Amazon Web Services 上的金融服务网格计算

2016 年 1 月

通告

本文档所提供的信息仅供参考，且仅代表截至本文件发布之日时 AWS 的当前产品与实践情况，若有变更恕不另行通知。客户有责任利用自身信息独立评估本文档中的内容以及任何对 AWS 产品或服务的使用方式，任何“原文”内容不作为任何形式的担保、声明、合同承诺、条件或者来自 AWS 及其附属公司或供应商的授权保证。AWS 面向客户所履行之责任或者保障遵循 AWS 协议内容，本文件与此类责任或保障无关，亦不影响 AWS 与客户之间签订的任何协议内容。

目录

摘要

Amazon Web Services （简称 AWS）为客户提供大规模计算网格，作为强大的风险运行方法并根据多种估算进行费率计算，同时缩短执行时间与运营成本。面对这些目标，传统基础设施与应用程序往往会带来一系列显著挑战。

本份白皮书面向金融服务机构内的架构师与开发人员，旨在帮助大家更为顺畅地扩展 AWS 上的网格计算。本文概述了 AWS 平台之上大型网格管理工作的最佳实践，同时提供一套参考架构，用于指导企业实现此类复杂系统的实际交付。

简介

金融服务网格计算定义

高性能计算（简称 HPC）允许最终用户利用大规模计算资源解决各类复杂的科学、工程以及商业问题，同时提供高通量以及具备延迟可预测性的网络。大多数系统中的高性能计算平台需要由多位用户共享，同时在构建、调整与维护等层面带来高昂的成本投入要求。

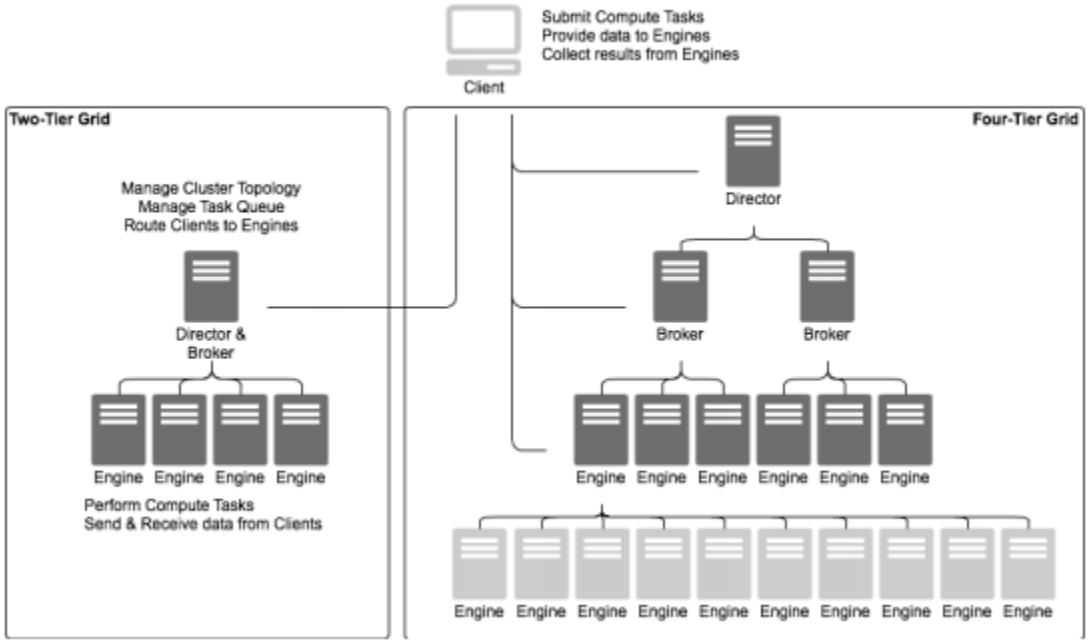
在本文当中，我们将专注于讨论金融服务行业中的高性能计算应用手段。其中包括价格计算与衍生风险、商用或者权益类产品; 交易者投资组合风险计算; 以及企业业务的整体市场定位。这些计算任务可能持续整个交易日，或者在交易日的结算结束后作为报告结果进行处理。

与其它类型的高性能计算工作负载不同，金融服务所关注的工作负载倾向于利用大规模并行架构承载一定程度的延迟水平，同时优化系统的整体数据吞吐能力。在本文中，我们将以“计算网格”这一术语指代高性能计算集群表现出的这些特征。对于其它用例，例如生命科学或者工程型工作负载，使用的具体方案将有所不同。

