Financial Services Grid Computing on Amazon Web Services

January, 2016

Notices

Contents

Abstract

Amazon Web Services (AWS) offers customers operating large compute grids a powerful method of running risk and pricing calculations of more scenarios, against larger datasets, in a shorter time and for lower cost. Meeting these goals with traditional infrastructure and applications presents a number of significant challenges.

This whitepaper is intended for architects and developers in financial services sectors who are looking to expand grid computation onto AWS. It outlines the best practices for managing large grids on the AWS platform and offers a reference architecture to guide organizations in the delivery of these complex systems.

Introduction

A Definition of Grid Computing for Financial Services

High performance computing (HPC) allows end users to solve complex science, engineering, and business problems using applications that require a large amount of computational resources, as well as high throughput and predictable latency networking. Most systems providing HPC platforms are shared among many users, and comprise a significant capital investment to build, tune, and maintain.

For this paper, we focus the discussion on high performance computing applied to financial services industry sectors. This may include calculating prices and risk for derivative, commodity, or equity products; the aggregate and ticking risk calculation of trader portfolios; or the calculation of aggregate position for an entire institution. These calculations may be run continuously throughout the trading day or run at the end of the day for clearing or reporting purposes.

Unlike other types of HPC workloads, FS focused workloads tend to utilize massively parallel architectures which can tolerate some latency, with optimization focused on overall throughput. In this paper we will use the term *compute grid* to refer to HPC Clusters exhibiting these characteristics. For other use cases, such as for life sciences or engineering workloads, the approach used is different.

Compute Grid Architecture & Performance

Many commercial and open source compute grids use HTTPS for communication and can run on relatively unreliable networks with variable throughput and latency. However, for ticking risk applications, and in some proprietary compute grids, network latency and bandwidth can be important factors in overall performance. Compute grids typically have hundreds to thousands of individual processes (engines) running on tens or hundreds of machines. For reliable results, engines tend to be deployed in a fixed ratio of compute cores to memory (for example, two virtual cores and 2 GB of memory per engine). Increased throughput is measured in terms of job tasks processed per period of time. To increase the throughput of the grid, just add additional engines.

The formation of a compute cluster is controlled by a grid "director" or "controller," and clients of the compute grid submit tasks to engines via a job manager or "broker." In many grid architectures, sending data between the task client and engines is done directly, while in other architectures data is sent via the grid broker. In some architectures (Figure 1), known as two-tier grids, the director and broker responsibilities are managed by a single component, while in larger three-tier grids a director may have many brokers, each responsible for a

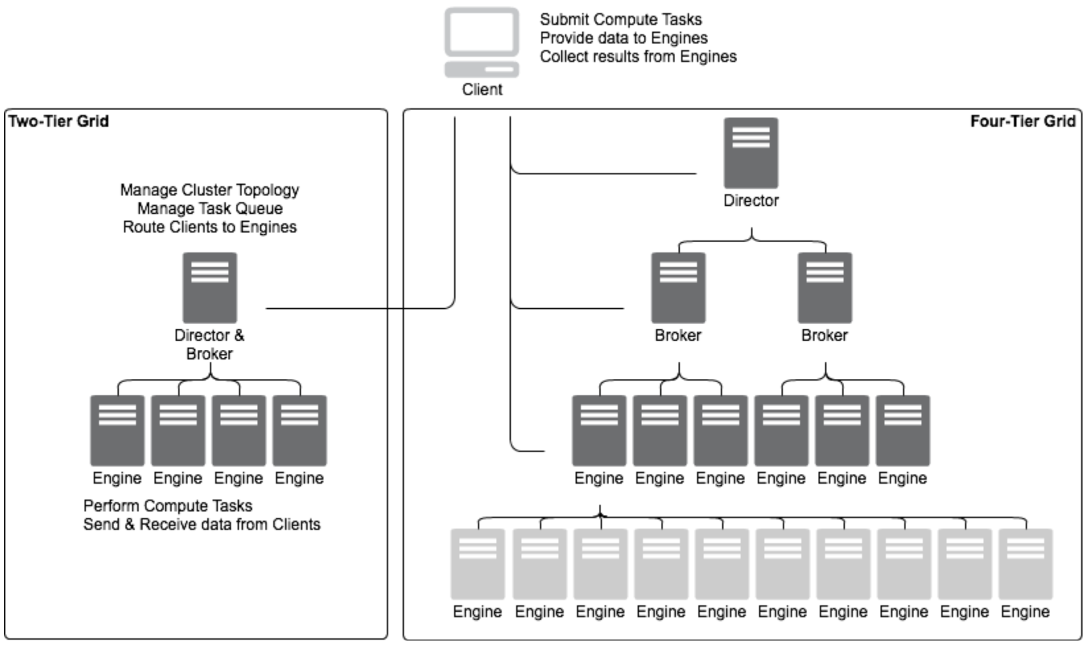

Figure 1 Two, Three and Four-Tier Grid Topologies

subset of engines. In the largest grids, a four-tier architecture may be used, where each engine has the ability to span multiple tasks onto other engines.

