Financial Services Grid Computing on AWS

September 2019

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

Financial services organizations rely on *high performance computing* (HPC) infrastructure *grids* to calculate risk, value portfolios, and provide reports to their internal control functions and external regulators. The scale, cost, and complexity of this infrastructure is an increasing challenge. Amazon Web Services (AWS) provides a number of services that enable these customers to surpass their current capabilities by delivering results quickly and at a lower cost than on-premises resources.

The intended audience of this paper is grid computing managers, architects, and engineers within financial services organizations who want to improve their service. It describes the key AWS services to consider, some best practices, and includes relevant reference architecture diagrams.

Overview

High performance computing (HPC) in the financial services industry is an ongoing challenge because of the pressures from ever-increasing demand across retail, commercial, and investment groups, combined with growing cost and capital constraints. The approaches to solving these problems have evolved over generations from centralized, monolithic solutions, to business-aligned clusters of commodity hardware, to modern grid architectures with centralized schedulers that manage disparate compute capacity.

Regulators and large financial institutions are increasingly accepting hyperscale cloud providers, which has resulted in significant interest in how to best leverage new capabilities while ensuring good governance and cost controls. Cloud concepts such as *capacity on demand* and *pay as you go* pricing models offer new opportunities to teams who run HPC platforms.

Historically, the challenge has been to manage a fixed set of on-premises resources, while maximizing utilization and minimize queuing. In a model with capacity that is effectively unconstrained, the focus shifts away from managing and throttling demand towards optimizing supply. With this model, decisions become more granular and tailored to each customer, and focus on *how fast* and at *what cost*, with the ability to make adjustments as required by the business. With this basically limitless capacity, concepts such as queuing and prioritization become irrelevant as clients are able to submit calculation requests and have them serviced immediately. This also results in upstream consumers increasingly expecting and demanding near instantaneous execution of their workloads at any scale.

Initial cloud migrations of HPC platforms are often seen as extensions or evolutions of on-premises grid implementations. However, forward-looking institutions see much in common with the patterns of HPC and serverless execution models, such as AWS Lambda. Both solutions focus on executing code on demand, and customers want the lowest cost allocation of capacity with no provisioning or management of servers.

As HPC environments move to the cloud, the applications that are associated with them start to migrate too. Risk management systems which drive compute grids quickly become a bottleneck when the downstream HPC platform is unconstrained. By migrating applications with the compute grid, they also benefit from the elasticity that the cloud provides. In turn, data sources such as market and static data are sourced natively from within the cloud, from the same providers that customers work with today.

Many of the building blocks required for fully serverless solutions for risk management and reporting already exist today within AWS services. As financial institutions become increasingly familiar and comfortable with these services, it's likely that serverless patterns will become the predominant HPC architectures of the future.

Introduction

In general, HPC systems are used to solve complex mathematical problems that require thousands or even millions of CPU-hours. These systems are commonly used in academic institutions, biotech, and engineering firms. In banking organizations, HPC systems are used to quantify the risk of given trades or portfolios, which allows traders to develop effective hedging strategies, price trades, and report positions to their internal control functions and ultimately to external regulators. Insurance companies leverage HPC systems in a similar way for actuarial modeling and in support of their own regulatory requirements.

Unpredictable, seasonal, and time-bound usage patterns contribute to a mixture of demands on HPC platforms. This includes short, latency-sensitive, intraday pricing tasks, and near real-time risk measures calculated in response to changing market conditions, or large overnight batch workloads and back-testing to measure the efficacy of new models to historic events. Combined, these workloads can generate hundreds of millions of tasks per day with a significant proportion running for less than a second.

Because of the regulatory landscape, demand for these calculations continues to outpace the progress of Moore's law. Regulations such as the *Fundamental Review of the Trading Book* (FRTB) and *IFRS 17* require even more analysis, with some customers estimating between 40% and 1000% increases in demand as a result. As a result, financial services organizations continue to grow their grid computing platforms and increasingly wrestle with the costs associated with purchasing and managing this infrastructure.

Risk and pricing calculations in financial services are most commonly embarrassingly parallel, do not require communication between nodes to complete calculations, and broadly benefit from horizontal scalability. Because of this, they are well suited to a shared-nothing architectural approach, in which each compute node is independent from the other. For example, a financial model based on the Monte Carlo method can create millions of scenarios to be divided across a large number of compute nodes for calculation in parallel. Each scenario shows a different market condition based on a number of variables. In general, doubling the number of compute nodes allows these tasks to be distributed more widely, which reduces by half the overall duration of the job.

Access to increased compute capacity through AWS allows for additional scenarios and greater precision in the results in a given timeframe. Alternatively, customers can use the additional capacity to complete the same calculations in less time.

Financial services firms typically use a grid scheduler to orchestrate their HPC workloads, which have these features in common:

- A central scheduler to coordinate multiple clients and a large number (typically hundreds or thousands) of compute nodes. The scheduler manages the loss of any given component, and reschedules the work accordingly.
- Deployment tools to ensure that software binaries and data are reliably distributed to compute nodes that are allocated a specific task.
- An engine to allow rules to be defined to ensure that certain workloads are prioritized over others in the event that the total capacity of the grid is exhausted.
- Brokers are typically employed to manage the direct allocation of tasks that are submitted by a client to the compute grid. In some cases, an allocated compute node makes a direct connection back to a client to collect tasks to reduce latency. Brokers are usually horizontally scalable and are well suited to the elasticity of cloud.

In some cases, the client is another grid node that generates further tasks. Such multi-tier, recursive architectures are not uncommon, but present further challenges for software engineers and HPC administrators who want to maximize utilization while managing risks, such as deadlock when parent tasks are unable to yield to child tasks.

The key benefit of running HPC workloads on AWS is the ability to allocate large amounts of compute capacity on-demand without the need to commit to the upfront and on-going costs of a large hardware investment. Capacity can be scaled minute-by-minute according to the customer needs at the time. This avoids pre-provisioning of capacity according to some estimate of future peak demand. Also, because AWS infrastructure is charged by consumption of CPU-hours, it's possible to complete the same workload in less time, for the same price, by simply scaling the capacity.

Figure 1 shows two approaches to provisioning capacity. In the first, two CPUs are provisioned for 10 hours. In the second, 10 CPUs are provisioned for two hours. In a CPU-hour billing model the overall cost is the same but the latter produces results in one fifth of the time.

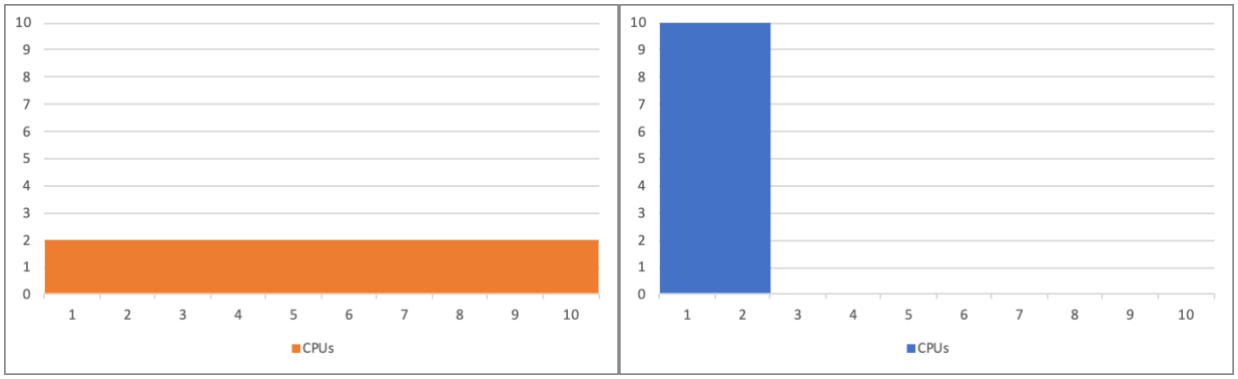

Figure 1 – Two approaches to provisioning 20 CPU-hours of capacity

Developers of the analytics calculations used in HPC applications can use the latest CPUs, GPUs, and FPGAs available through the many Amazon EC2 instance types. This drives efficiency-per-core, and differs from on-premises grids that tend to be a mixture of infrastructure, which reflects historic procurement rather than current needs. In addition, diverse pricing models offer flexibility to these customers. For example, Amazon EC2 Spot Instances can reduce compute costs by up to 90%. These instances are occasionally interrupted by AWS, but HPC schedulers with a history of managing scavenged CPU resources can react to these events and rescheduling tasks accordingly.

This document includes several recommended approaches to building HPC systems in the cloud, and highlights AWS services that are used by financial services organizations to help to address compute, networking, storage, and security requirements.

