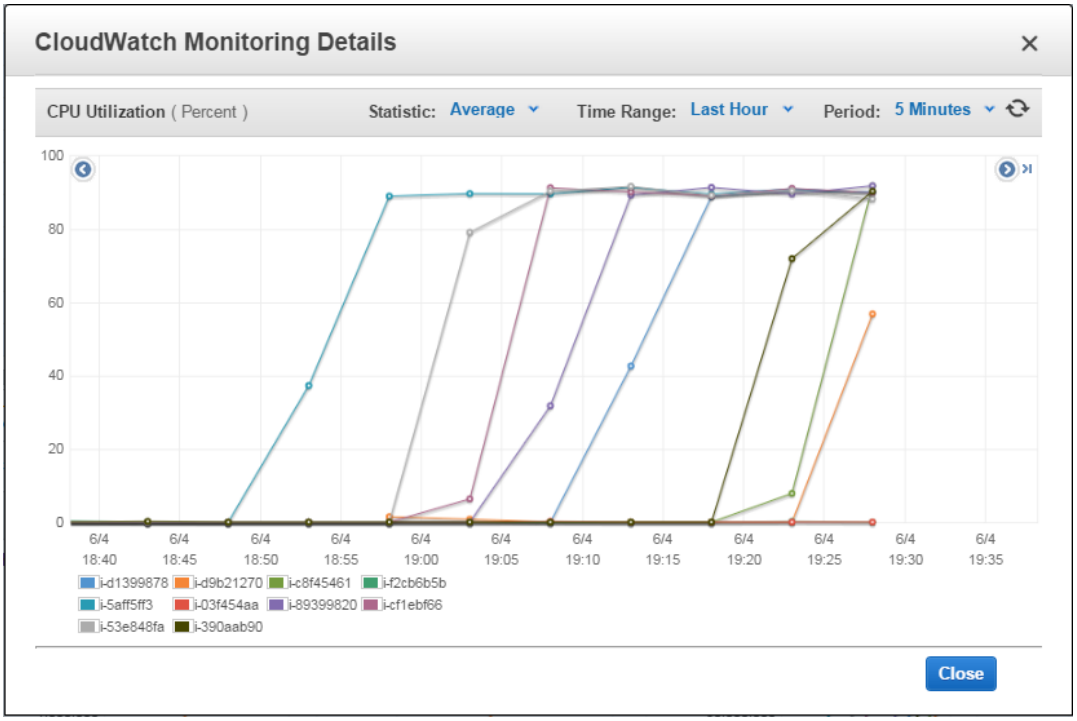

Figure 8 | Consumer CPU Usage

Monitoring of volume metrics, consider the read or write bandwidth and throughput. If the workload is similar, all volumes should be performing at level. Look for abnormalities. They can be a sign of a weak hardware link. If a hardware volume is not performing to speed, the efficacy of the task will be lost to the rate determining step. In these examples, figure 9 shows properly performing volumes, where figure 10 shows a particular volume that is impaired.