计算网格架构&性能

大多数商业与开源计算网格采用 HTTPS 进行通信，且可运行在通量能力及延迟不稳定的非可靠网络之上。然而，对于某些风险敏感型应用以及专有计算网格，网络延迟与传输带宽可能对整体性能产生巨大影响。一般来讲，计算网格当中包含数百乃至数千项独立进程（引擎），且运行在数十到数百台设备之上。为了获得可靠结果，各引擎倾向于采用固定的计算核心与内存搭配比例（例如每引擎双虚拟核心加 2 GB 内存）。吞吐量提升以特定时间段内能够处理的任务量而定。要提升网格的吞吐能力，我们需要添加更多引擎。

计算集群的信息由一套网格“主管”或者“控制器”负责实现，而各计算网格客户端则通过任务管理器或者“中介”将任务提交至引擎。在大多数网格架构当中，任务客户端与引擎间的数据发送操作可直接进行，但也有一些架构采取利用网格中介发送数据的方式。在一部分架构（如图一所示）中，即双层网格，主管与中介的功能由单一组件负责管理。而大规模三层网格中的单一主管则可对应多套中介，其中每个中介负责引擎的一个子集。在规模最大的网格，即四层架构当中，每套引擎都有能力跨越至其它引擎当中的多项任务。



图一：双、三与四层网格拓扑结构

网格计算中的时间帧往往以分钟或者小时为单位，而非毫秒。不同引擎间的计算操作会分别且并发进行，进而建立起一套非共享架构。各计算客户端间的通信往往能够接受相对较高的延迟水平，并可在发生错误时进行重试。

云环境下网格计算的优势

利用 AWS，大家能够根据需要分配计算容量，而无需规模数据中心、风格以及服务器基础设施等前期准备工作。您可以立足上体需求访问包含不同 CPU、内存、本地磁盘及网络连接配置的多种实例类型。基础设施可运行在任意数量的全球服务区之内，意味着客户将彻底摆脱漫长的合同谈判与本地供货周期。这种方法能够显著提升交付速度，特别是在新兴市场当中。

利用 Amazon Virtual Private Cloud（即虚拟专有云，简称 VPC），大家可以定义一套与自有数据中心内传统网络体系高度相似的虚拟网络拓扑结构。您可以对该虚拟网络加以完全控制，包括选择 IP 地址范围、创建子网（VLAN）以及配置路由表与网络网关等。通过这种方式，大家能够建立起切实满足隔离性需求、内部合规审计安全规范或者外部监管条款的网络拓扑方案。

由于 AWS 自身的按需资源分配特性，大家可以根据不同业务线要求分别构建网格与基础设施，或者出于成本优化的考虑进行基础设施共享。另外，其弹性容量与基础设施动态变更能力还允许大家在特定时间点将多数容量分配给特定负载，而非像传统方案那样只能选择配置峰值利用率。另外，轻松的规模向上扩展能力能够轻松解决黑天鹅事件，从而适应企业经营者应对各类市场变化的能力。

在 AWS 之上对由数百节点构建的计算网格进行任务运营的流程亦得到显著简化，这是因为大家可以充分利用自动化手段处理配置、任务规模伸缩等工作，并将基础设施视为代码库的组成部分。AWS 资源可利用 AWS 管理控制台进行交互式控制，通过 API 或者 SDK 进行编程，甚至利用第三方运营工具进行管理。大家能够利

用内部角色定义或者成本中心进行实例标记，从而立足于运行时与账单实现可视化与透明化。负责监控 AWS 资源的 Amazon CloudWatch[1] 能够提供面向整体网格利用率的可视化效果，同时以细粒度方式调整个别实例或者数据存储机制。

大家可以将弹性计算容量与其它 AWS 服务相结合，从而降低计算客户端的复杂性水平。举例来说，Amazon DynamoDB 作为一项完整的 NoSQL 数据库管理服务，能够以无缝化可扩展方式提供快速与可预测的性能成效，进而立足任意规模及吞吐容量[2]从计算网格当中获取结果。QA 库等敏感数据产品可以安琪方式进行存储，并轻松利用 AWS CodeDeploy[3] 服务实现部署。Compute Engines 则可享受到各类高性能文件系统带来的优势，例如英特尔 Lustre Cloud Edition（已在 AWS Marketplace 中正式上架）可用于临时文件存储与工作区交付[4]。在计算任务完成之后，相关资源会被中止且不再带来任何成本。

目前众多企业都在积极寻求新的方式，旨在以更低成本执行计算敏感型任务。快速配置、最低程度管理以及灵活的实例规模与容量伸缩，外加初创型第三方网格协调与数据管理支持，这一切使得 AWS 平台成为一套引人注目的解决方案。