Grid Computing on AWS

A key driver for the migration of HPC workloads from on-premises environments to the cloud is flexibility. AWS offers HPC teams the opportunity to build reliable and cost-efficient solutions for their customers, while retaining the ability to experiment and innovate as new solutions and approaches become available.

HPC teams that want to migrate an existing HPC solution to the cloud, or to build a new solution, should review the AWS Well-Architected Framework. This framework applies to any cloud deployment and seeks to ensure that systems are architected according to best practices. The HPC specific lens document also identifies key elements to help ensure the successful deployment and operation of HPC systems in the cloud.

The following sections include information about AWS services that are most relevant to HPC systems, particularly those that support financial services customers.

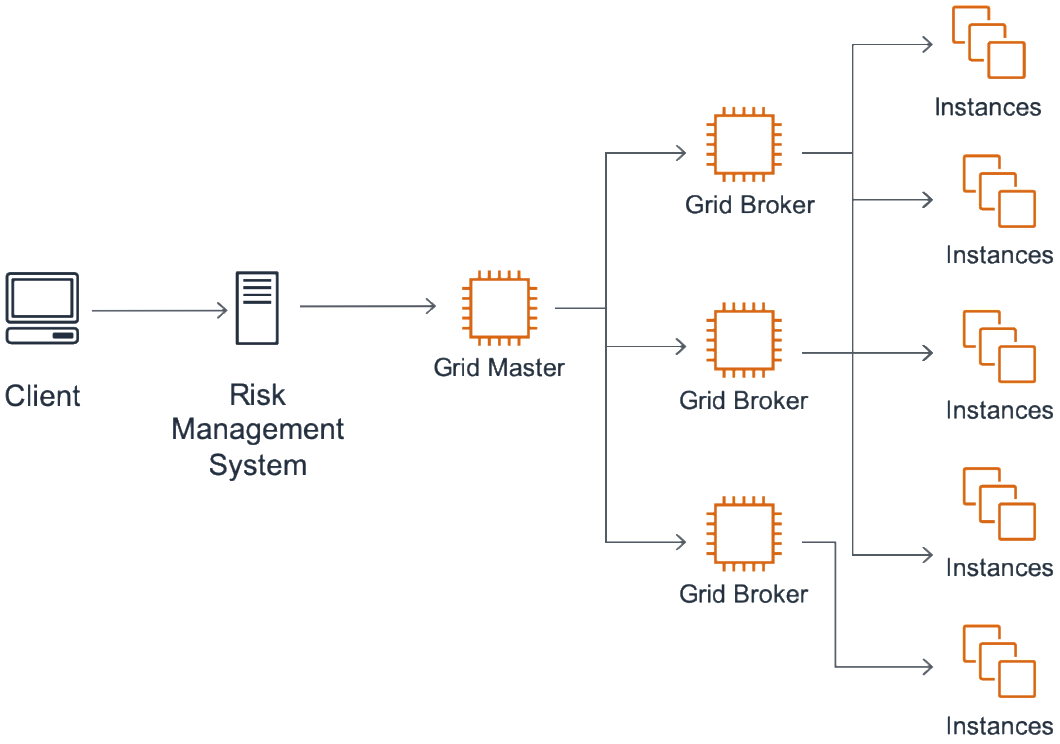

Figure 2 – A typical HPC architecture with the key components, including the risk management system (RMS), grid master, grid brokers, and two compute instance pools

Compute and Networking

AWS offers a wide range of Amazon Elastic Compute Cloud (Amazon EC2) instance types, which enables customers to select the configuration that is best suited to their needs at any given time. This is a departure from the typical *Bill of Materials* approach, which limits the configurations available on-premises in favor of deployment simplicity. It also offers *ever greening*, which enables you to take advantage of the latest CPU technologies as they are released without consideration for any prior investment. HPC customers in financial services should consider the following instances types.

- Amazon EC2 Compute Optimized instances – *C* class instances are optimized for compute-intensive workloads and deliver cost-effective high performance at a low price per compute ratio.
- Amazon EC2 General Purpose instances:
    - *M* class – Commonly used in HPC applications because they offer a good balance of compute, memory, and networking resources.
    - *Z* class– Offer the highest CPU frequencies with a high memory footprint.
    - *T* series — Provide a baseline level of CPU performance with the ability to burst to a higher level when required. The use of these instances for HPC workloads can be attractive for some workloads, however, their variable performance profile can result in inconsistent behavior which might be undesirable.
- Instances with the suffix *a,* are AMD processors, for example, *R5a*.
- Amazon EC2 Accelerated Computing instances use hardware accelerators, or co-processors, to perform functions, such as floating point number calculations, graphics processing, or data pattern matching, more efficiently than is possible in software running on CPUs.
    - *P* class instances are intended for general-purpose GPU compute applications.
    - *F* class instances offer customizable hardware acceleration with field programmable gate arrays (FPGAs).

Many of the latest AWS instances are based on the AWS Nitro System. The Nitro system is collection of AWS-built hardware and software components that enable high performance, high availability, high security, and bare metal capabilities to eliminate virtualization overhead. By selecting Nitro based instances, HPC applications can

expect performance levels that are indistinguishable to a bare-metal system while retaining all of the benefits of an ephemeral virtual host.

Table 1 – Amazon EC2 instance types that are typically used for HPC workloads

Instance Type	Class	Description
General Purpose	T	Burstable, general purpose, low cost
	M	General purpose instances
Compute Optimized	C	For compute intensive workloads
Memory Optimized	R	For memory intensive workloads
	X	For memory intensive workloads
	Z	High compute capacity and high memory
Accelerated Computing	P / F	General purpose GPU (P) or FPGA (F) capabilities

This diverse selection of instance types helps support a wide variety of workloads with optimal hardware and promotes experimentation. HPC teams can benchmark diverse sets of instances to optimize their scheduling strategies. Quantitative developers can try new approaches with GPUs, FPGAs, or the latest CPUs, without upfront costs or protracted procurement processes. You can immediately deploy at scale your optimal approach, without the traditional hardware lifecycle considerations.

When you run experiments, or if a subset of production workloads require a specific instance type, grid schedulers typically enable tasks to be directed to the appropriate hardware through compute resource groups.

Amazon EC2 instances support multithreading, which enables multiple threads to run concurrently on a single CPU core. Each thread is represented as a virtual CPU (vCPU) on the instance. An instance has a default number of CPU cores, which varies according to instance type. To make sure that each vCPU is used effectively, it's important to understand the behavior of the calculations running in the HPC environment. If all processes are single-threaded, a good initial strategy is to have the scheduler assign one process per vCPU on each instance. However, if the calculations require multithreading, tuning might be required to maximize the use of vCPUs without introducing excessive CPU context switching.

Amazon EC2 instances have hyperthreading (HT) enabled by default. You can disable HT either at boot or at runtime if the analytics perform better without it, which you can

establish through benchmarking. The Disabling Intel Hyper-Threading Technology on Amazon Linux blog post has an explanation of the methods you can use to configure HT on an Amazon Linux instance.

Customers might typically tune their infrastructure to increase processor performance consistency or to reduce latency. Some Amazon EC2 instances allow control of processor C-states (idle state power saving) and P-states (optimization of voltage and CPU frequency during execution). The default settings for C-state and P-state are tuned for maximum performance for most workloads. If an application might benefit from reduced latency in exchange for lower frequencies, or from more consistent performance without the benefit of Turbo Boost, then changes to the C-state and P-state configurations might be worth considering. For information about the instance types that support the adjustment and how to make these changes to an Amazon Linux 2-based instance, see Processor State Control for Your EC2 Instance in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

Another potential optimization is over-subscription. This approach is useful when you know processes spend time on non-CPU intensive activities, such as waiting on data transfers or loading binaries into memory. For example, if this overhead is estimated at 10%, you might be able to schedule one additional task on the host for every 10 vCPUs to achieve higher CPU utilization and throughput.

Compiler selection is another consideration. The use of a complier that is optimized for the target CPU architecture can yield performance improvements. For example, quantitative analysts might see value in developing analytics using the Intel C++ Compiler and running on instances that support AVX-512 capable CPUs.

In addition to the instance types and classes shown in Table 1, there are also options for procuring instances.

- Amazon EC2 On-Demand Instances offer capacity as required, for as long as they are needed. Customers are only charged for the time that the instance is active. These are ideal for components that benefit from elasticity and predictable availability, such as brokers, compute instances hosting long-running tasks, or tasks that generate further generations of tasks.

