Financial Services Grid Computing on Amazon Web Services

January 2013

Ian Meyers

(Please consult **http://aws.amazon.com/whitepapers** for the latest version of this paper)

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 the financial services sector 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 the financial services industry. 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.

Compute Grid Architecture & Performance

Many commercial and open source compute grids use HTTP for communication and can accept 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 to compute cores and memory (for example, two virtual cores and 2 GB of memory per engine). 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 client and engines is done directly, while in other architectures, data is sent via the grid broker. In some architectures, 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 subset of engines.

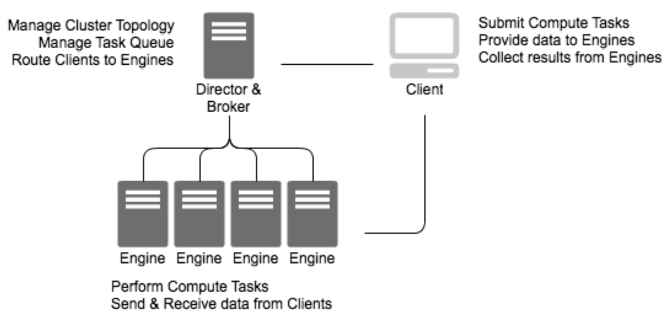

Figure 1: A Two-Tier Grid with Combined Director & Broker

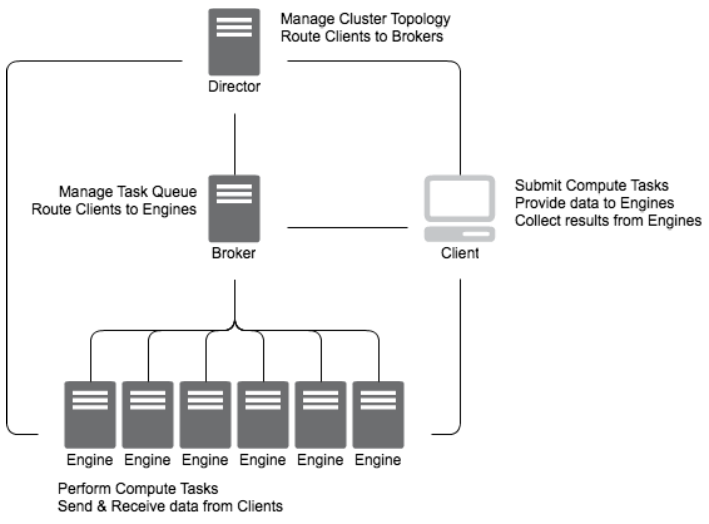

Figure 2: A Larger Three-Tier Grid

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, enabling faster delivery especially in emerging markets.

With Amazon Virtual Private Cloud (Amazon VPC) capability, you can define a virtual network topology that closely resembles a traditional network that you might operate in your own data center. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways. This allows you to build network topologies to meet demands for 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 sharing of infrastructure for cost optimization. With elastic capacity and the ability to dynamically change the infrastructure, you can now choose how much capacity to dedicate to a business line or project based upon what you are actually using at a point in time, rather than having to provision for utilization spikes.

Operational tasks of running compute grids of hundreds of nodes are simplified on AWS because you can fully automate such tasks. In general, AWS resources can be controlled interactively using the AWS Management Console, or programmatically using APIs or SDKs, or 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 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[2]. Amazon Elastic MapReduce (Amazon EMR), a hosted Hadoop framework running on the web-scale infrastructure, can perform aggregation of results stored in Amazon DynamoDB or Amazon Simple Storage Service (Amazon S3)[3]. 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.

Grid Computing on AWS

Initial Implementation

You can achieve an initial implementation of grid computing on AWS by simply moving some of the compute grid on the Amazon Elastic Compute Cloud (Amazon EC2)[4]. An Amazon Machine Image (AMI) [5] is built with the required grid management and QA software and grid calculations reference dynamic data sources via a VPN connection over Amazon VPC[6]. The VPC configuration for this architecture is very simple, comprising a single subnet for the engine instances. AWS DirectConnect[7] is used to ensure predictable latency and throughput for the large amount of customer data being transferred to the grid engines. You can also leverage Amazon Relational Database Service (Amazon RDS)[8], Amazon DynamoDB, or Amazon SimpleDB[9] for configuration or instrumentation data.