The timeframe for the completion of grid calculations tends to be minutes or hours rather than milliseconds. Calculations are partitioned among engines and computed in parallel, and thus lend themselves to a shared-nothing architecture. Communications with the client of the computation tend to accept relatively high latency and can be retried during failure.

Benefits of Grid Computing in the Cloud

With AWS, you can allocate compute capacity on demand without upfront planning of data center, network, and server infrastructure. You have access to a broad range of instance types to meet your demands for CPU, memory, local disk, and network connectivity. Infrastructure can be run in any of a large number of global regions, without long lead times of contract negotiation and a local presence. This approach enables faster delivery, especially in emerging markets.

With Amazon Virtual Private Cloud (VPC), you can define a virtual network topology that closely resembles a traditional network that you operate in your own data center. You have complete control over your virtual networking, including selection of your own IP address range, creation of subnets (VLANs), and configuration of route tables and network gateways. This allows you to build network topologies to meet demands of isolation and security for internal compliance and audit or external regulators.

Because of the on-demand nature of AWS, you can build grids and infrastructure as required for isolation between business lines, or for sharing of infrastructure for cost optimization. With elastic capacity and the ability to dynamically change the infrastructure, you can choose how much capacity to dedicate based upon what you are actually require at a point in time, rather than having to provision for peak utilization. Furthermore, black-swan events can be handled with ease by scaling up grids to accommodate the demands of business owners reacting to major market changes.

Operational tasks of running compute grids of hundreds of nodes are simplified on AWS because you can fully automate such provisioning and scaling tasks, and treating the infrastructure as part of your codebase. AWS resources can be controlled interactively using the AWS Management Console, programmatically using APIs or SDKs, or by using third party operational tools. You can tag instances with internal role definitions or cost centers to provide visibility and transparency at runtime and on billing statements. AWS resources are monitored by Amazon CloudWatch[1], providing visibility into the utilization rates of whole grids as well as fine granularity measures of individual instances or data stores.

You can combine elastic compute capacity with other AWS services to minimize complexity of the compute client. For example, Amazon DynamoDB, a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability, can capture results from the compute grid at any scale and throughput[2]. Sensitive software artifacts such as QA Libraries can be stored security and deployed easily with AWS CodeDeploy[3]. Compute Engines can take advantage of high performance file systems such as Intel Lustre Cloud Edition (offered on the AWS Marketplace) for temporary file storage and working space[4]. When the compute job is completed, the resources are terminated and no longer incur cost.

Many organizations are now looking for new ways to perform compute intensive tasks at a lower cost. Fast provisioning, minimal administration, and flexible instance sizing and capacity, along with innovative third party support for grid coordination and data management, make the AWS platform a compelling solution.

[1] *http://aws.amazon.com/cloudwatch*

[2] *http://aws.amazon.com/dynamodb*

[3] http://aws.amazon.com/codedeploy

[4] *https://wiki.hpdd.intel.com/display/PUB/Intel+Cloud+Edition+for+Lustre*+Software*

Grid Computing on AWS

Initial Implementation

You can build a POC or initial implementation of an HPC Grid on AWS by simply moving some of the compute grid on the Amazon Elastic Compute Cloud (Amazon EC2)[5]. An Amazon Machine Image (AMI) [6] is built with the required grid management and QA software, and grid calculations reference existing dynamic data sources via a VPN connection over Amazon VPC[7].

The VPC configuration for this architecture is very simple, comprising a single subnet for the engine instances. AWS DirectConnect[8] is used to ensure predictable latency and throughput for the large amount of customer data being transferred to the grid engines. You could also leverage Amazon Relational Database Service (Amazon RDS)[9], or Amazon DynamoDB for configuration or instrumentation data.

[5] *http://aws.amazon.com/ec2*

[6] *Please see https://aws.amazon.com/amis for a full list of available images*

[7] *http://aws.amazon.com/vpc*

[8] *See http://aws.amazon.com/directconnect for more information*

[9] *http://aws.amazon.com/rds*

The architecture shown in Figure 2 allows for a simple extension of existing grid compute jobs onto an elastic and scalable platform. By running engines on Amazon EC2 only when compute jobs are required, you can benefit from lowered cost and increased flexibility of the infrastructure used for these grids.