- Amazon EC2 Spot Instances are particularly appropriate for HPC compute instances because they benefit from substantial savings over the equivalent on-demand cost. Spot Instances can occasionally be terminated by AWS when capacity is constrained, but grid schedulers can accommodate the occasional interruptions. Additionally, it might be possible for some HPC applications to react to an impending termination notification and save a snapshot of their current state to a datastore (checkpointing). For applications with relatively long tasks, this approach ensures that progress on the calculation is not lost, and a future Amazon EC2 instance can continue from that point.
- Amazon Reserved Instances provide a significant discount of up to 75% based on a one-year or three-year commitment. Convertible Reserved Instances offer additional flexibility on the instance family, operating system, and tenancy of the reservation. Relatively static hosts, such as HPC Master nodes or data caching hosts, might benefit from Reserved Instances.

Compute Instance Provisioning and Management Strategies

In response to a Spot Instance interruption, Amazon EC2 supports hibernation. This feature operates like closing and opening the lid on a laptop computer, and saves the memory state to an Amazon Elastic Block Store (EBS) disk. However, this approach to managing interruptions should be used with caution because the grid scheduler might not be able to track such quiesced workloads, which could result in timeouts and rescheduling tasks if the hibernated image is not reactivated quickly.

To reduce the frequency of interruptions, Amazon EC2 Spot Blocks try to make sure that instances are available for a predefined period of up to 6 hours and are only interrupted in rare situations. Instances are generally available for the duration specified, at a cost premium over the usual Spot Instance pricing. Instances are automatically terminated when the *BlockDuration* time expires.

Amazon EC2 Spot Fleets enable customers to launch a fleet of Spot Instances that span various EC2 instance types and Availability Zones. By defining the target capacity using an appropriate metric (for example, a *Slot* for an HPC application) the fleet sources capacity from EC2 Spot Instances at the best possible price. HPC teams can define Spot Fleet strategies that use diverse instance types to make sure customers have the best experience at the lowest cost.

Amazon EC2 Fleet takes the Spot Fleet model and enables customers to quickly create fleets that are further diversified by using EC2 On-Demand Instances, Reserved

Instance, and Spot Instances. With this approach, you can optimize your HPC capacity management plan according to the changing demands of your workloads.

Amazon EC2 Launch Templates contain the configuration information used to launch an instance. The template can define the AMI ID, instance type, and network settings for the compute instances. You can use templates with EC2 Fleet, Spot Fleet, or EC2 Auto Scaling and make it easier to implement and track configuration standards.

One option to begin an HPC deployment is to use only On-Demand Instances. After you understand the performance of your workloads, you can determine the correct process to develop and optimize a strategy to provision instances using Amazon EC2 Fleet. For example, you can deploy a number of Reserved Instances to host core grid services, such as schedulers that are required to be available at all times. You can provision On-Demand Instances during the intraday period to ensure predictable performance for synchronous pricing calculations. For the overnight batch, you can use large fleets of Spot Instances to provide massive volumes of capacity at a minimum cost, and supplement it as necessary by On-Demand Instances to ensure predictable performance for the most time sensitive workloads.

Figure 3 shows two approaches to provisioning. In each case, 10 vCPUs of Reserved Instance capacity remain online for the stateful scheduling components. In the first case, 20 further vCPUs are provisioned using On-Demand Instances for 10 hours to accommodate a batch that runs for 200 vCPU hours with a 10-hour SLA. In the second approach, the 20 vCPUs are also provisioned at the outset to provide confidence in the batch delivery, but low-cost Spot Instances are also added. Because of the volume of Spot Instances, the batch completes much more quickly (in about 3 hours) and at a significantly reduced cost. However, if the Spot Instances were not available for any reason, the batch would still complete on time with the On-Demand Instances provisioned.

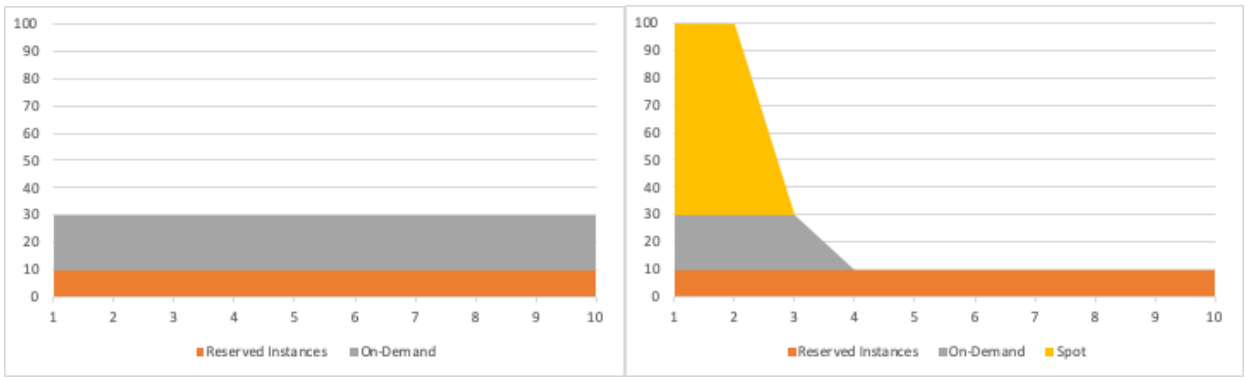

Figure 3 – AWS instance provisioning strategies

One of the key benefits of deploying applications in the AWS Cloud is *elasticity*. AWS Auto Scaling enables HPC managers to configure Amazon EC2 instance provisioning and decommissioning events based on the real-time demands of their platform. Though grids were previously provisioned based on predictions of peak demands (with periods of both constraint and idle capacity), Auto Scaling automatically adds and removes instances based on demand.

When you remove hosts from a running cluster, make sure to allow for a *drain down* period. During this period, the targeted host stops taking on new work, but is allowed to complete work in progress. When you select nodes for removal, make sure to avoid any long-running tasks, so that the shutdown is not delayed and you don't lose effort on long-running tasks. If the scheduler allows a query of total execution time of tasks in progress, grouped by instance, you can use this group to identity which are the optimal candidates for removal, specifically the instances with the lowest aggregate total of execution time by tasks in progress.

Storage and Data Sharing

In HPC systems, there are two primary data distribution challenges. The first is the distribution of binaries. In financial services, large and complex analytical packages are common. These packages are often 1GB or more in size, and often multiple versions are in use at the same time on the same HPC platform, to support different businesses or back-testing of new models.

In a constrained, on-premises environment, you can mitigate this challenge by relatively infrequent updates to the package and a fixed set of instances. However, in a cloud-based environment, instances are short-lived and the number of instances can be much larger. As a result, multiple packages are distributed to thousands of instances on an hourly basis as new instances are provisioned and new packages are deployed.

There are a number of possible approaches to this problem. One is to maintain a build pipeline that incorporates binary packages into the Amazon Machine Images (AMIs). This means that once the machine has started, it can process a workload immediately because the packages are already in place. A limitation of this approach is that it doesn't accommodate the deployment of new packages to running instances, and it requires them to be terminated and replaced to get new versions.

Another approach is to update running instances. There are two different methods for this type of update, which are sometimes combined.

The first method is *pull* (or *lazy*) deployment. In this mode, when a task reaches an instance and it depends on a package that is not in place, the engine *pulls* it from a central store before it executes the task. This approach minimizes the distribution of packages and saves on local storage because only the minimum set of packages is deployed. However, these benefits are at the expense of delaying tasks in an unpredictable way, such as the introduction of a new instance in the middle of a latency sensitive pricing job. This approach may not be acceptable if large volumes of tasks have to wait for the grid nodes to pull packages from a central store which could struggle to service very large numbers of requests for data.

The second method is *push* deployment. In this mode you can instruct instance engines to proactively get a specific package before they receive a task that depends on it. This approach allows for rolling upgrades and ensures tasks are not delayed by a package update. One challenge with this method is the possibility that new instances (which can be added at any time) might miss a *push* message, which means you must keep a list of all currently *live* packages.

In practice, a combination of these approaches is common. *Standard* analytics packages are *pushed* because they're likely to be needed by the majority of tasks. Experimental packages are then *pulled*, perhaps to a smaller set of instances.

It might also be necessary to purge deprecated packages, especially if you deploy experimental packages. In this case, you can use your list of *live* packages to enable your compute instances to purge any packages that are not in the list and thus are not current.

Figure 4 shows a cloud-native implementation of these approaches. It uses a centralized package store in Amazon Simple Storage Service (Amazon S3) with agents that respond to messages delivered through an Amazon Simple Notification Service (Amazon SNS) topic.

After the package is in place on Amazon S3, notifications of new releases can be generated either by an operator or as a final step in an automated build pipeline. Compute instances subscribed to an SNS topic (or to multiple topics for different applications) use these messages as a trigger to retrieve packages from Amazon S3. You can also use the same mechanism to distribute *delete* messages to remove packages, if required.