[1] *http://aws.amazon.com/cloudwatch*
[2] *http://aws.amazon.com/dynamodb*
[3] *http://aws.amazon.com/elasticmapreduce*
[4] *http://aws.amazon.com/ec2*
[5] *Please see https://aws.amazon.com/amis for a full list of available images*
[6] *http://aws.amazon.com/vpc*
[7] *See http://aws.amazon.com/directconnect for more information*
[8] *http://aws.amazon.com/rds*
[9] *http://aws.amazon.com/simpledb*

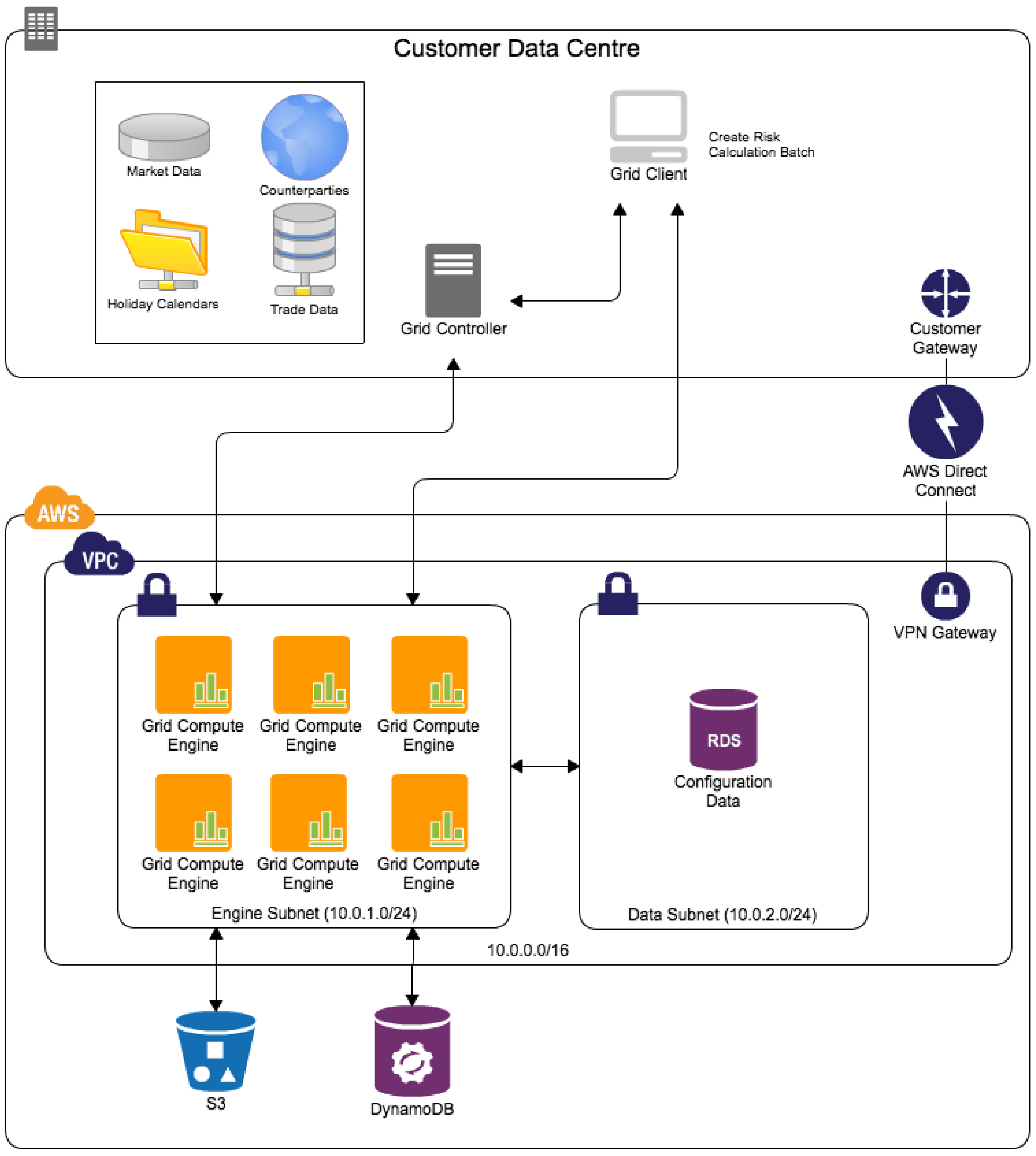

Figure 3: An initial implementation with Grid Engines on AWS

This architecture 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 see lowered cost and increased flexibility of the infrastructure used for these grids.