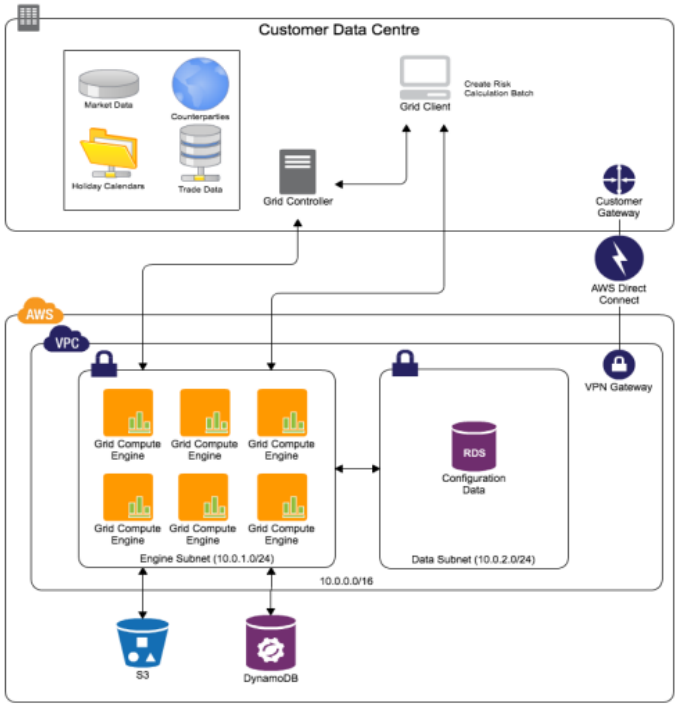
AWS 上的网格计算

初步实施

大家可以尝试将部分计算网络迁移至 Amazon Elastic Compute Cloud（即弹性计算云，简称 Amazon EC2）[5] 之上，从而初步建立起一套高性能计算网格概念验证或者实现方案。要实现网格管理与 QA 软件，大家可以选择 Amazon Machine Image（简称 AMI）[6]，而网格计算会通过 Amazon VPC[7] 经由 VPN 连接参考现有各动态数据源。

这套架构当中的 VPC 配置方式非常简单，仅包含单一子网以容纳各引擎实例。AWS DirectConnect[8] 用于确保延迟及吞吐量的可预测性，从而在大量客户数量向网格引擎进行传输时提供更为稳定的执行表现。大家也可以利用 Amazon Relational Database Service（即关系数据库服务，简称 Amazon RDS）[9] 或者 Amazon DynamoDB 对这些数据进行配置或者操作。

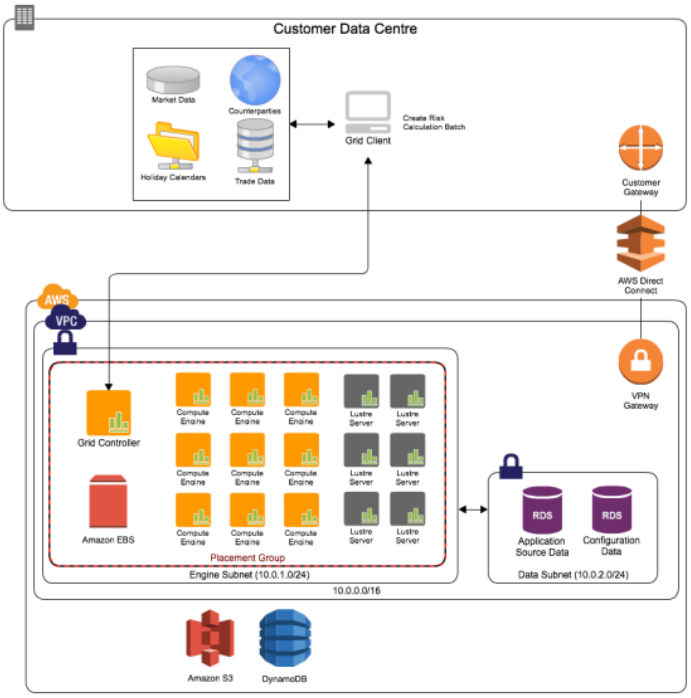
图二所示的架构用于将现有网格计算任务简单扩展到一套弹性及可扩展平台之上。由于仅在存在必要计算任务时才在 Amazon EC2 之上运行引擎，大家可以有效降低成本并提升网格基础设施的灵活性水平。



图二：AWS 上网格引擎初步实施示意图

完整云实施

参考架构



图三：网格计算参考架构

图三为一套参考架构示例，大家亦可通过 http://aws.amazon.com/architecture 中的 AWS 架构中心进行查看，同时遵循以下最佳实践：