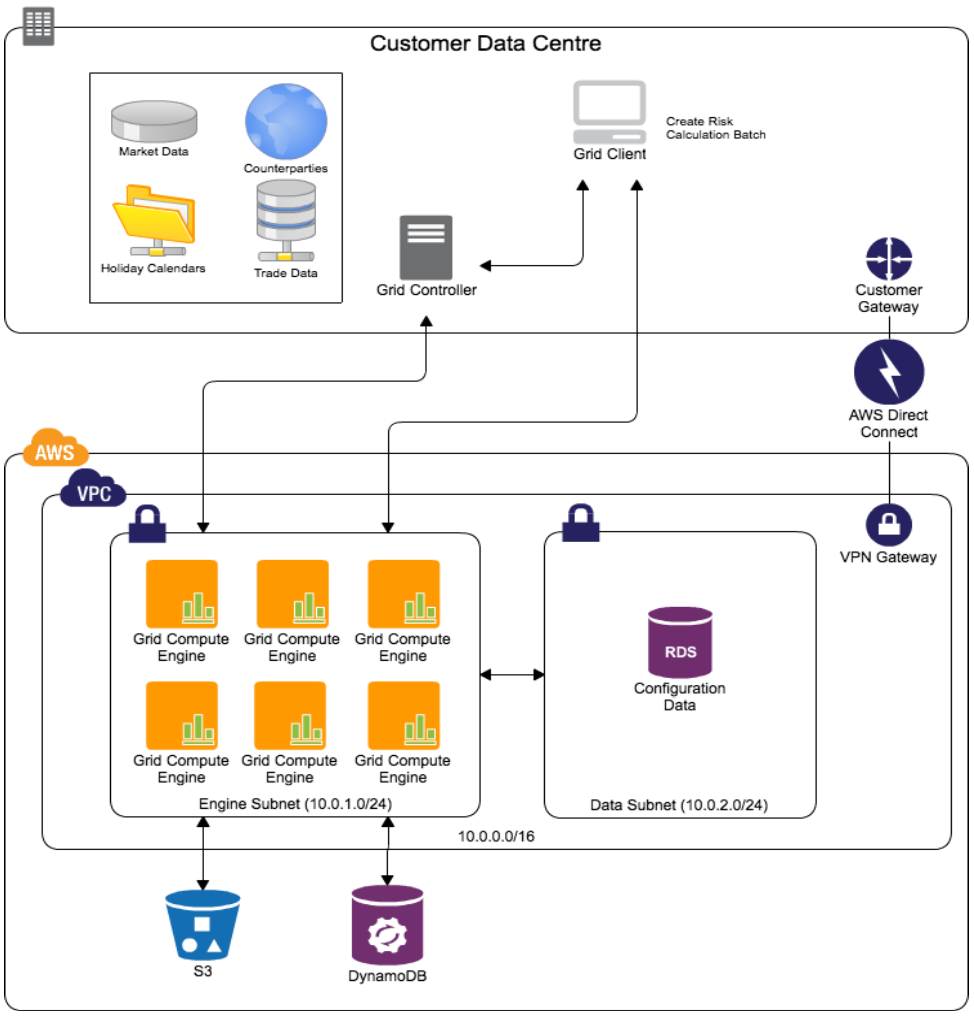

Figure 2 Initial Implementation with Grid Engines on AWS

Full Cloud Implementation

Reference Architecture

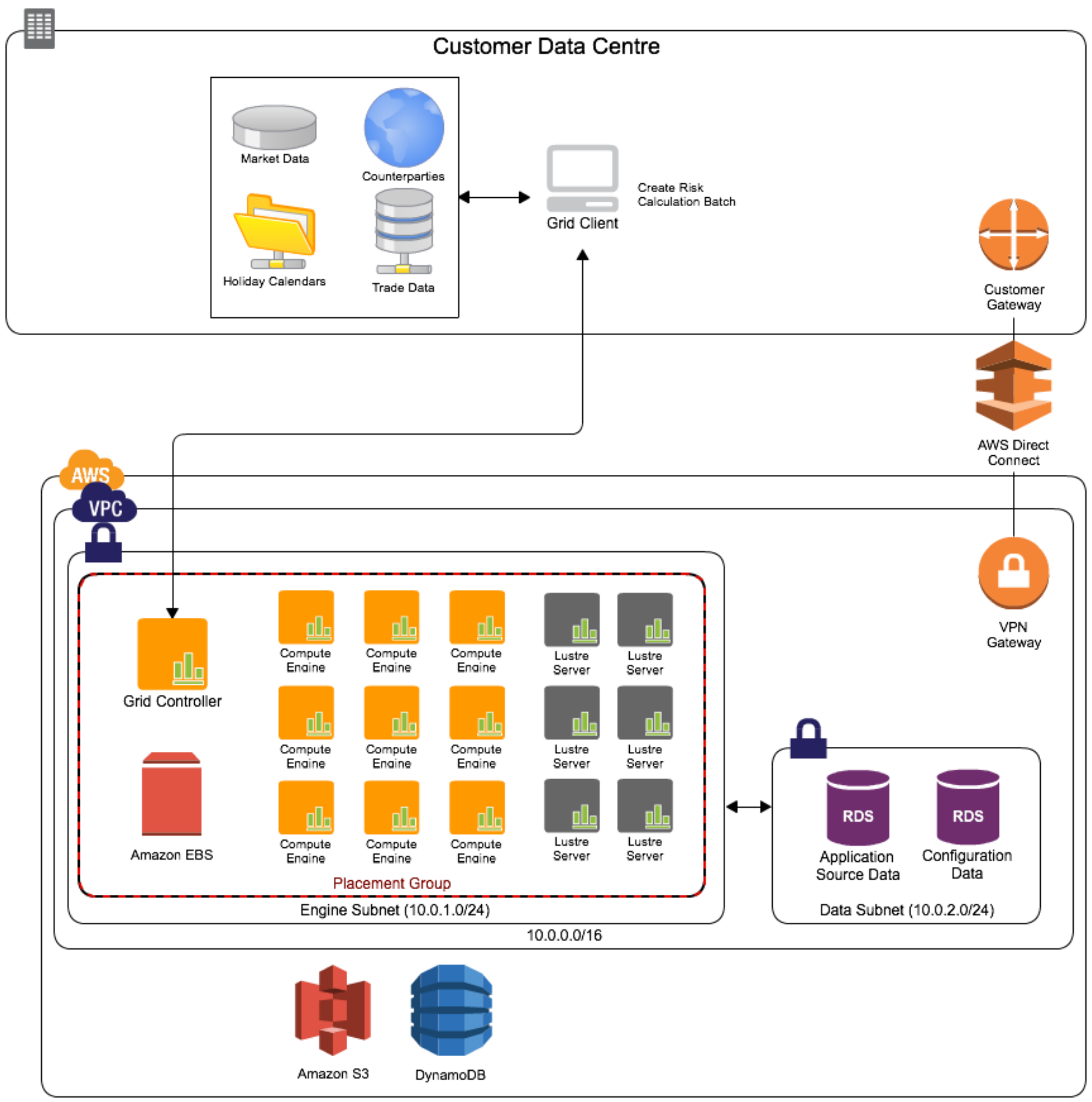

Figure 3 Grid Computing Reference Architecture

Figure 3 is an example of a reference architecture which can also be viewed in the AWS Architecture Center at http://aws.amazon.com/architecture and should be considered with the following best practices:

Security

Security of customer, trade, and market data is of paramount importance to our customers and to AWS. AWS builds services in accordance with security best practices, provides appropriate security features in those services, and documents how to use those features. In addition, AWS customers must use those features and best practices to architect and appropriately secure the application environment. AWS manages a comprehensive control environment that includes the necessary policies, processes, and control activities for the delivery of each of the web service offerings. As of the publish date of this document, AWS is compliant with various certifications and third-party attestations, including SOC1 Type 2, SOC 2 and Soc 3 compliance, PCI DSS Level 1 compliance, ISO 27001 and 9001 certification, and FISMA Moderate authorization. For a more complete and up-to-date list of AWS certifications and accreditations, please visit the AWS Security Center at http://aws.amazon.com/security.

It is important to note that AWS operates a shared responsibility model in the cloud. While you as a customer can leverage the secure underlying infrastructure and foundation services to build secure solutions, you are responsible for designing, configuring, and managing secure operating systems, platforms, and applications, while still retaining full responsibility and control over your information security management systems, security policies, standards, and procedures. You are also responsible for the compliance of your cloud solution with relevant regulations, legislation, and control frameworks.