Full Cloud Implementation

Reference Architecture

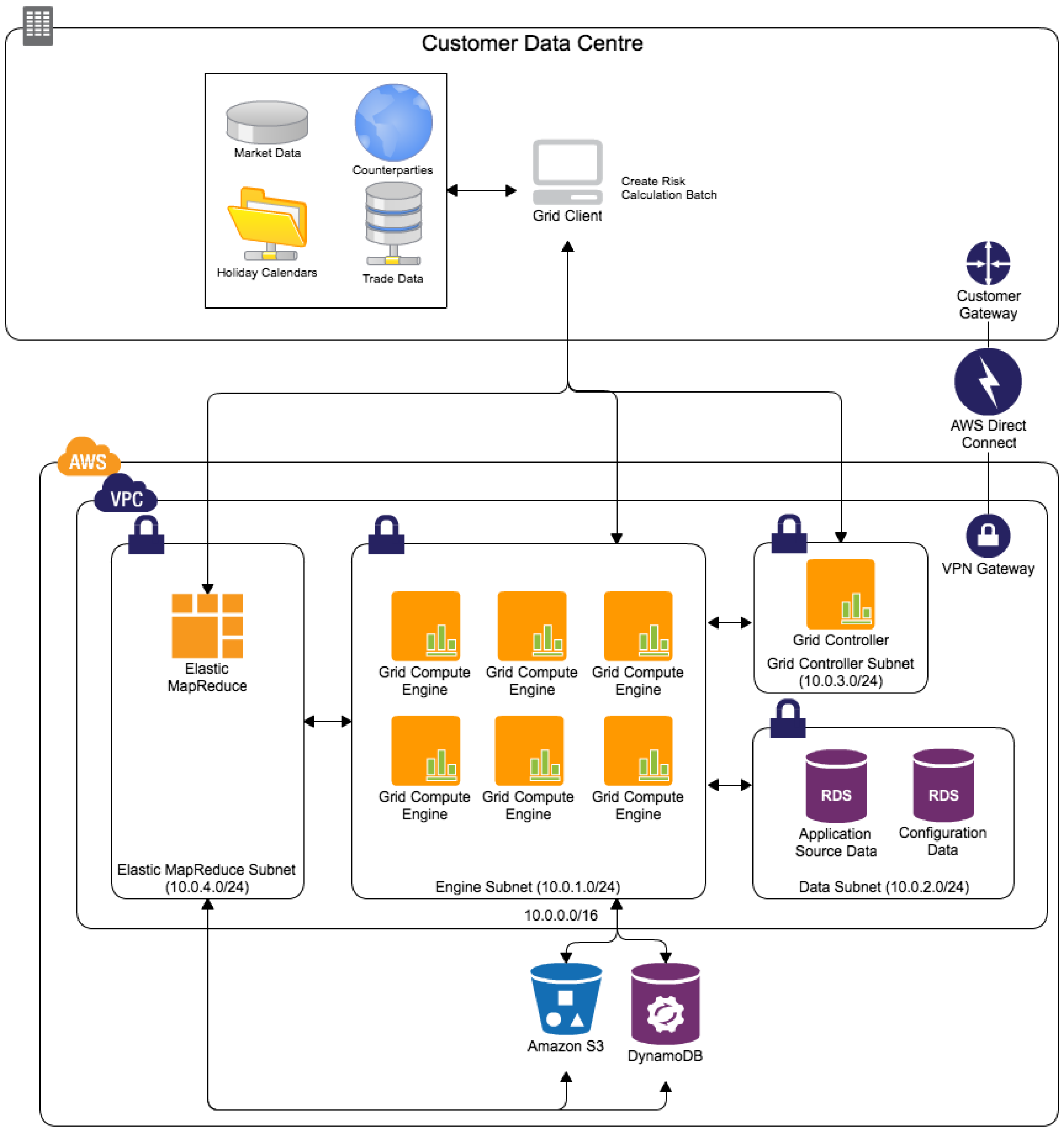

Figure 4: Grid Computing Reference Architecture

This reference architecture can be also 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 and SOC 2 compliance, PCI DSS Level 1 compliance, ISO 27001 certification, and FISMA Moderate authorization. For a more complete 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.

Besides provisioning individual accounts, or dispensing temporary credentials on an as-needed basis, you can federate user identity with existing identity and access management systems. For more information on identity broker that provides federation with Active Directory identities and user credentials, please refer to the sample implementation[12] on our website.

Besides password authentication, AWS also supports Multi Factor Authentication (MFA) for both Web console and APU 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-premise security control frameworks and information security management systems to be extended to the AWS cloud.

[10] *For more information, please see aws.amazon.com/iam*
[11] *http://docs.amazonwebservices.com/STS/latest/UsingSTS/Welcome.html*
[12] http://aws.amazon.com/code/1288653099190193

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 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[13]) to further reduce the attack surface. Your site can then be easily connected using a VPN Gateway software or hardware device[14]. You can also use static or dynamic routing protocols to provide for a single routing domain and dynamic reachability information between your on-premise and cloud environments.

Another networking feature that is ideal for grid computing is the use of Cluster Placement Groups[15]. This allows for fully bisected 10 Gb networking between EC2 Cluster Compute instances, and reduces latency of communications between grid engine nodes.