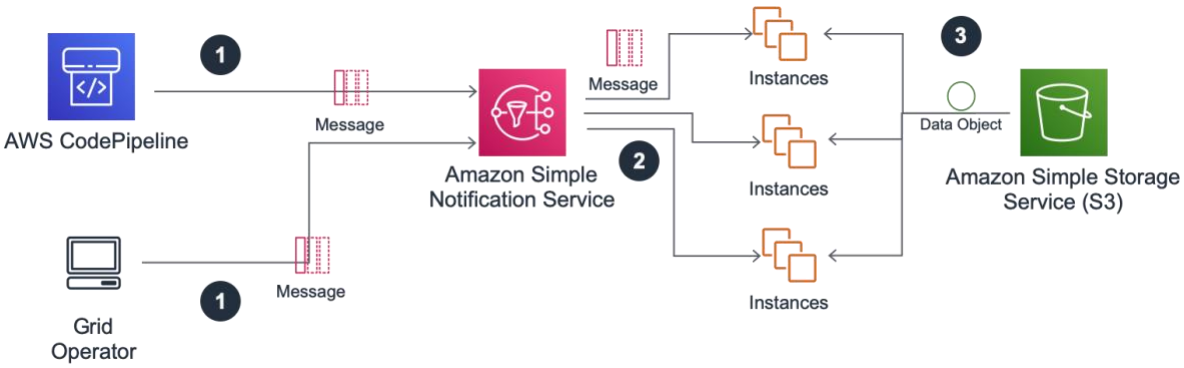

Figure 4 – Data distribution architecture using Amazon SNS messages and S3 Object Storage

The second data distribution challenge in HPC is managing data related to the tasks being processed. Typically, this is bi-directional, with data flowing to the engines that support the processing and resulting data passed back to the clients. There are three common approaches for this process.

In the first approach, communications are *inbound* (see Figure 5) with all data passing through the grid scheduler along with task data. This is less common because it can cause a performance bottleneck as the cluster grows.

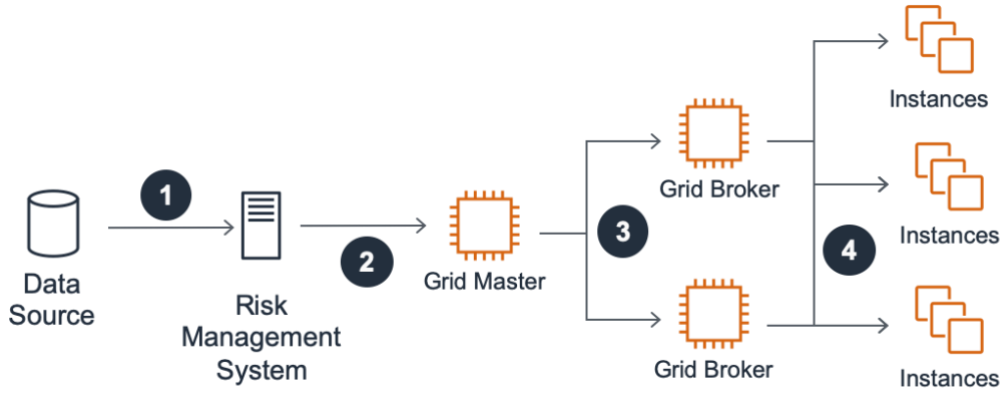

Figure 5 – An inbound *data distribution approach*

In another approach, tasks pass through the scheduler, but the data is handled *out-of-bounds* through a shared, scalable data store or an in-memory data grid (see Figure 6). The task data contains a reference to the data's location and the compute instances can retrieve it as required.

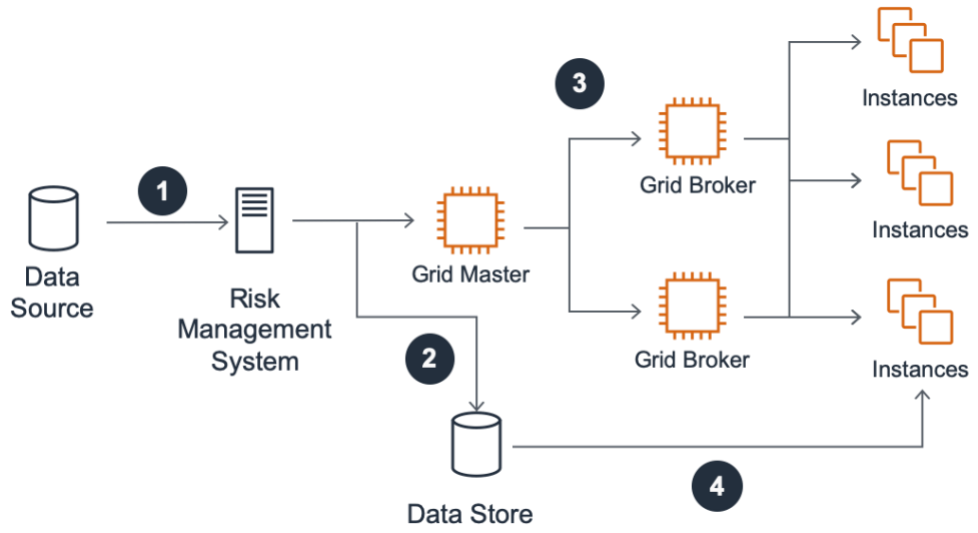

Figure 6 – An out-of-bounds *data distribution approach*

Finally, some schedulers support a direct data transfer (DDT) approach. In this model the scheduler grid broker allocates compute instances which then communicate directly with the client. This architecture can work well, especially with very short running tasks with little data. However, in a hybrid model, with thousands of engines running on AWS that need to access a single, on-premises client, this can present challenges to on-premises firewall rules, or to the availability of ephemeral ports on the client host.

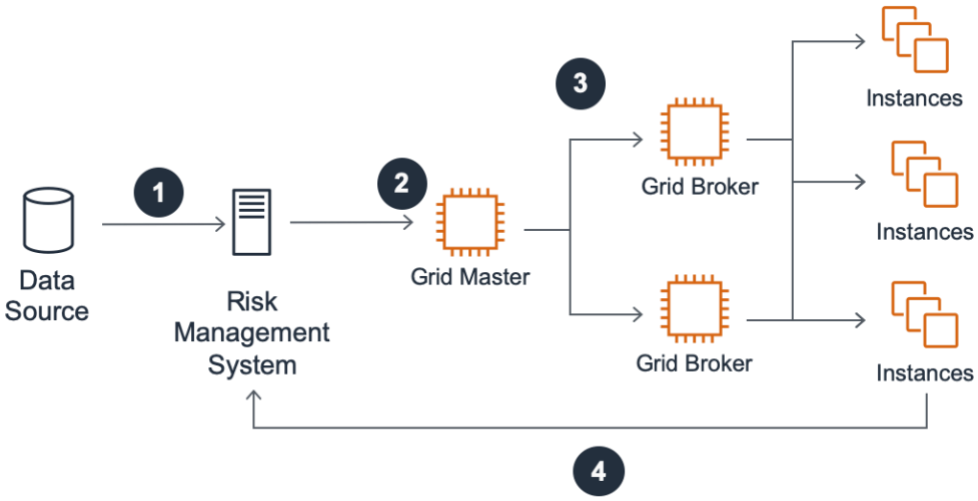

Figure 7 – DDT (direct data transfer) data distribution approach

All of these approaches can be enhanced with caches located as close as possible to, or hosted on, the compute instances. Such caches help to minimize the distribution of data, especially if a significantly similar set is required for many calculations. Some

schedulers support a form of data-aware scheduling that tries to ensure that tasks that require a specific dataset are scheduled to instances that already have that dataset. This cannot be guaranteed, but often provides a significant performance improvement at the cost of local memory or storage on each compute instance.

Though the combination of grid schedulers and distributed cache technologies used on premises can provide solutions to these challenges, their capabilities vary and they are not typically engineered for a cloud deployment with highly elastic, ephemeral instances. You can consider the following AWS services as potential solutions to the typical HPC data management use cases.

Amazon Simple Storage Service

The Amazon Simple Storage Service (Amazon S3) provides virtually unlimited object storage designed for 11 9's of durability and high availability. For binary packages, it offers both versioning and various immutability features, such as Object Lock, which prevents deletion or replacement of objects. You can include this feature in your deployment pipeline to make sure that the analytics binaries you use in production are the same as those that you validated. Binary immutability is a common audit requirement in regulated industries, which require you to demonstrate that the binaries approved in the testing phase are identical to those used to produce reports. This service also offers easy to implement encryption and granular access controls.