AWS Identity and Access Management (IAM)[10] provides a robust solution for managing users, roles, and groups that have rights to access specific data sources. You can issue users and systems individual identities and credentials, or provision them with temporary access credentials relevant to their access requirements within a restricted timeframe using the Amazon Security Token Service (Amazon STS)[11]. Using standard Amazon IAM tools, you can build fine-grained access policies to meet your security requirements for cloud resources.

[10] *For more information, please see aws.amazon.com/iam*

[11] *http://docs.amazonwebservices.com/STS/latest/UsingSTS/Welcome.html*

Besides provisioning individual accounts, or dispensing temporary credentials on an as-needed basis, you can federate user identity to existing identity and access management systems such as Active Directory or LDAP. The AWS Directory Service[12] AD Connector allows you to connect to your existing Microsoft Active Directory to enable users to seamlessly access AWS resources and applications using their existing corporate credentials, and Simple AD is a stand-alone managed directory, powered by Samba 4 Active Directory Compatible Server.

Besides password authentication, AWS also supports Multi Factor Authentication (MFA) for both Web console and API access. MFA provides both secure authentication and enhanced authorization for AWS resource access. For example, you could use this feature to prevent accidental deletion of data in Amazon S3 such that only MFA-authenticated users can delete objects.

Partner solutions are also available for encrypted network overlays, privileged user access and federation, and intrusion detection systems, allowing for on-premises security control frameworks and information security management systems to be extended to the AWS cloud.

Patching

While AWS, Microsoft, Red Hat or other Operating Systems providers will release new AMI's into AWS, you must ensure that a given EC2 instance is patched to the required level based upon internal security & compliance requirements and controls. To manage this, many customers will adopt a 'rolling AMI' architecture as depicted in Figure 4.