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-premise source or Amazon S3[16], and installing such components on instance start-up using AWS CloudFormation[17], shell scripts, or other 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[18] or Amazon DynamoDB. 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 to allow for configurable read IOPS. 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[19] can help to ensure predictable throughput and latency. Direct Connect establishes a dedicated network connection from on-premise 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.

[13] *http://docs.amazonwebservices.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html*
[14] *http://aws.amazon.com/vpc/faqs/#C8*
[15]*http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/using_cluster_computing.html*
[16] *http://aws.amazon.com/s3*
[17] *http://aws.amazon.com/cloudformation*
[18] *http://aws.amazon.com/rds*
[19] *http://aws.amazon.com/directconnect*

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 run-time costs. 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.

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[20], open source tools such as MIT StarCluster[21] are available, as well as commercial AWS partner solutions such as Bright Cluster Manager[22] or Cycle Computing CycleCloud[23]. This is often motivated by a desire to utilize Spot Instances[24] to drive down cost with a minimum of operational involvement. Another option is provided by grid computing platforms, many of which are now able to manage infrastructure built on Amazon EC2 via their internal configuration options.

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 run on hundreds or thousands of grid compute engines will return very large data sets back to the client. A significant engineering effort may be required to 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 and where the final calculations are performed using parallel aggregation or map/reduce. An alternative to building complex data grids is to use Amazon Dynamo DB and Amazon Elastic MapReduce as an aggregation tier. As grid engines complete calculations, the results are written to DynamoDB. The client can then run an EMR Job Flow[25] to aggregate the results and return a small aggregated result to the client.