安全性

客户、贸易与市场数据的安全性对于客户以及 AWS 都非常重要。AWS 的构建服务时严格遵循安全最佳实践，在此类服务当中提供对应的安全功能，外加与功能相关的指导说明文件。另外，AWS 客户还必须利用这些功能与最佳实践构建并妥善保护应用程序环境安全。AWS 管理一套综合性控制环境，其中包含必要的策略、流程以及控制举措，用于交付各类 Web 服务方案。截至目前，AWS 已经符合多种认证并得到众多第三方认可，其中包括 SOC1 Type 2、SOC 2 以及 SOC 3 合规性，PCI DSS Level 1 合规性，ISO 27001 以及 9001 认证，以及 FISMA Moderate 授权。要了解更为完整的 AWS 认证与证书列表，请通过 http://aws.amazon.com/security 访问 AWS 安全中心。

需要强调的是，AWS 在云环境中采用了一套责任分担模式。作为客户，大家可以利用安全的底层基础设施与基础服务构建安全解决方案，同时亦需要负责对操作系统、平台及应用程序的安全进行设计配置与管理，而后全面负责并控制信息安全管理系统、安全策略、标准以及规程等等。另外，大家也需要承担起云解决方案在相关监管、法规与控制框架之内的合规性保障工作。

AWS 身份与访问管理（简称 IAM）[10] 提供一套强大的解决方案，可用于管理用户、角色以及群组等有权对特定数据源进行访问的对象。大家可以为用户及系统提供单独凭证与身份，或者根据实际访问需求，利用 Amazon Security Token Serivce（即安全令牌服务，简称 Amazon STS）[11] 在特定时间段内对其访问凭证进行临时配置。利用各标准 Amazon IAM 工具，大家可以建立起细粒度访问策略，最终满足云资源层面的各类安全要求。

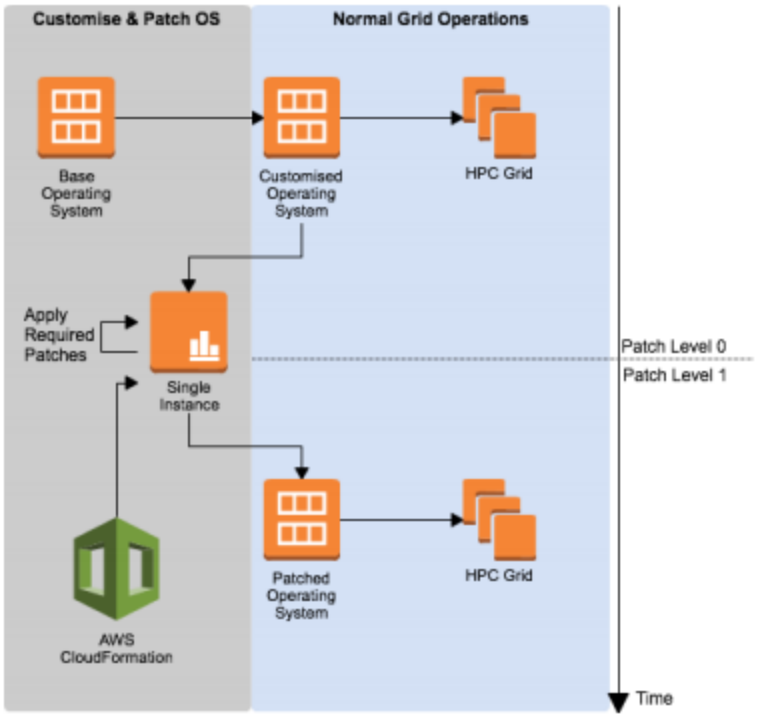
除了根据需求提供个别账户或者分配临时凭证，大家还可以将用户身份导入至现有身份与访问管理系统当中，例如 Active Directory 或者 LDAP。AWS Directory Service[12] AD Connector 允许大家接入现有微软 Active Directory，从而帮助用户以无缝化方式利用现有业务凭证访问 AWS 资源与应用程序。另外，Simple AD 提供一套独立的管理目录，由 Samba 4 Active Directory 兼容服务器负责支持。

除了密码验证之外，AWS 还为 Web 控制台与 API 访问提供多因素验证（简称 MFA）支持。MFA 能够同时提供安全验证机制，并强化我们对 AWS 资源的访问控制能力。举例来说，大家可以利用这项功能避免 Amazon S3 中存储的数据被意外删除——例如仅允许通过了 MFA 验证的用户执行对象删除操作。