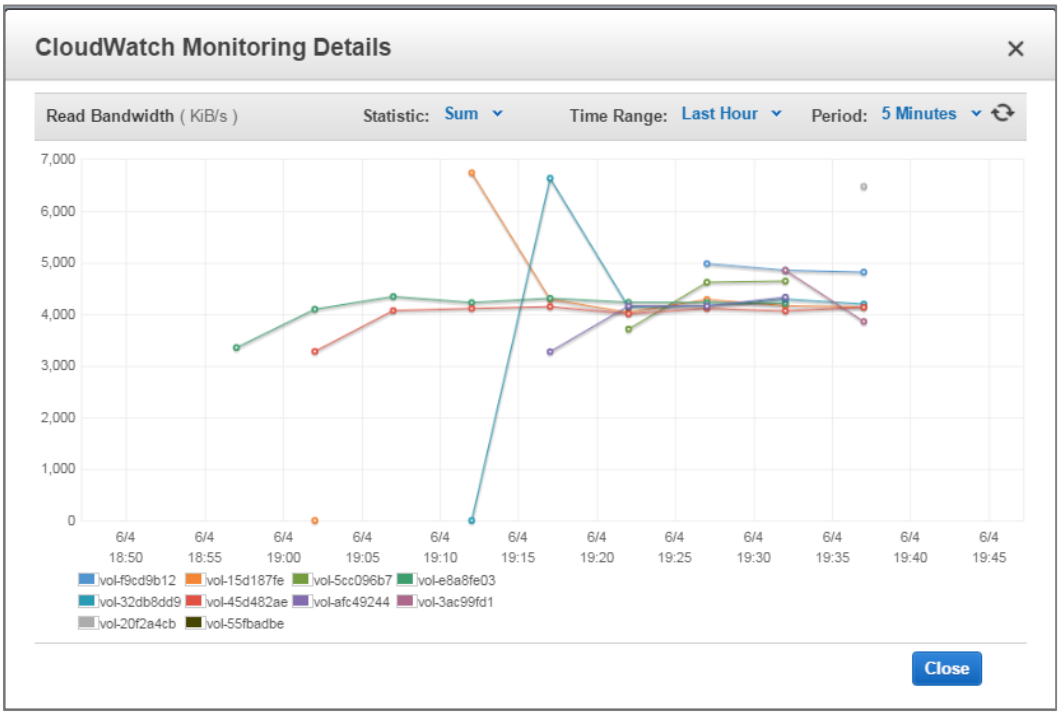

Figure 9 | Healthy volume performance

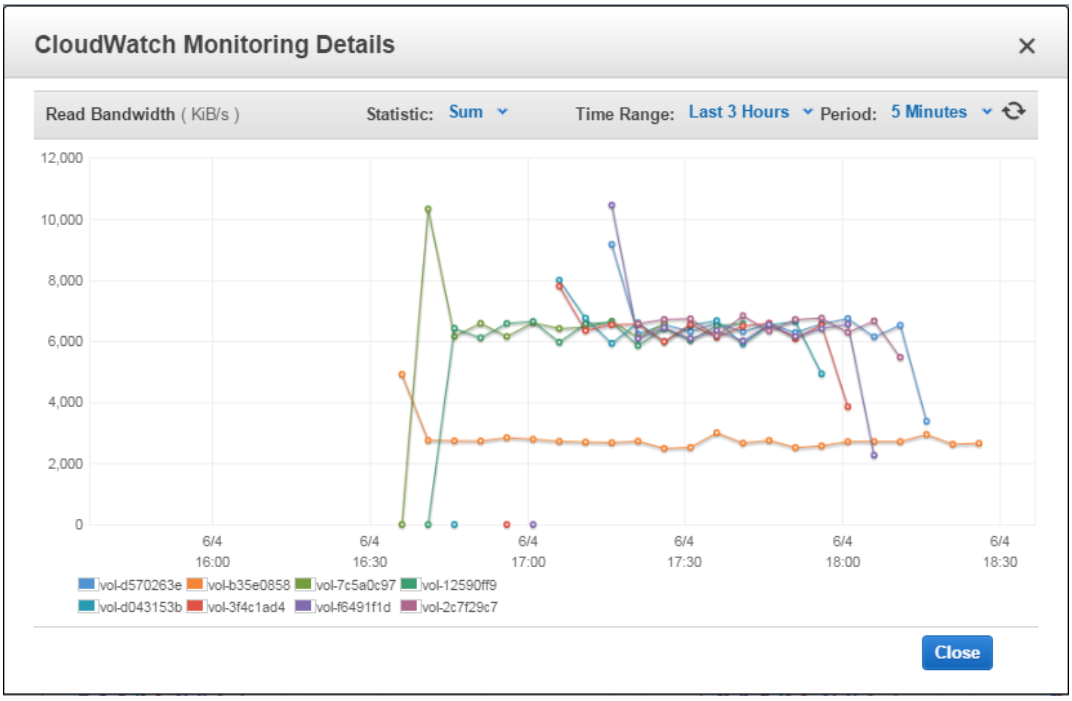

Figure 10 | Single Unhealthy Volume - vol-b35e0858

Costs and Time

ABC required this job be completed in 5 days following receipt of data and approval of data scripts. As such, we benchmarked the time required to parse one tb and scaled the project accordingly. 20 instances were required to accomplish the task. We took advantage of cheap spot market pricing to further reduce costs.

Storage Costs	
EBS Data Storage (50 tb for 2 weeks) *$0.10/tb per month*	$ 2,250
Processing Power	
20 Spot Instances (5 days @ 24 hours each) *$0.17 - $0.25/hr*	$ 500
Data Egress Costs (transfer output data from AWS) 3tb @ $0.09/gb	$ 270
Tick Data Fees	
50 tb Data Processing	$ x
Algorithm Development, Testing, and QA	$ x
Total	**$ x**
Total Cost per terabyte	**$ x/tb**

Summary

Using custom code, Tick Data's extensive data processing libraries, and the AWS bottom layer infrastructure a high level system named ShuttleService™ was configured and built. The system is highlighted by the clever coupling of several subsystems and the application of expert AWS knowledge allowing configuration throughout several different features of the AWS services. Tick Data, Inc. has leveraged all the pieces that Amazon AWS has in place to build a high powered multi-instance / multi-processing data processing and data distribution system. ShuttleService™ helps tie them together into an elegant solution that can be leveraged into an existing production system cost effectively without over taxing the current production systems output and performance. At the same time, it allows for high volume processing and analysis of data in new and different ways to produce a unique view.

In the case of ABC we performed all of the processing in our AWS environment. We could have designed and built an AWS instance for ABC; handing over keys at completion of the data processing for ABC's future use. This decision is largely a function of the expected repetitive nature of the process.

Please contact Tick Data at sales@tickdata.com for additional information on this project or to inquire about Tick Data assisting you in processing large data sets to your specifications.