[20] *A list of solutions for cluster management can be found at http://aws.amazon.com/hpc-applications, section 'Leverage a Vibrant Ecosystem'*
[21] *http://aws.amazon.com/customerapps/2824*
[22] *https://aws.amazon.com/solution-providers/isv/bright-computing*
[23] *https://aws.amazon.com/solution-providers/isv/cycle-computing*
[24] *http://aws.amazon.com/ec2/spot-instances*
[25] *http://aws.amazon.com/elasticmapreduce/faqs/#start-2*

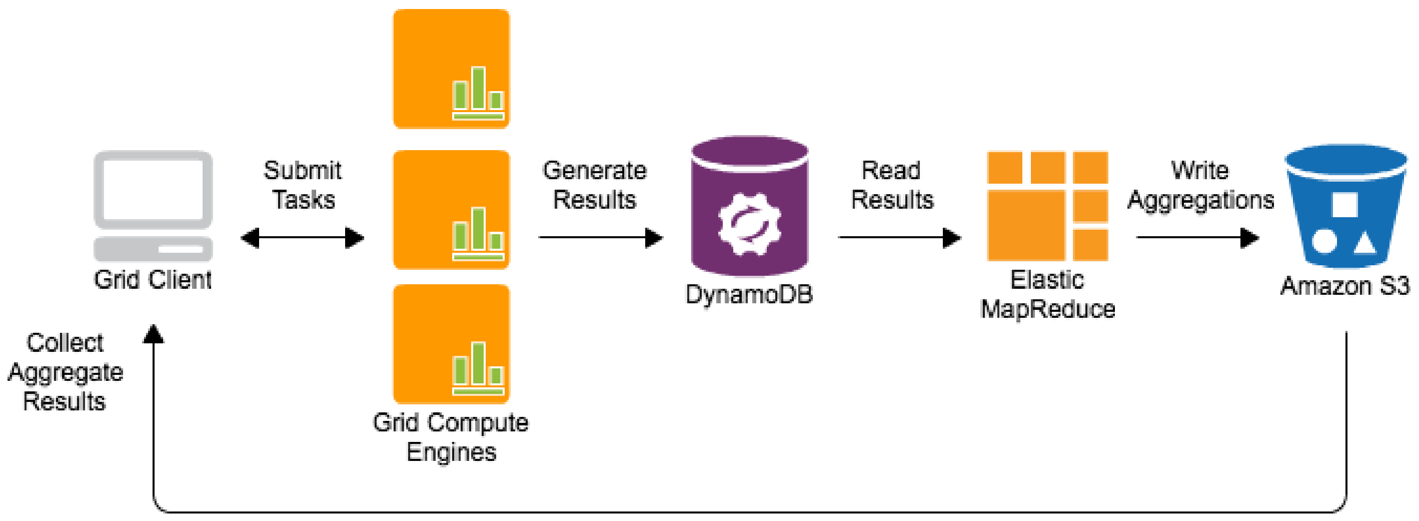

Figure 5: Task Creation and Result Aggregation Flow

Results stored in Amazon S3 can be stored for as long as required for future analysis and retrieval. Large result sets can be migrated to Amazon Glacier[26], 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 or with external parties using Identity and Access Management policies.

High Availability

AWS Regions are divided into Availability Zones[27], 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.

You should still use multiple Availability Zones during start-up of the grid cluster instances[28], 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[29] 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.

[26] *http://aws.amazon.com/glacier*
[27] *http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html*
[28] *See best practice section on Cluster Machine Availability for more information*
[29] *http://aws.amazon.com/cloudformation*

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 on-premise sites, 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 able to satisfy regulatory and compliance requirements.

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 may require a high CPU to Memory ratio. For computations requiring a large degree of parallelism, we recommend using Cluster Compute instances, 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, Cluster GPU instances provide high performance of CPU combined with multiple GPU units. And in the case of the requirement for high performance database workloads, High I/O instances backed by SSDs can offer very high IOPS.

Reserved Instances

Compute grids tend to run throughout the business day for a given region, or for batch processing at the end of a region's business day. It is relatively rare for a compute grid to be busy 24x7 even when shared between business lines. For any compute application that requires a predictable amount of resources at certain times, we recommend that you use the appropriate type of Reserved Instances (RIs) for cost efficiency. Reserved Instances enable you to maintain the benefits of elastic computing while lowering costs and reserving capacity. With Reserved Instances, you pay a low, one-time fee and in turn receive a significant discount on the hourly charge for that instance.