合作伙伴解决方案同样适用于加密网络叠加层、高权限用户访问与联合以及入侵检测系统等，并允许内部安全控制框架与信息安全管理系统扩展至 AWS 云当中。

补丁安装

虽然 AWS、微软、红帽乃至其它操作系统供应商都将向 AWS 提供新型 AMI，但大家仍然需要确保特定 EC2 实例根据内部安全与合规性要求及控制机制安装了必要的更新补丁。要实现这一控制，大多数客户可以使用如图四所示的“滚动 AMI”架构。



图四：滚动 AMI 架构示例

从基本的 Amazon Linux、红帽或者微软 Windows 等 Amazon Machine Image 选项起步，客户将一次性构建起一套符合内部补丁安装、服务启用/禁用、防火墙配置、其它操作系统配置以及入侵检测/防御软件安装需求的“当前”定制化操作系统。此 AMI 随后会被用于启动 HPC Grids（或者该 Grid 中的特定层）。当需要使用某种特定补丁安装机制时，AWS CloudFormation[13] 可用于立足于“当前”AMI 建立单一实例，应用对应的操作系统补丁，而后创建一套新的已安装补丁的“当前”AMI，并以此为基础启动各新的 Grid 组件。完成之后，Patch Level 0 的 AMI 可通过标记策略等方式被弃用。

网络

对于金融服务网格而言，除了 IAM 政策控制之外，其相关最佳实践还包括利用 Amazon VPC 在 AWS 内部创建一套逻辑隔离型网络。这种作法将使得 VPC 之内的实例通过预设的 IP 地址分析模式进行建起。大家也可利用子网及路由表启动由源数据系统指向 AWS 网格以及客户平台的特殊路由体系，并利用虚拟机管理程序层级的防火墙（VPC Security Groups[14]）进一步缩小攻击面。大家的站点随后可轻松利用 VPN Gateway 软件或者硬件设备 [15] 进行接入。我们也能够利用静态或者动态路由协议在内部与云环境之间进行单一路由域及动态可达信息传递。

另一种理想的网格计算网络功能，在于利用 Cluster Placement Groups[16] 以在多个 EC2 集群计算实例之间平分 10 GB 网络，在各网格引擎节点间提供可靠的网络延迟水平。我们建议需要运行网格集群以及其它任何高性能文件系统节点的客户选择这种办法。

静态数据源分发

我们可以通过多种方式进行静态数据源分发，例如在执行计算的实例中使用网格管理软件、假期日历以及 gridlibs 等。另一种选项是在 AMI 当中预先安装并绑定此类组件，这样当实例在这里启动之后即可实现更快

的启动时间。然而，由于 gridlibs 需要利用补丁进行更新，而假期日历则经常变更，意味着这些 AMI 将因此而必须进行重构。这一过程从运营角度来看往往需要耗费大量时间。相反，我们应当考虑将静态数据源保存在内部源当中，例如使用一套文件系统、Amazon S3[17] 或者使用 AWS CodeDeploy，后者能够自动加密软件资产。在此之后，将这些组件安装在实例中并利用 AWS CloudFormation[18]、shell 脚本、ELC2 Config、EC2 Run 命令 [19] 乃至其它系统配置工具实现实例启动。

访问动态数据源

动态数据源包括市场、贸易以及交易对象数据等等，我们可以利用 Amazon RDS 或者 Amazon DynamoDB 等服务将其安装地存储在 AWS 云或者英特尔 Lustre 等高性能文件系统当中。这些解决方案能够显著降低备份与恢复、可扩展性以及弹性等要求的运营复杂性。要求高读取吞吐量的数据集——例如随机种子数据或者交易对象数据——最好存放在 DynamoDB（一套分布式文件系统）或者数据网格当中，从而以向外扩展方式实现读取 IOPS 的可配置能力。客户配置、调度或者网格元数据等数据集亦可作为简单属性文件、xml/json 配置或者二进制调文件被直接存储在 Amazon S3 当中。

如果在任务执行过程中，大家需要从中央市场数据系统或者贸易存储方案内将大量动态数据传输至各计算网格，则应当使用 AWS Direct Connect[20] 以确保吞吐量与延迟的可预测性。Direct Connect 会立足内部系统建立一条指向 AWS 的专用网络连接。在大多数情况下，这种作法能够削减网络成本，同时为计算网格引擎提供更具一致性的网络性能体验。举例来说，在延迟敏感型应用当中，更趋一致的底层数据源访问性能将变得极为重要。