[12] *https://aws.amazon.com/directoryservice/details/#managed-service*

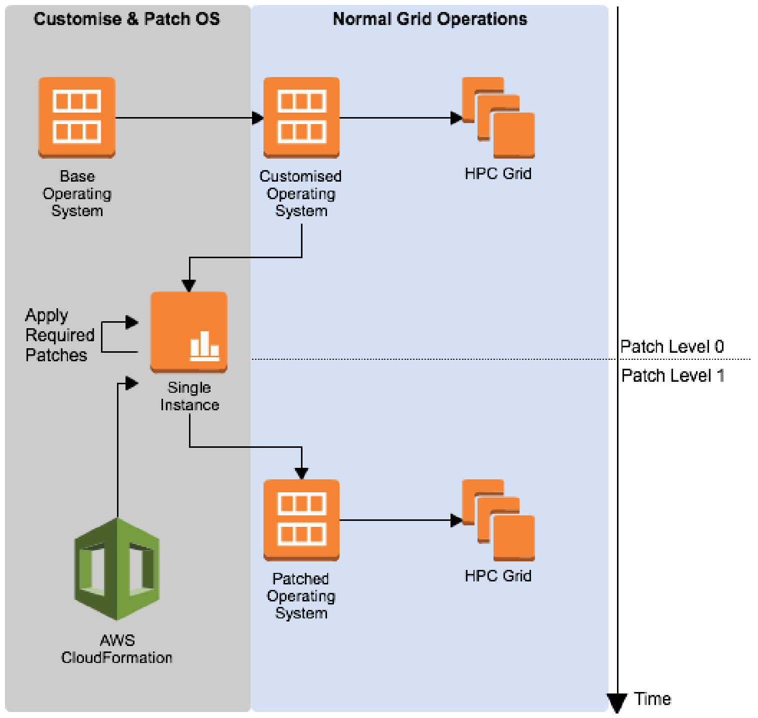

Figure 4 Example of a Rolling AMI Architecture

Starting with a base Amazon Machine Image of Amazon Linux, Red Hat or Microsoft Windows, the customer will do a one-time build of a 'current' Customized Operating System that meets the internal requirements for patching, services enabled/disabled, configuration of firewalls or other OS configuration, as well as installation of Intrusion Detection/Protection software. This AMI is then used to launch HPC Grids (or specific tiers within the Grid). At some point when patching is required, AWS CloudFormation[13] can be used to stand up a single instance from this 'current' AMI, apply the required OS patches, and then create a new 'current' patched AMI from which new Grid components can be launched. Once done, the AMI from Patch Level 0 would be deprecated, for instance through the use of Tagging Policies.

Network

For financial services grids, in addition to IAM policy control, it is a best practice to use Amazon VPC to create a logically isolated network within AWS. This allows for instances within VPC to be addressed using an IP addressing scheme of your

[13] *http://aws.amazon.com/cloudformation*

choice. You can use subnets and routing tables to enable specific routing from source data systems to the grid and client platforms on AWS, and you can use hypervisor level firewalls (VPC Security Groups[14]) to further reduce the attack surface. Your site can then be easily connected using a VPN Gateway software or hardware device[15]. You can also use static or dynamic routing protocols to provide for a single routing domain and dynamic reachability information between your internal and cloud environments.

Another networking feature that is ideal for grid computing is the use of Cluster Placement Groups[16]. This allows for fully bisected 10 GB networking between EC2 Cluster Compute instances, providing reliable network latency between grid engine nodes, and is highly recommended for the grid cluster and any high performance filesystem nodes being run.

Distribution of Static Data Sources

There are several ways to distribute static data sources, such as grid management software, holiday calendars, and gridlibs onto the instances that will be performing calculations. One option is to pre-install and bundle such components into an AMI from which an instance is launched, resulting in faster start-up times. However, as gridlibs are updated with patches and holiday dates change, these AMIs must be rebuilt. This can be time consuming from an operational perspective. Instead, consider storing static sources on an on-premises source such as a filesystem, on Amazon S3[17], or through the use of AWS CodeDeploy, which encrypts software assets automatically. Then install these components on instance start-up using AWS CloudFormation[18], shell scripts, EC2

[14] *http://docs.amazonwebservices.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html*

[15] *http://aws.amazon.com/vpc/faqs/#C8*

[16] *http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/using_cluster_computing.html*

[17] *http://aws.amazon.com/s3*

[18] *http://aws.amazon.com/cloudformation*

Config, EC2 Run Commands[19] or your preferred configuration management tools.

Access to Dynamic Data Sources

Dynamic data sources such as market, trade, and counterparty data can be safely and securely stored in the AWS cloud using services such as Amazon RDS or Amazon DynamoDB, or on a high performance filesystem such as Intel Lustre. These solutions significantly reduce operational complexity for backup and recovery, scalability, and elasticity. Datasets which require high read throughput, such as random seed data or counterparty data, are ideally placed on DynamoDB, a distributed filesystem, or datagrid to allow for configurable read IOPS by scaling out. Datasets such as client configuration, schedule, or grid metadata may also be stored simply and reliably on Amazon S3 as simple properties files, xml/json configuration, or binary settings profiles.

In situations where you must transfer large amounts of dynamic data from centralized market data systems or trade stores to compute grids during job execution, using AWS Direct Connect[20] can help to ensure predictable throughput and latency. Direct Connect establishes a dedicated network connection from internal systems to AWS. In many cases, this can reduce network costs while providing a more consistent network experience for compute grid engines. In ticking risk applications, for example, consistent performance of access to underlying data sources is vitally important.

Cluster Machine Availability & Workflow

The elasticity and on-demand nature of the AWS platform enables you to run grid infrastructure only when it is required. Unlike in a traditional data center, where grid servers are powered up and available at all times, AWS instances may be shut down, saving significant cost. This model requires the instances be started before the grid is deployed and brought online, and it requires shutting down instances when the grid is not active.