Some HPC architectures use checkpointing (compute instances save a snapshot of their current state to a datastore) to minimize the computational effort that could be lost if a node fails or is interrupted during processing. For this purpose, a distributed object store (such as Amazon S3) might be an ideal solution. Because the data is likely to only be needed for the life of the batch, you can use Amazon S3 life cycling rules to automatically purge these objects after a small number of days and reduce cost.

Amazon Elastic File System

Amazon Elastic File System (Amazon EFS) offers shared network storage that is elastic, which means it grows and shrinks as required. Thousands of Amazon EC2 instances can mount EFS volumes at once, which enables shared access to common data such as analytics packages. Amazon EFS does not currently support Windows clients.

Amazon FSx for Lustre

For transient job data, the Amazon FSx for Lustre service provides a high performance file system that offers sub-millisecond access to data and read-write speeds of up to hundreds of gigabytes per second with millions of IOPs. Amazon FSx for Lustre can link to an Amazon S3 bucket, which makes it easy for clients to write data objects to the bucket (including clients from an on-premises system) and have those objects available to thousands of compute nodes in the cloud (see Figure 8).

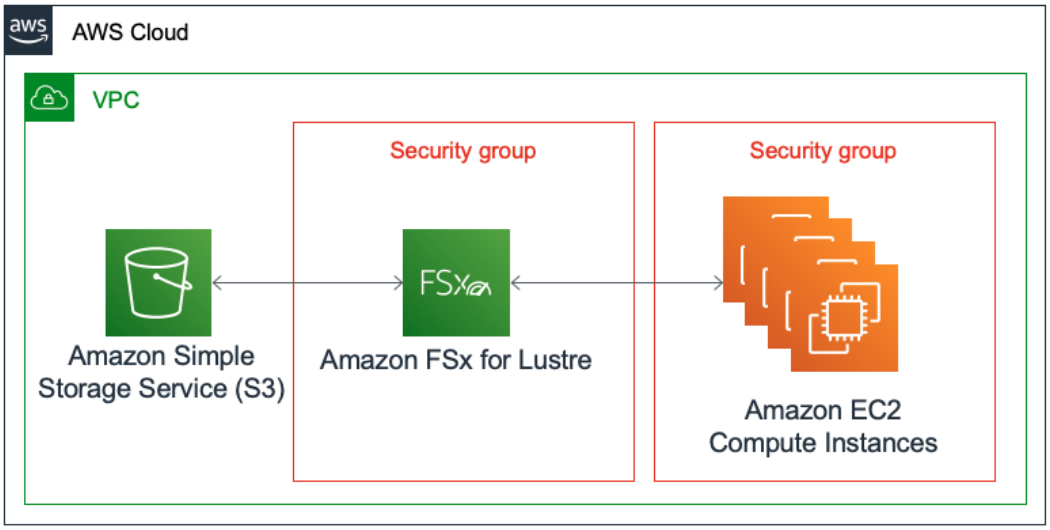

Figure 8 – An example of an Amazon FSx for Lustre implementation

Amazon Elastic Block Store

After a compute instance has binary or job data, it might not be possible to keep it in memory, so you might want to keep a copy on a local disk. Amazon Elastic Block Store (Amazon EBS) offers persistent block storage volumes for Amazon EC2 instances. Though the volumes for compute nodes can be relatively small (10GB can be sufficient to store a variety of binary package versions and some job data) there might be some benefit to the higher IOPS and throughput offered by the Amazon EBS provisioned IOPS SSD drives. These offer up to 64,000 IOPS per volume and up to 1,000MB/s of throughout, which could be valuable for workloads that require frequent, high-performance access to these datasets. Because these volumes incur additional cost, you should complete an analysis of whether they provide any additional value over the standard, general purpose volumes.

Data Management and Transfer

Although HPC systems in financial services are typically loosely coupled, with limited need for *East-West* communication between compute instances, there are still significant demands for *North-South* communication bandwidth between layers in the stack. A key consideration for networking is where in the stack any separation between on-premises systems and cloud-based systems occurs. This is because communication within the AWS network is typically of higher bandwidth and lower cost than communication to external networks. As a result, any architecture that causes hundreds or thousands of compute instances to connect to an external network—particularly if they're requesting the same binaries or task data—would create a bottleneck.

Ideally, the *fan out* point (the point in the architecture at which large numbers of instances are introduced) is in the cloud. This means that the larger volumes of communication stay in the AWS network with relatively few connections to on-premises systems.

AWS offers networking services that complement the financial services HPC systems. A common starting point is to deploy AWS Direct Connect connections between customer datacenters and an AWS region through a third-party point of presence (PoP) provider. A Direct Connect link offers a consistent and predictable experience. You can employ multiple diverse Direct Connect links to provide highly resilient, high-bandwidth connectivity.

Though most HPC applications within financial services are loosely coupled, this isn't universal and there are times when network bandwidth is a significant component of overall performance. AWS offers the Elastic Network Adapter (ENA) which is available for a number of instance types, including C5 and M5 instances. The ENA can provide up to 20Gbps of consistent, low-latency performance to other instances with a Placement Group. A Placement Group attempts to place Amazon EC2 instances physically close together in an Availability Zone to reduce network latency.

The Elastic Fabric Adaptor service (EFA) enhances ENA and is specifically engineered to support tightly-coupled HPC workloads which require low latency communication between instances. An EFA is a virtual network device which can be attached to an Amazon EC2 instance. Most suited to workloads using the Message Passing Interface (MPI) the service may be worthy of consideration for some financial services workloads such as weather predictions as part of an insurance industry catastrophic event model. EFA traffic that bypasses the operating system (OS-bypass) is not routable, so it's limited to a single subnet. As a result, any peers in this network must be in the same

subnet and Availability Zone, which could alter resiliency strategies. The OS-bypass capabilities of EFA are also not supported on Windows.

Some Amazon EC2 instance types support *jumbo frames* where the Network Maximum Transmission Unit (the number of bytes per packet) is increased. AWS supports MTUs of up to 9001 bytes. By using fewer packets to send the same amount of data, end-to-end network performance is improved.

Operations and Management

HPC systems are traditionally highly decoupled and resilient to the failure of any given component, with minimal disruption. However, HPC systems in financial services organizations tend to be both mission critical and limited by the capabilities of traditional approaches, such as physical primary and secondary datacenters. In this model, HPC teams have to choose between having secondary infrastructure sitting mostly idle in case of the loss of a datacenter, or using all of the infrastructure on a daily basis but with the possibility of losing up to 50% of that capacity in a disaster event. Some add a third or fourth location to reduce the impact of the loss of a site, but at the cost of an increased likelihood of an outage and network inefficiencies.

When you move to the cloud, you not only open up the availability of new services, but also new approaches to solving these problems. AWS operates a model with Regions and Availability Zones that are always active and offer high levels of availability.

By architecting HPC systems for multiple AWS Availability Zones, financial services customers can benefit from high levels of resiliency and utilization. In the unlikely event of the loss of an Availability Zone, additional instances can be automatically provisioned in the remaining Availability Zones to enable workloads to continue without any loss of data and only a brief interruption in service.