集群设备可用性&工作流

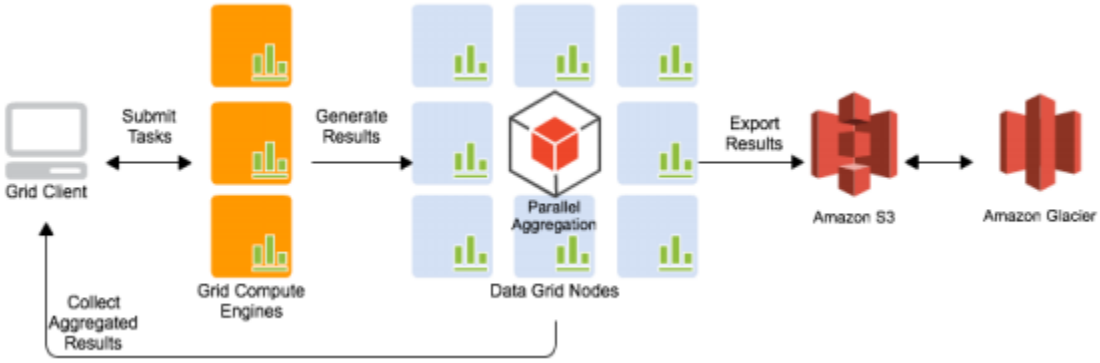
AWS 平台的弹性与按需特性允许大家仅在必要时运行网格基础设施。与传统数据中心内始终开启并运行网格服务器的作法不同，AWS 实例可被关闭以显著节约成本。这种新模式要求各实例在网格部署并上线之前即启动完成，并在网格不再活动时将各实例关闭。

很多客户发现他们需要利用定制化软件组件管理实例启动工作，而这些系统可能已经部署到位并用于为大规模网格安装软件，同时提供良好的实例配置集成点。为了实现成百上千套实例[21]的启动与可用性，客户应当选择 AWS cfn-cluster[22] 或者 MIT Starluster[23] 等开源工具，同时亦可选择 Bright Cluster Manager[24] 或者 Cycle Computing CycleCloud[25] 等来自 AWS 商业合作伙伴的解决方案。这些解决方案的一大突出优势，在于能够利用 Spot Instances[26] 将运营所需资源控制在最低水平。另外，部分网格计算平台，例如 Tibco Silver Fabric[27] 等，能够通过其内部配置工作流对建立在 Amazon EC2 上的基础设施加以管理。

无疑具体使用哪种软件，网格本身都必须先于负责提交任务的客户端完成启动及上线，因此使用可用实例数量及网格组合指标就成为构建云体系时必须认真考虑的重要前提。

结果聚合&客户端伸缩

跨越成百甚至上千套网格计算引擎的网格计算体系会向客户端返回大量数据集，这就要求我们投入大量工程精力以确保客户端能够进行对应的规模伸缩，从而在规定的时间段内收集全部计算结果。这种复杂性因素的存在，迫使很多金融服务客户选择直接将数据网格建立在计算结果的存储位置，并在这里利用并发聚合或者映射/规约执行最终计算。



图五：任务创建与结果聚合流程图

大型结构集也可被导出至 Amazon S3（如图五所示）以实现历史访问，随后则可被归档至 Amazon Glacier[28]——一项成本极低且安全稳定的数据归档与备份存储服务。大家可以在各业务部门之间进行数据共享、利用这部分数据进行回溯测试，或者允许外部合作方进行访问甚至利用身份与访问管理（简称 IAM）策略交由监管机构审查。

高可用性

AWS 服务区被拆分成多个可用区 [29]，各可用区处于不同位置以确保故障发生时负载可转移至其它可用区，亦能够确保同一服务区内的客户可以低成本、低延迟网络连接接入其它可用区。利用多个可用区建立高可用性与故障隔离架构是一项值得采纳的最佳实践。网格计算架构亦不例外，但考虑到网格执行效率保障要求，大家最好将全部组件运行在单一可用区当中。通过这种方式，数据将无需在不同物理站点之间往来迁移，因此运行时间也能够大幅降低。

大家应当在网格集群实例 [30] 启动时使用多个可用区，意味着使用单一服务区中的全部可用区。一旦发生可能影响到当前运行中网格的罕见可用区故障，集群管理人员将有能够在其它备用可用区中重建网格基础设施。而当原本网格再次上线并恢复运行后，任务即可重新提交至此进行处理。

可重复组合

对于任意构建在 AWS 之上的架构而言，一项最佳实践在于采取可重复资源组合。AWS CloudFormation 能够以模板驱动形式创建本份白皮书中提到的全部 AWS 技术，且可被用于快速按需创建开发环境，从而实现新功能、紧急修复或者并发交付流等效果。这种作法还使得我们能够利用自动化构建系统在性能与功能测试过程中创建基础设施。