[19] *https://aws.amazon.com/ec2/run-command*

[20] *http://aws.amazon.com/directconnect*

Many customers find that they want to manage the start-up of instances themselves with custom software components. These systems may already be in place and used to install software across very large grids, and provide a good integration point for instance provisioning. To accomplish the start-up and availability of hundreds or thousands of instances[21], open source tools such as AWS cfn-cluster[22] or MIT StarCluster[23] are available, as well as commercial AWS partner solutions such as Bright Cluster Manager[24] or Cycle Computing CycleCloud[25]. A great feature of these solutions is to utilize Spot Instances[26] to drive down cost with a minimum of operational involvement. Additionally, some grid computing platforms such as Tibco Silver Fabric[27] are able to manage infrastructure built on Amazon EC2 via their internal provisioning workflow.

Regardless of the software used, the grid must be brought online prior to clients submitting tasks, so exposing metrics on number of instances available and the grid composition is of high importance when building on the cloud.

Result Aggregation & Client Scaling

Grid calculations that are run on hundreds or thousands of grid compute engines can return very large data sets back to the client, which can require a significant engineering effort ensure that the client can scale to collect all the calculation results in the required timeframe. This complexity has led many financial services customers to build data grids into which calculation results are stored

[21] *A list of solutions for cluster management can be found at http://aws.amazon.com/hpc-applications, section 'Leverage a Vibrant Ecosystem'*

[22] *https://github.com/awslabs/cfncluster*

[23] *http://aws.amazon.com/customerapps/2824*

[24] *https://aws.amazon.com/solution-providers/isv/bright-computing*

[25] *https://aws.amazon.com/solution-providers/isv/cycle-computing*

[26] *http://aws.amazon.com/ec2/spot-instances*

[27] *http://www.tibco.com/products/cloud/platform-as-a-service*

and where the final calculations are performed using parallel aggregation or map/reduce.

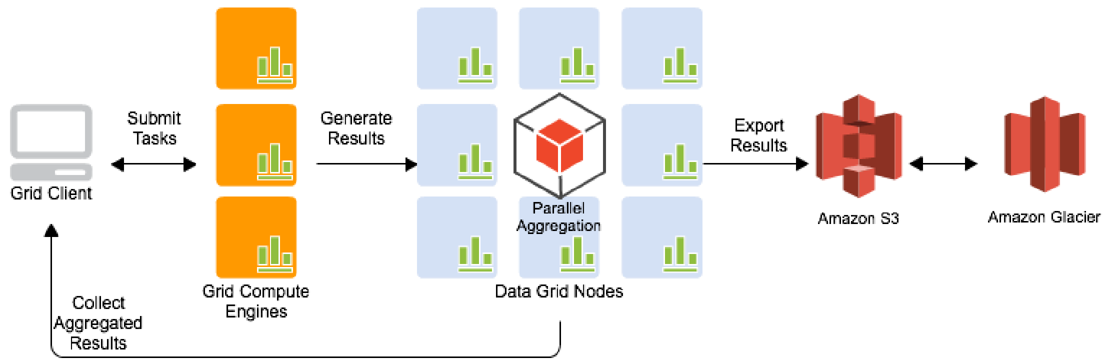

Figure 5 Task Creation and Result Aggregation Flow

Large result sets can also be exported (as shown in Figure 5) to Amazon S3 for historical access and subsequently archived to Amazon Glacier[28], an extremely low-cost storage service that provides secure and durable storage for data archiving and backup. You can share this data across business units, utilize the data for back-testing, or enable access to external parties or regulators using Identity and Access Management (IAM) policies.

High Availability

AWS Regions are divided into Availability Zones[29], which are distinct locations that are engineered to be insulated from failures in other Availability Zones and provide inexpensive, low latency network connectivity to other Availability Zones in the same Region. It is a well-established best practice to build architectures that use multiple Availability Zones for high availability and failure isolation. Grid computing architectures are no exception, but for efficiency of grid execution it is best to run all components within a single Availability Zone. By doing so, data is not being moved across multiple physical sites and thus run times are reduced.

[28] *http://aws.amazon.com/glacier*

[29] *http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html*

You should still use multiple Availability Zones during start-up of the grid cluster instances[30], as any one of the available zones in a region should be used. In the rare event of Availability Zone issues that affect a running grid, cluster management should be capable of rebuilding the grid infrastructure in an alternate Availability Zone. Once the grid is up and running again, jobs can be resubmitted for processing.

Repeatable Assembly

A best practice for any architecture built on AWS is to employ repeatable assembly of the resources in use. AWS CloudFormation enables template-driven creation of all the AWS technologies in this whitepaper, and can be used to quickly create development environments on demand, whether for new features, emergency fixes, or concurrent delivery streams. This also allows for the creation of infrastructure during performance and functional testing from automated build systems.

Compute Regions & Locations

AWS offers customers the flexibility of choosing where infrastructure is deployed across a global set of regions and Availability Zones in North and South America, Europe, and Asia. With a global business hosting dynamic data sources in multiple locations, compute can be run near to the data to minimize compute time due to network data transfer. Alternatively, you may choose to run compute jobs in a region that is optimal from a cost perspective. Lastly, you may choose to run compute jobs where they are best able to satisfy regulatory and compliance requirements, and in fact very large grids may build across multiple AWS Regions for quick and inexpensive access to capacity on demand.