Because compute grids may run only 10–14 hours per day, customers will benefit from using either Light or Medium Utilization RIs. Light Utilization RIs are ideal for periodic workloads that run only a couple of hours a day, a few days per week, or sporadically, such as month-end clearing activities. Using Light Utilization RIs, you can save up to 42% for a 1-year term and 56% for a 3-year term vs. running On-Demand Instances. Medium Utilization RIs offer a cost savings for higher utilization customers or where there is variability in workload. Using Medium Utilization RIs, you can save up to 49% for a 1-year term and 66% for a 3-year term vs. running On-Demand Instances. For more information on Reserved Instances, please see http://aws.amazon.com/ec2/reserved-instances.

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. The following figure shows an example of a grid augmented with Spot Instances.

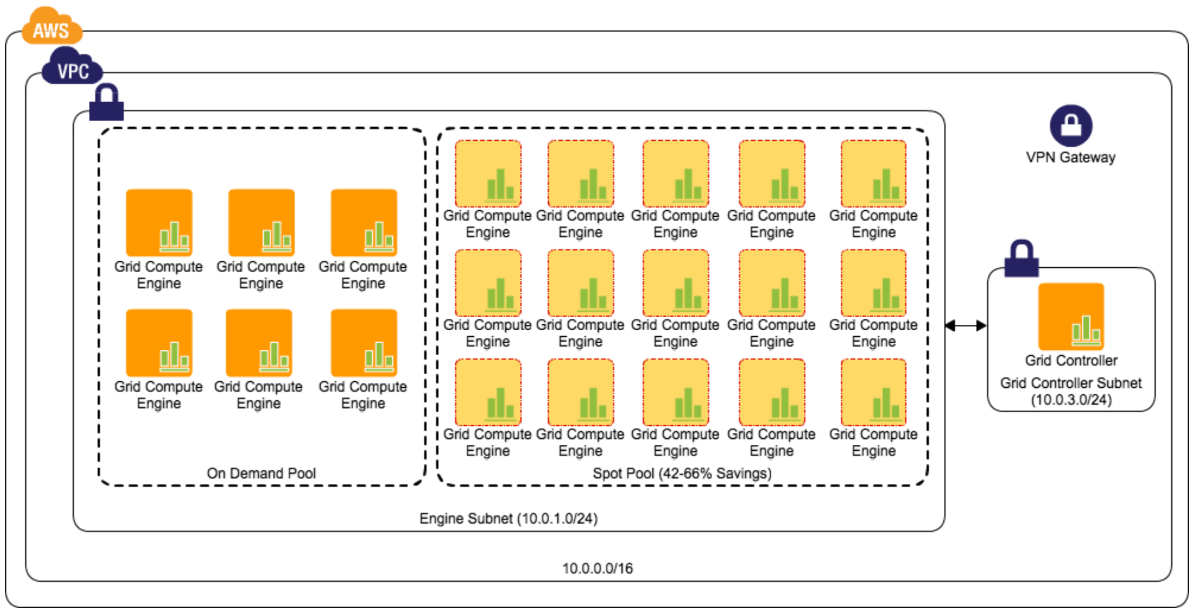
AWS
VPC
Grid Compute Engine
On Demand Pool
Spot Pool (42-66% Savings)
Engine Subnet (10.0.1.0/24)
VPN Gateway
Grid Controller
Grid Controller Subnet (10.0.3.0/24)
10.0.0.0/16

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%.

For more information on Spot Instances, please 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 Amazon Elastic MapReduce, 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 (or Monte Carlo experiments) are a class of computational algorithms that rely on repeated random sampling to compute their results. Monte Carlo methods are often used in computer simulations of physical and mathematical systems. These methods are most suited to calculation by a computer and tend to be used when it is infeasible to compute an exact result with a deterministic algorithm…" – *from* *http://en.wikipedia.org/wiki/Monte_Carlo_method*

Back Testing
"…a specific type of historical testing that determines the performance of the strategy if it had actually been employed during past periods and market conditions…" – *from* *http://en.wikipedia.org/wiki/Backtesting#Backtesting_in_finance_and_economics*

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 architecture
"A shared nothing architecture (SN) is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system. More specifically, none of the nodes share memory or disk storage…" – *from* *http://en.wikipedia.org/wiki/Shared_nothing_architecture*

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.