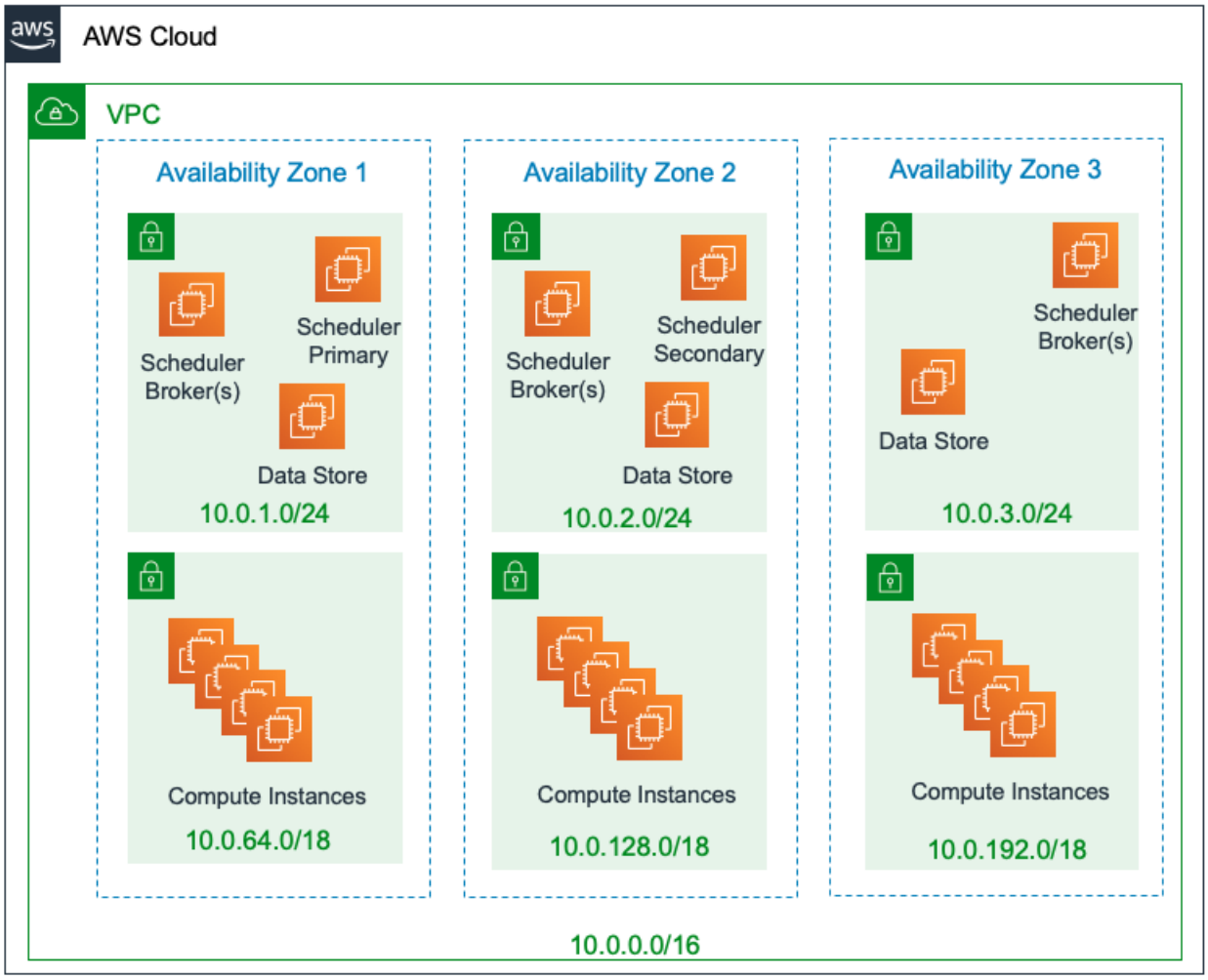

Figure 9 – A sample HPC architecture for a Multi-AZ deployment

The high-level architecture in Figure 9 shows the use of multiple Availability Zones and separate subnets for the stateful scheduler infrastructure (including schedulers, brokers, data stores) and the compute instances. You can base your scheduler instances on long-running Reserved Instances with static IP addresses to help them communicate with on-premises infrastructure by simplifying firewall rules. Conversely, you can base your compute instances on On-Demand Instance or Spot Instances with dynamically allocated IP addresses. Security Groups act as a virtual firewall, which you can configure to allow the compute instances to communicate only with scheduler instances.

One of the keys to effective HPC operations are the metrics you collect and the tools to explore and manipulate them. A common question from end users is, “Why is my job slow?” It’s important to set up your HPC operation in a way that enables you to either answer that question, or to empower users to find it for themselves.

AWS offers tools you can use to collect metrics and logs, at scale. Amazon CloudWatch is a monitoring and management service that not only collects metrics and logs related to AWS services, but through an agent, it can also be a target for telemetry from HPC systems and the applications running on them. This provides a valuable central store for your data, and allows diverse data sources to be presented on a common timeseries, and helps you to correlate events when you diagnose issues. You can also use CloudWatch as an auditable record of the calculations that were completed, with the analytics binary versions that were used. You can export these logs to Amazon S3 and protect them with the object lock feature for long term, immutable retention.

Some grid schedulers require a relational database for the retention of statistics data. For this purpose, you can use Amazon Relational Database Service (Amazon RDS), which provides cost-efficient and resizable database capacity, while automating administration tasks such as hardware provisioning, patching, and backups.

Another common challenge with shared tenancy HPC systems is the apportioning of cost. The ability to provide very granular cost metrics according to usage can drive effective business decisions within financial services.

The *pay as you go* pricing model of AWS empowers HPC managers and their end customers to realize the benefits from the optimization of the system or its use. AWS tools such as resource tagging and the AWS Cost Explorer can be combined to provide rich cost data and to build reports that highlight the sources of cost within the system. Tags can include details of report types, cost centers, or other information pertinent to the client organization. There's also an AWS Budgets tool which can be used to create reports and alerts according to consumption.

When you combine detailed infrastructure costs with usage statistics, you can create very granular cost attribution reports. Some trades are particularly demanding of HPC capacity, to the extent that the business might decide to exit the trade instead of continuing to support the cost.

Orchestration

It's common for financial services organizations to use a third-party grid scheduler to coordinate HPC workloads. As mentioned in the Overview section, these schedulers are highly optimized for making low-latency scheduling decisions to maximize usage of a fixed set of resources. When you plan a migration, a valid option is to migrate the on-premises solution first, and then consider optimizations. For example, an initial implementation might use Amazon EC2 On-Demand Instances to provision capacity, which yields some immediate benefits from elasticity. Some of the commercial

schedulers also have integrations with AWS, which enable them to add and remove nodes according to demand.

When you are comfortable with running critical workloads on AWS, you can then further optimize your implementation with options such as using more native services for data management and capacity provisioning. Ultimately, the scheduler might be in scope, at which point you can consider a few different approaches.

Though financial services workloads are often composed of very large volumes of relatively short-running calculations, there are some cases where longer-running calculations need to be scheduled. In these situations, AWS Batch could be a viable alternative or a complementary service. AWS Batch plans, schedules, and executes batch workloads while dynamically provisioning compute resources using containers. Customers can configure parallel computation and job dependencies to allow for workloads where the results of one job are used by another. AWS Batch is also offered at no additional charge, only the AWS resources it consumes generate costs.

Customers looking to simplify their architecture might consider a queue-based architecture in which clients submit tasks to a stateful queue. This can then be serviced by an elastic group of *hungry worker* processes that take pending workloads, process them, and then return results. The Amazon Simple Queue Service (Amazon SQS) can be used for this purpose. Amazon SQS is a fully managed message queuing service that is ideal for this type of decoupled architecture. As a serverless offering, it reduces the administrative burden of infrastructure management and offers seamless elastic scaling.

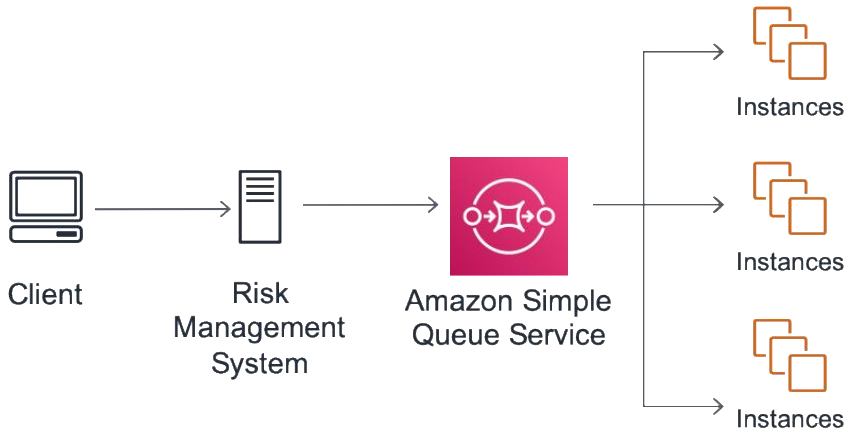

Figure 10 – A simple HPC approach with Amazon SQS

Amazon SQS queues can be serviced by groups of Amazon EC2 instances that are managed by AWS Auto Scaling Groups. You can configure the AWS Auto Scaling

Groups to scale capacity up or down based on metrics such as average CPU load or the depth of the queue. AWS Auto Scaling Groups can also incorporate provisioning strategies that can combine Amazon EC2 On-Demand Instances or Spot Instances to provide flexible and low-cost capacity.

With serverless queuing provided by Amazon SQS, it's logical to think about serverless compute capacity. With AWS Lambda, you can run code without provisioning or managing any servers. This function-as-a-service product allows you to only pay for the computation time you consume. You can also configure Lambda to process workloads from SQS, scaling out horizontally to consume messages in a queue. Lambda attempts to process the items from the queue as quickly as possible, and is constrained only by the maximum concurrency allowed by the account, memory, and runtime limits.