Instance Type Considerations

Amazon EC2 offers a variety of instances types to choose from, including very small instances for simple testing, instances with high CPU or memory ratios, I/O optimized instances including those with solid state drives (SSDs) for random I/O intensive workloads (such as databases), and cluster compute optimized instances. Grid calculations that build large intermediate result sets may benefit from High Memory instances, while computations that are processor intensive

[30] *See best practice section on Cluster Machine Availability for more information*

may require a high CPU to Memory ratio. For computations requiring a large degree of parallelism, we recommend using the C4 Instance type, which offers a custom Intel Xeon E5-2666 v3 (Haswell) which runs at a base speed of 2.9 GHz, and can achieve clock speeds as high as 3.5 GHz with Intel® Turbo Boost (complete specifications are available here). C4 is part of our Cluster Compute Family of instance which provide high performance using fast processors and high bandwidth, low latency networking. For QA Models that can benefit from the efficiency gains of GPGPU, CG1 and G2[31] instances provide high performance of CPU combined with multiple GPU units comprising up to 1536 CUDA Cores and 4GB RAM each. And in the case of the requirement for high performance database workloads, hi1/i2 High I/O instances backed by SSDs can offer very high IOPS.

Spot Instances

Spot Instances are an option for scaling compute grids with lower cost on AWS. You simply bid on spare Amazon EC2 instances and run them whenever your bid exceeds the current Spot Price, which varies in real time based on supply and demand. You can use Spot Instances to augment the number of engines in a grid in order to speed processing for a lower cost than would be available on demand or through RIs, in the same way that you may use Scavenging on a large grid today.

[31] *http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using_cluster_computing.html*

Figure 6 shows an example of a grid augmented with Spot Instances.

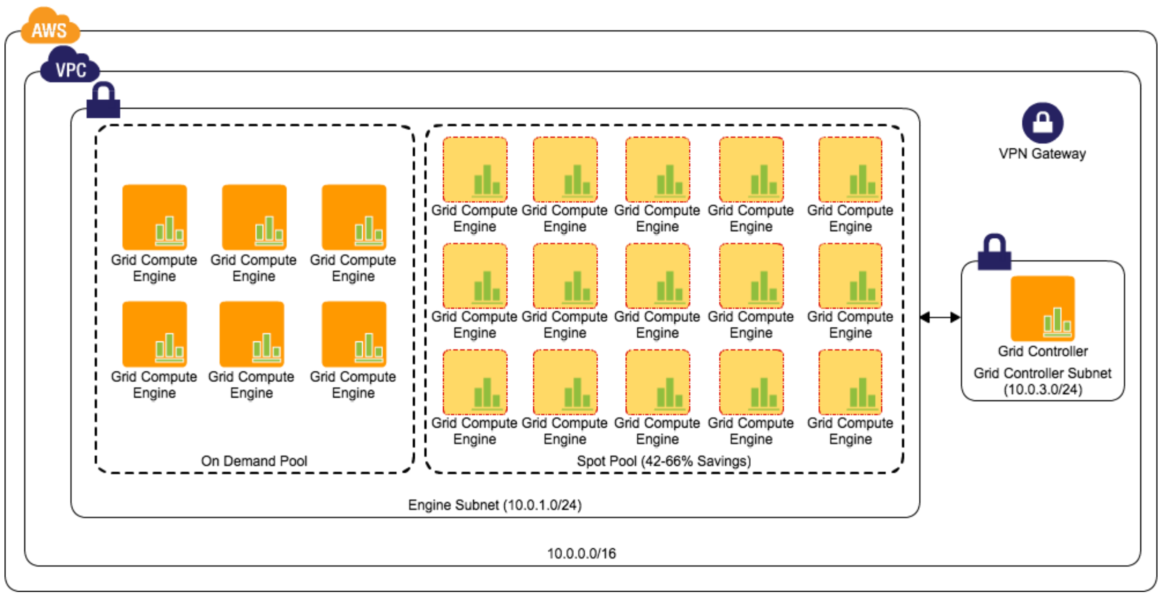
AWS
VPC
VPN Gateway
Grid Compute Engine
On Demand Pool
Spot Pool (42-66% Savings)
Grid Controller
Grid Controller Subnet (10.0.3.0/24)
Engine Subnet (10.0.1.0/24)
10.0.0.0/16

Figure 6 Grid Engines using On-Demand and Spot Instances

With Amazon EC2, it costs the same to run 100 instances for 1 hour as it does to run 1 instance for 100 hours. When using Spot Instances, the on-demand portion of the grid can be run for a shorter period of time, and in real-world scenarios, Spot Instances have allowed a time savings of 50% with a total cost reduction of 20%.

Furthermore, Spot Fleet is a feature of EC2 that allows you to describe the total capacity you require in terms of CPU and RAM, from a variety of different instance types. Spot Fleet then works to automatically deliver you that amount of capacity through Spot Instances with the lowest possible spot price. For more information on Spot Fleet, please see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html, and for Spot Instances see http://aws.amazon.com/ec2/spot-instances.