计算区&位置

AWS 为客户提供充分的灵活性，允许大家选择将基础设施部署在位于北美洲、南美洲、欧洲以及亚洲的全球各服务区与可用区当中。凭借这种立足于多个位置托管全球动态数据源的能力，计算执行将近可能贴近数据所在位置以降低由网络数据传输造成的处理延迟。另外，大家也可以选择将计算任务运行在最具成本优势的位置。最后，大家亦可以选择将计算任务运行在监管与合规要求最符合实际情况的位置。事实上，大规模网格还可以跨越多个 AWS 服务区进行构建，从而同时满足速度与成本两方面要求。

关于实例类型的注意事项

Amazon EC2 提供多种实例类型可供选择，其中包括用于简单测试的小型实例、拥有高 CPU 或者内存资源比例的实例、采用固态存储驱动器（SSD）以承载随机 I/O 敏感型工作负载（例如数据库）的 I/O 优化型实例以及集群计算优化实例等等。面对大型中间结果的网格计算一般能够在 High Memory（高内存容量）实例中获得良好运行效果，而处理器敏感型高强度计算任务则比较适合交由高 CPU/内存比例实例打理。对于大规模并发型任务，我们建议大家选择 C4 实例类型，其提供一块基础主频达 2.9 GHz 的英特尔至强 E5-2666 v3（Haswell 架构）处理器，且时钟速率最高可利用英特尔睿频加速技术的支持下达到 3.5 GHz（可点击此处查阅完整参数信息）。C4 是我们 Cluster Compute Family（集群计算家族）实例中的组成部分，旨在通过高速处理器与高传输带宽、低延迟网络为客户提供高性能实例选项。对于 QA 模型等适合利用 GPGPU 提高效率的任务，大家可以选择 CG1 与 G2[31] 实例，它们可将高性能 CPU 与各自拥有 1536 个 CUDA 核心及 4 GB RAM 的多个 GPU 单元相结合。而在高性能数据库工作负载方面，hi1/i2 High I/O 实例则可在 SSD 的支持下提供极高 IOPS 表现。

Spot Instances

Spot Instances 属于 AWS 上的一种低成本规模伸缩计算网格选项。大家能够以投标方式通过高于当前 Spot Price 的价格获得 Amazon EC2 实例的使用权——Spot Price 会根据实际供求关系而不断变化。大家可以利用 Spot Instances 以提升网格当中的引擎数量，从而以更低成本加快处理速度。

图六所示为 Spot Instances 网格增强示例。

图六：使用按需与 Spot Instances 相结合的网格引擎

在 Amazon EC2 当中，以一小时为周期运行 100 套实例的成本等同于以一百小时为周期运行 1 套实例。而在使用 Spot Instances 时，这种按需模式的时间周期被大幅缩短。在真实场景当中，Spot Intances 往往能够将整体运行时间缩短 50%，而整体资源使用成本亦降低 20%。

另外，Spot Fleet 也是 EC2 提供的一项功能，允许大家以 CPU 与 RAM 的形式描述整体容量并设定所需要的实例类型。Spot Fleet 随后会自动通过 Spot Intances 交付与容量要求相符的实例，且成本同样非常低廉。欲了解更多与 Spot Fleet 相关的信息，请参阅 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html，Spot Instaces 的相关信息则可参阅 http://aws.amazon.com/ec2/spot-instances。

总结思考

Amazon Web Services 提供一套强大的机制，可供金融服务企业在内部管理基础设施之外扩展网格体系的风险与计费控制方式。无论是立足于 Amazon Elastic Compute Cloud 的简单网格引擎扩展，还是利用 AWS 托管完整的计算网格，大家都能够在更短时间周期内处理更多数据总量，而无需承担任何基础设施设置成本。另外，使用 Amazon Dynamo DB 以及英特尔 Lustre Cloud Edition 等服务，大家能够轻松简化用于管理这些大规模环境的代码库，同时利用 Spot Instances 进一步降低实现成本。由此带来的结果是，我们将能够以更高频度处理更多数据，建立远超过以往的估算与合约期数量，而成本亦得到有效削减。立足于此，大家将最终摆脱传统大规模创新投入的束缚，同时制定出更为明智的业务决策。

词汇表

Gridlibs

Grid 库是指包含可执行代码或者静态参考数据的软件包，其必须被复制到各计算引擎当中以实现计算执行。其中包含用于定义风险或者计费模式（QA 库）的应用代码，外加由假期日历用于计算截止日期的二进制数据。这些元素通常会频率变更（每季度 2 到 3 次），但自身规模比较小。它们在计算过程中会被频繁引用，因此必须在引擎上拥有直接可用性。

QA 库

用于构建分析模型的软件，该模型则负责为证券或者金融证书进行风险或者价格计算。大家可以查阅 http://en.wikipedia.org/wiki/Quantitative_analysis_(finance) 以了解更多与定量分析相关的信息。