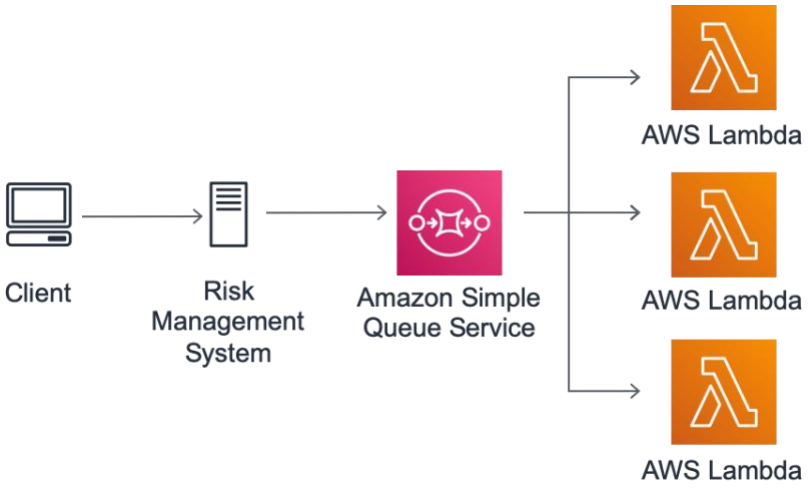

Figure 11 – A serverless, event-driven approach to HPC

When you explore these simplified approaches, especially in comparison to established schedulers, it's important to consider all of the features required to run what can be a critical system. Metrics gathering, data management, and management tooling are only some of the typical requirements that must be addressed and should not be overlooked.

A key benefit of running HPC workloads on AWS is the flexibility of the offerings that can allow you to combine various solutions to meet very specific needs. An HPC architect can use Amazon EC2 reserved instances for long-running, stateful hosts. You can use Amazon EC2 On-Demand Instances for long-running tasks, or to secure capacity at the start of a batch. Additionally, you can provision Amazon EC2 Spot Instances to try to deliver a batch more quickly and at lower cost. Some workloads can then be directed to alternative platforms, such as GPU enabled instances or Lambda

functions. You can optimize the overall mix of these options on a regular basis to adapt to the changing needs of your business.

Security and Compliance

The approach to security in High Performance Computing systems running in the cloud is often different from other applications. This is because of the ephemeral and stateless nature of the majority of the resources. Issues of patching, inventory tooling, or human access can be eliminated because of the short-lived nature of the resources.

- Patching – When you use a pre-patched Linux AMI, the host is in a known compliant state at startup. If a relatively short limit is placed on the life of the instance, it's likely that this approach will meet all necessary patching standards.
- Inventory Tooling – On-premises hosts typically interact with compliance and inventory systems. Controls around the instance image and the delivery of binaries mean that instances remain in a known state, so these controls might not be necessary. Because highly scalable and elastic resources can put excessive load on such systems, Amazon CloudTrail might provide a more suitable alternative.
- Root Access – When you enable all debugging by centralized metrics, you can mandate zero access to the compute nodes. Without any root access, you can avoid key rotation and access control issues.

When you consider migrating to the cloud, an important early step is to decide which internal tools and processes (if any) need to be replicated in the cloud. Amazon EC2 instances that are unencumbered by tooling tend to start up more quickly, which is important when additional capacity is required to meet a business need.

Because of the stateless nature of the workloads, there is often little need to store data for long periods, particularly when the job data isn't especially sensitive, doesn't include personally identifying information (PII), and largely consists of public market data sets. Regardless, encryption by default is easy to implement across a wide range of AWS services.

Because binary analytics packages often contain proprietary code that has intellectual value, you should give particular consideration the security of these assets. Financial services organizations encrypt these binaries while in transit and use built-in AWS tools to ensure they're encrypted while at rest in AWS storage. If compute instances are configured for minimal or no access, the risk of exfiltration while the binaries are in memory is minimized.

AWS has a wide range of certifications and attestations relevant to financial services and other industries. For full details of AWS certifications, see AWS Compliance.

Before you design secure systems in AWS, to make sure you understand the respective areas of responsibility for AWS and the customer, review the Shared Responsibility Model.

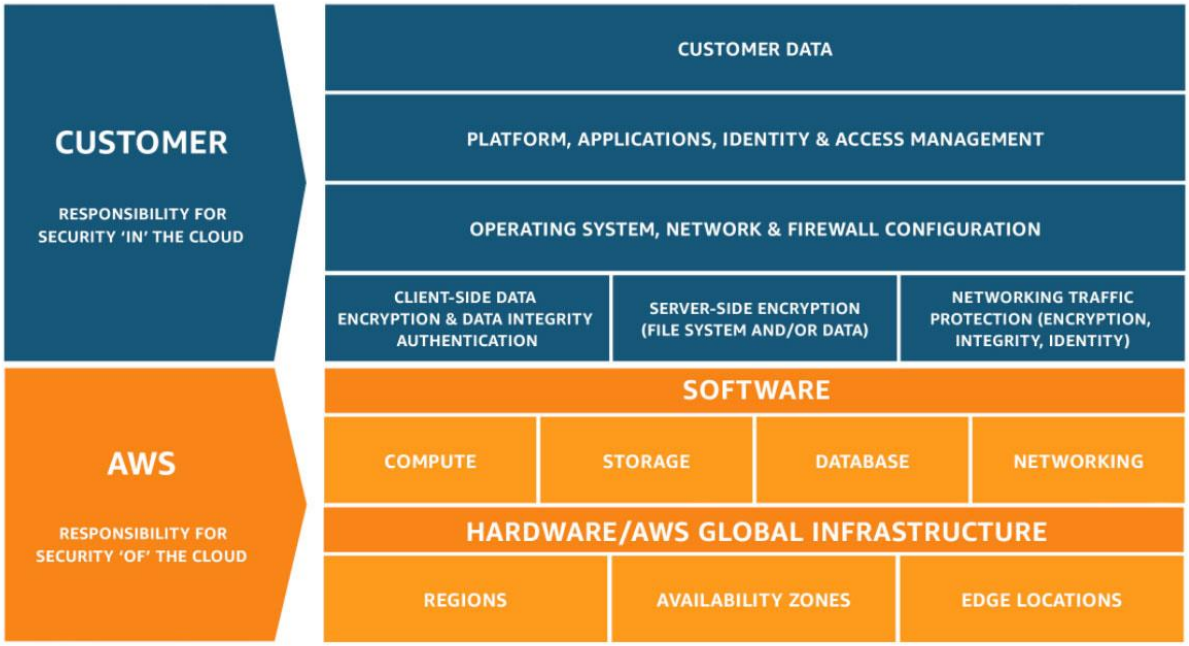

Figure 12 – The AWS Shared Responsibility Model

This model is complemented by an extensive suite of tools and services to help customers be secure in the cloud. For more detailed information, review the AWS Well-Architected Framework Security Pillar.

One service of particular interest to HPC applications is the AWS Identity and Access Management (IAM) service, which provides fine-grained access control across all of the AWS services included in this paper. IAM also offers integration with your existing identity providers through identity federation.

Interactions with the AWS APIs can be tracked with AWS CloudTrail, a service that enables governance and auditing across the AWS account. This event history simplifies security analyses, changes to resources, and troubleshooting.

Encryption by default is becoming increasing common within financial services, and many AWS services now offer simple encryption features that integrate with AWS Key Management Service (AWS KMS). This service makes it easy for customers to create

and manage keys that can be used across a wide variety of AWS services. For HPC applications, keys managed by AWS KMS might be used to encrypt AMIs or Amazon S3 buckets that contain analytics binaries, or to encrypt data stored in the Parameter Store.

AWS KMS uses FIPS 140-2 validated hardware security modules (HSMs) to generate and protect customer keys. The keys never leave these devices unencrypted. Customers with specific internal or external rules regarding HSMs can choose AWS CloudHSM, which is a fully managed FIPS 140-3 level 3 validated HSM cluster with dedicated, single-tenant access.

Migration Approaches, Patterns and Anti-Patterns

Many financial services organizations already have some form of HPC environment hosted in an on-premises datacenter. If you're migrating from such an implementation, it's important to consider what might be the best method to complete the migration. The optimal approach depends on the desired outcome, risk appetite, and timescale, but typically begins with one of the 6 R's: Rehosting, Replatforming, Repurchasing, Refactoring/Re-architecting, and (to a lesser degree) Retiring, or Retaining (revisiting).

HPC cloud migrations typically progress through three stages. The nuances and timings of each stage depends on the individual businesses involved.

The first stage is *Bursting* capacity. In this mode, very little changes with the existing on-premises HPC environment. However, at times of peak demand, Amazon EC2 instances can be created and added to the system to provide additional capacity. The trigger for the creation of these instances is usually either:

- Scheduled – If workloads are predictable in terms of timing and scale, then a simple schedule to add and remove a fixed number of hosts at predefined times can be effective. The schedule can be managed by an on-premises system, or with Amazon CloudWatch Event triggers.
- Demand based – In this mode, a component can monitor the performance of workloads, and add or remove capacity based on demand. If a task queue starts to increase, additional instances can be requested through the AWS API, and if the queue decreases, some instances can be removed.