Closing Thoughts

Amazon Web Services provides a powerful mechanism for financial services organizations to expand risk and pricing grids outside of their internally managed infrastructure. Either with the simple expansion of a set of grid engines onto Amazon Elastic Compute Cloud, or using AWS to host an entire compute grid, you can process more data in a shorter period of time with no infrastructure setup costs. And by utilizing services such as Amazon Dynamo DB and Intel Lustre Cloud Edition, you can simplify the codebase used to manage these large scale environments, while using Spot Instances to drive down cost even further. The end result is the ability to process more data, more frequently, using a larger number of scenarios and tenors than before, and at a lower cost. The end result is better business decisions made quickly without the traditional large investment required to innovate.

Glossary

Gridlibs

Grid libraries are the packages of executable code or static reference data that must be copied onto each compute engine in order to perform calculations. This includes the application code that defines the risk or pricing model (the QA Library) as well as binary data for holiday calendars that aid in calculating instrument maturity dates. These elements typically change infrequently (2 or 3 times per quarter) and are small in size. They are referenced with very high frequency during calculation and so must be available directly on the engine.

QA library

Software that comprises the analytical models used to calculate risk or pricing for a security or instrument. Please see http://en.wikipedia.org/wiki/Quantitative_analysis_(finance) for more information on Quantitative Analysis.

Engine

Software component that performs calculations as part of a compute grid. Engines do not direct the flow of the overall client calculation, but instead only perform the required operation using supplied QA libraries, gridlibs, using static and dynamic data sources.

Static data

Static data sources are typically small in size and change infrequently (perhaps only several times per year) but are used many times during a risk or pricing calculation. They tend to be deployed directly to engines rather than accessed at run time.

Holiday calendar

Static data used to calculate the maturity date of a security or instrument. A holiday calendar provides information on which dates are trading days, public holidays, weekends, and so on.

Tenor

Metric indicating the time to maturity for an instrument or scenario. Risk calculations will express tenors for 1 week, 1 month, 3 months, 1 year, 10 years, and so on. Risk values are expressed at each hypothetical maturity date.

Trade data

This is the trade or portfolio data that is the subject of the calculation. This will include value, term, and information on which tenors (time to maturity) must be priced. This data is typically unique per calculation and per engine and so must be supplied each time a computation is to be run.

Market data

Market data is supplied to calculations for the purposes of understanding market movement, and the forecasting of risk based upon it. Most risk and pricing systems ”tick” only periodically, and the results of the calculations only change every 30 minutes or a multiple of hours. This data can be cached for efficiency between market ticks.

Counterparty data

Counterparty data is supplied for many types of calculations when risk is calculated at a company or group level. For example, end of day P&L is often aggregated at the various levels within an organization with which positions have been traded. This data changes infrequently, but is typically very large in size.

P&L

Profit & Loss.

Random seeds

Random input data used for Monte Carlo analysis or Back Testing. This data set must be generated up front and distributed to all engines. This data is unique for a given collection of model calculations, and is often very large in size.

Monte Carlo analysis

“Monte Carlo methods are computational algorithms that rely on random numbers to compute a result. The larger the random set of numbers the more accurate the result. This technique is used to approximate the probability of an outcome by running simulations, or trial runs using the random numbers.

Back Testing

Back testing is used to determine the effectiveness of a strategy using past historical time periods and conditions to determine expected outcome.

Grid management software

Grid management software ensures that machines and engines are available for performing calculations and controls the distribution of work. This software often takes responsibility for the distribution of gridlibs and provides a management console to change grid metadata, size, priority, and sharing settings. Grid management software is often custom built for an application by the business, but there are a many third-party products available.

Grid client

The client software that is orchestrating the calculation grid is central to the architecture. Written in a variety of different languages and running on any platform, this software must draw together all of the components of the architecture and ensure that the right calculation is performed against the data at the right time. This software is unique to each organization and even business line, and has significant performance and scaling requirements

Shared nothing (SN) architecture

An architectural pattern where each compute environment operates independently of any other compute environment. Memory and storage usage and maintenance is the responsibility of the compute environment. A contrasting pattern is a shared disk pattern where compute environments leverage common data storage.

Ticking risk

Application where market changes are consumed frequently, every 15 minutes to 1 hour, and positions are calculated for every change of the market. These systems also tend to show fine-grained impacts to prices of market movements over a day or week.

Scavenging

Ability to extend a compute Grid onto lower reliability or periodically unavailable infrastructure for the purposes of increased throughput. Typically used by extremely large grids by extending onto internal desktop machines which are idle outside of business hours.

Contributors

The following individuals and organizations contributed to this document:

- Ian Meyers, AWS

Document Revisions

Content added to this version of the whitepaper includes:

- Changed topology option to include 2, 3 and 4 tier grids
- Removed references to EMR and instead added Intel Lustre
- Added section on Patching process
- Added references to cfn-cluster for grid orchestration
- Added tibco silver reference
- c4 and g2 instance type detail
- Contrasts to life sciences/engineering HPC