引擎

作为计算网格组成部分负责执行计算的软件组件。引擎并不会指引整体客户端计算的流程，而是利用接收到的 QA 库、gridlibs 以及静态与动态数据源单纯执行必要操作。

静态数据

静态数据源通常体积很小且变更频率不高（也许是每年数次），但却会在风险或计费计算中被大量用到。其往往被直接部署到引擎当中，而非在运行时进行访问。

假期日历

用于计算证券或者金融证书到期日期的静态数据。一套假期日历负责提供交易日的日期信息、公共节假日以及周末等信息。

合约期

这项指标用于指定金融证书或者估算的到期日期。风险计算将设定为期 1 周、1 个月、3 个月、1 年乃至 10 年的合约期。风险值则会在每个假想到期日期内进行显示。

贸易数据

作为计算对象的贸易或者投资组合数据。其中包含的值、期限以及信息用于描述必须进行计费的合约期（剩余期限）。此数据通常在每项计算及每套引擎内以惟一形式存在，且必须在每次计算运行时重新提供。

市场数据

市场数据被用于计算以理解市场活动的目的，并以此为基础进行风险预测。大多数风险与定价系统只会周期性执行，而计算结果则每 30 分钟或者数小时进行一次变更。此数据可进行缓存以提高市场活动更新时的执行效率。

交易对象数据

交易对象数据通常会在进行企业层级或者集团层级风险计算时使用。举例来说，一天之内的 P&L 通常会立足于企业内的多个贸易层级进行聚合。此数据变更频率很低，但自身规模往往非常庞大。

P&L

利润&亏损。

随机种子

蒙特卡洛分析或者回溯测试所使用的随机输入数据。这部分数据集必须提前生成并分发至全部引擎。此数据在给定的模型计算流程当中以惟一形式存在，且通常规模很大。

蒙特卡洛分析

蒙特卡洛方法是经由计算算法对随机数字处理后得到的结果。随机数字集的规模越大，结果就越准确。这项技

术被广泛应用于近似概率模拟或者随机数试运行。

回溯测试

回溯测试用于过去某历史时期内所使用策略的有效性，并据此确定预期结果。

网格管理软件

网格管理软件确保设备与引擎可用于实际计算执行与任务分发控制。此软件通常负责分发 gridlibs 并提供一套管理控制台，用于变更网格元数据、规模、优先级以及共享设置。网格管理软件通常会由业务根据实际应用进行定制化构建，但市面上亦存在大量第三方产品可供选择。

网格客户端

此客户端软件用于编排计算网格，而后者将成为整体架构的核心组成部分。由于通过多种不同语言编写而成且可运行在任意平台之上，此软件必须能够消化所有架构组件并确保在正确的时间对数据进行正确的计算。此软件在每家企业甚至业务线中以惟一形式存在，且对性能与规模扩展能力有着明确要求。

无共享（简称 SN）架构

在这套架构模式中，每套计算环境都独立于其它计算环境保持运行。内存与存储资源的使用与维护由当前计算环境负责。与之对应的则是共享磁盘模式，其中每套计算环境利用共同的数据存储体系。

短周期风险

指当市场变化频率极高（每 15 分钟到 1 小时变化一次）时，企业需要顺应每次市场变化进行定位计算。这些系统往往也会在 1 天或者 1 周之内显示出对市场走向的细粒度影响效果。

废物利用

能够将计算网格扩展至低可靠性或者周期性不可用基础设施当中，旨在实现增加吞吐能力的目标。通常由超大规模网格体系采用，用于利用长期处于闲置的内部桌面设备。

贡献者

以下个人及组织为本份白皮书的撰写做出贡献：

- AWS 公司，Ian Meyers

文档修订

本白皮书在最新版本中添加了以下内容：

- 变更拓扑选项，添加了 2、3 与 4 层网格。
- 移除 EMR 相关引用，并替代为英特尔 Lustre。
- 添加了与补丁安装流程相关的章节。
- 添加了与 cfn-cluster 相关的网格编排相关引用。
- 添加了 tibco silver 相关引用。
- 添加了 c4 与 g2 实例类型的细节信息。
- 对生命科学/工程技术高性能计算做出比较。

参考

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

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

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

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

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

[6] *请参阅 https://aws.amazon.com/amis 获取完整列表*

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

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

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

[10] *欲了解更多信息，请参阅 aws.amazon.com/iam*

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

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

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

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

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

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

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

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

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

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

[21] *访问 http://aws.amazon.com/hpc-applications, section 'Leverage a Vibrant Ecosystem'以查看集群管理解决方案清单*

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

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

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

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

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

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

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

[30] *请参阅集群设备可用性章节内的最佳实践以获取更多信息*

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