- Predictive – In some cases, especially when the startup time for a new instance is long (perhaps because of very large package dependencies or complex OS builds), it might be desirable to use a simple machine learning model to analyze historic demand and determine when to bring capacity online. This approach is rare, but can work well when combined with a demand-based approach.

As customers build confidence in their ability to supplement existing capacity with cloud-based instances, they often make a decision to complete a migration. However, with existing on-premises hardware still available, customers want to keep the value of that infrastructure before it can be decommissioned. In this case, it can make sense to provision a new strategic grid—with all of the same scheduler components—into the cloud, and retain the existing on-premises grid. It's then left to the upstream clients to direct workloads accordingly, switching to the cloud-based grid as the on-premises capacity is gradually retired.

When customers have completed migration and are running all of their HPC workloads in the cloud, the on-premises infrastructure can be removed. At this point customers, have completed a *Rehosting* approach. When their infrastructure is in the cloud, they then have the flexibility to look at *Replatforming* or *Refactoring* their environment. The ability to build entirely new architectures in the cloud alongside existing production systems means that new approaches can be fully tested before they're put into production.

One anti-pattern that's occasionally proposed by customers involves *platform stacking*. In this approach, solutions such as virtualization and container platforms are placed under the HPC platform, to try to create portability between cloud-based systems and on-premises systems. This approach has a number of disadvantages:

- Computational inefficiency – By adding more layers between the analytics binaries and CPUs performance, computational efficiency is inevitably degraded as CPU cycles are consumed by the abstraction layers.
- Licensing costs – HPC environments are large and continue to grow. Though enterprise licenses can keep the up-front costs of using these technologies very low, the large number of CPU cores involved in HPC workloads can mean significant additional costs when the licenses are due for renewal.

- Management overhead – In the simplest approach, an Amazon EC2 instance can be created on demand using an Amazon Linux 2 AMI. This AMI is patched and up to date and because it exists for just a few hours, it requires no further management. However, by building HPC stacks on top of other abstractions, those long-running layers need patching and upgrading, and when multiple layers are involved, the scope for disruption through planned maintenance or an unplanned outage increases significantly.
- Scaling challenges – Amazon EC2 instances can be available very quickly on demand. If scaling out involves the creation of a complex stack before processes can execute, this adds to the billing time of the instance before useful work can be done. In worst-case scenarios, there can be a temptation to leave large numbers of instances running so that they're available if additional workloads arise.
- Optimization challenges – HPC systems are already complex, especially when supporting huge volumes of variable workloads with different CPU and memory requirements. Knowing where CPU and memory resources are consumed is vital to identifying bottlenecks or debugging failures. If an HPC platform is based on a series of abstraction layers, this can introduce additional variables that make it difficult to see where inefficiencies exist, and as a result they might never be found.
- Security challenges – Securing a more complex stack can be challenging because there are more components to configure, monitor, and maintain to ensure the integrity of the system.

Keeping HPC systems as simple as possible provides the best performance at the lowest cost. Most HPC solutions are already platforms by design and offer portability through simple deployment patterns to standard operating systems.

Conclusion

AWS has a long history of helping customers from various industries—including financial services—to optimize their HPC workloads. This experience over many years from customers with diverse requirements has directly contributed to the products and services offered today, and will continue to do so. AWS regularly accommodates very large-scale requests for Amazon EC2 instances.

For example, Western Digital ran a hard disk drive simulation on AWS to simulate crucial elements of upcoming head designs for their next-generation hard disk drives.

The simulation involved the completion of around 2.5 million tasks, and ran to completion in just 8 hours on a 1 million vCPU Amazon EC2 cluster. This was a significant reduction from the 20 days that the calculation would have taken on their usual cluster configuration. This offered a significant advantage in an industry driven by innovation.

In another example, a group of researchers from Clemson University created a high-performance cluster on the AWS Cloud using more than 1.1 million vCPUs on Amazon EC2 Spot Instances running in a single AWS Region. This cluster was used to study how human language is processed by computers.

High Performance Computing platforms are crucial enablers for many different types of financial services organizations including capital markets, insurance, banking and payments. However, as demands on these platforms increase as a result of regulatory demands, it's clear that the traditional approaches to provisioning HPC infrastructure are inefficient and ultimately unsustainable. Constraints on capital and capital expenditure further compound the challenge.

By migrating these systems to AWS, customers benefit from a wide variety of compute instances and relevant services, but also from a fundamental change in the delivery of compute capacity. This new approach offers tremendous flexibility, both in terms of the management of workloads that vary day-to-day, but also in the overall approach to cost optimizations, security, availability, and operations.

HPC workloads already have much in common with stateless, function-as-a-service architectural patterns. As these approaches mature, it's likely that just as financial services moved from local calculations to clusters and into grids, they will move to decentralized, *serverless* approaches. As scaling become transparent, bottlenecks will continue to be removed until processing becomes near real-time.

If you have challenges with the scale, cost, and capacity challenges of managing a High-Performance Computing system today, AWS has a number of services and partner relationships that can help.

To learn more, you can contact AWS Financial Services through the AWS Financial Services – Contact Sales form.

Contributors

Contributors to this document include:

- Alex Kimber, Solutions Architect, Global Financial Services, Amazon Web Services
- Ian Meyers, Solutions Architect Head of Technology, Amazon Web Services

Further Reading

For additional information, see:

- AWS Well-Architected Framework
- AWS Well-Architected Framework – HPC Lens

Document Revisions

Date	Description
September 2019	Updates to services, diagrams, and topology.
January 2016	Updates to services and topology.
January 2015	Initial publication.

Glossary of Terms

The following are the definitions for the terms that appear throughout this document.

Binary Package – A set of binaries that execute Tasks. A typical HPC environment can support multiple packages of various versions running in parallel. The package and version required are defined by the client or risk system at the point of job submission. These packages typically contain proprietary models that are built by the firm's Quantitative Analysis teams (*quants*) and are often the subject of intellectual property concerns as they can form competitive differentiation.

Broker – A component of a typical HPC / Grid platform. The broker is typically responsible for coordinating tasks and/or client connections to compute instances. As grids and task volumes grow, the number of brokers is typically scaled out to ensure throughput can be maintained.

Client – A software system, accessed by a user, that generates job requests and presents results. In financial services, this is generally some form of risk management system (RMS).

Engine – A software component responsible for invoking the calculation of a task using a given binary package. A compute instance can run multiple engines in parallel, perhaps one or more within each *Slot*.

Instance – An Amazon EC2 virtual server. Each instance has a number of available virtual CPUs (vCPU) and an allocation of memory.

Job (or Session) – The definition of a series of one or more related tasks. For example, a job might define a series of scenarios and how they are sub-divided into a set of tasks.

Job Data – The set of data that is required in addition to the Task metadata. Typically, Job Data is passed to the compute instance as a reference, bypassing the scheduler itself. In investment banking applications, Job Data is generally a combination of static reference data (such as holiday calendars used to calculate trade expiration dates), market data (used to build the market environment), and trade data (referencing the trade or portfolio of trades which are the focus of the calculation).

Master – A component of a typical HPC / Grid platform. The master is responsible for tracking the state of compute instances and *Brokers* as well as hosting API or GUI interfaces and metrics. The Master host is generally not involved in the scheduling of individual tasks.

Quantitative Analysts / Quants – The team associated with the development of mathematical models to predict the behavior of financial products.

Risk Management System (RMS) – To improve oversight of risk calculations, centralize operations and improve efficiency financial services firms are increasingly leveraging Risk Management Systems to sit between the users and the HPC platform.

Scheduler / Grid Scheduler – A software component responsible for managing the lifecycle of tasks through receipt, allocation to compute instances, collection of results, and metrics and management processes.

Slot – A unit of compute currency used to approximate homogeneity within a heterogenous compute environment. For example, a *slot* might be defined as two CPU cores and 8GB of RAM, and would be considered interchangeable, regardless of whether the compute instance was able to provide two or 32 slots.

Task – A unit of work to be scheduled to a compute instance. A task can define external dependencies (such as market and reference data). In recursive workload patterns, a parent task can spawn a child Job or a series of other child tasks.

Thread – An Engine runs either single-threaded or multi-threaded processes. Ideally, each thread runs on a separate vCPU to minimize the overhead of CPU context switching.

User – In financial services, a user is typically a member of the front office, either a trader managing positions or desk-head who wants oversight, and ensures successful internal or external reporting is completed.