HongKe 虹科

NIMBIX

白皮书

Kubernetes上的HPC

实用且全面的方法

简 介

2012 年，Nimbix 开始使用 Linux 容器运行 HPC 应用程序，并于 2013 年推出了世界上第一个名为JARVICE™的容器原生超级计算云平台。不论过去还是现在，这个平台在各个方面都很新颖。首先，它直接在裸机而不是虚拟机上运行应用程序，利用 Linux容器来实现安全和多租户隔离，以及工作负载移动性。其次，它以软件即服务的风格提供了商业和开源工作流的现成服务目录以及平台即服务接口，供开发人员和 ISV使用自定义应用程序和工作流程。在当时以及在今天，很大程度上，多数高性能云平台利用管理程序虚拟化，主要提供基础设施即服务接口—该机制适用于信息技术专业人员而非科学家或希望直接使用 HPC的工程师。由JARVICE提供支持的 Nimbix云克服了虚拟化处理的性能损失以及以IT为中心的接口（如 IaaS）易用性挑战。此后，JARVICE软件作为企业平台（称为 JARVICE XE）发布，在本地或第三方基础设施上使用，但保留了 Nimbix云本身（在裸机、计算加速器和低延迟互连上运行时）的所有可用性和性能优势。

Nimbix开始在容器中部署工作流时，既没有用于打包应用程序的标准稳定企业级格式，也没有用于部署所述机器上容器的可用编排机制。转瞬到今天，我们认为Docker和Kubernetes等技术都是理所当然的。但在此之前，Nimbix必须开发机制和“技艺”，才能将产品推向市场。此后，Nimbix云和 JARVICE XE 以无数不同的形式运行了数百万个容器化工作负载，解决了几乎所有行业和垂直领域的实际 HPC问题。2019年，Nimbix发布了 HyperHub™这一加速和容器化应用市场，作为交钥匙工作流交付给JARVICE平台，无论该平台是被什么基础设施驱动。得益于JARVICE，科学家和工程师不仅可以无缝地使用容器化的HPC，而且支持这些用户的 ISV可以盈利并安全地分发他们的代码，而无需重新开发系统来这样做。

在一些相关的背景下，过去几年开始出现各种容器 Web服务平台，最著名的是 Kubernetes。谷歌于2014年向全世界发布了开源 Kubernetes平台，作为它内部使用的工具的演变，用于扩展基于Web的应用程序，例如 Gmail。自上而下，Kubernetes旨在扩展基于（主要是）负载均衡器和Web代理（称为“入口”控制器）背后的 Docker式容器的 Web服务。该架构非常适合提供无状态请求/响应类型的服务（例如RESTful API和其他Web应用程序）。通过按照标准化实践自动化部署模式，它真正简化了所述应用程序的开发。

正如 JARVICE 并非旨在为无状态 Web 应用程序提供服务一样，Kubernetes 也并非旨在运行 HPC 和其他紧密耦合的工作流程。这两个平台都将容器用于应用程序的运行时，但它们支持的工作负载和计算方法截然不同。然而，在提高运营效率的标准化世界中，为不同类型的应用程序维护不同类型的平台并不理想。IT 组织越来越希望使用“分层蛋糕”方法进行整合——例如“基础设施层”应该能够

运行任何类型的应用程序，以减少对昂贵的专业实践和额外劳动力的需求。

更高级别的“层”是为了满足具体需求，但还要依靠底层来满足基础需求。单一用途平台通常会被淘汰，取而代之的是通用平台—参考自 1970 年代以来 Unix 和 Linux 系统的兴起以及它们如何越来越多地取代大型机。除了最敏感的政府和研究类型的部署之外，通用平台也应该开始取代传统的HPC。缺失的环节是统一技术，可以在公共基础设施管理层上启用具有更具体的科学和工程应用程序的商品企业应用程序。正如本白皮书将展示的那样，JARVICE XE 与 Kubernetes 相结合提供了一种实用的方法来实现这一点。

目 录

1. Linux 容器基础

Linux 容器，最常被格式化为 Docker 容器，可以被认为只是应用程序位的运行时环境。与虚拟机不同，容器与其他容器共享主机操作系统内核。这使得容器成为运行应用程序更轻便的机制，因为它们不需要将整个操作系统堆栈与内核和驱动程序打包在一起。相反，重点是封装的应用程序及其运行所需的依赖项（共享库、配置文件等）。容器可以被认为是一种应用程序虚拟化，传统上虚拟机管理程序语境中的虚拟化是机器级别的（因此术语为“虚拟机”）。

静态容器提供应用程序运行所需的文件（二进制文件、脚本、配置文件和库），以及一些描述如何组装容器运行环境本身的元数据。在 Docker 术语中，容器被打包为可以单独缓存在目标系统上的堆叠层，但由运行时环境来组装容器化应用程序所运行的文件系统。

在运行时，容器提供 3 种基本机制：

1. 文件系统——通常表现为一个完全组装的“监狱”，容器化应用程序无法“逃脱”；这也意味着应用程序需要的一切都必须存在于这个文件系统中，因为它不能轻易地访问底层主机来利用现有的文件和目录（除非明确告诉运行时环境从主机“绑定”文件和/或目录—— 由于其固有的安全问题，这一做法应谨慎执行）。

2.“命名空间”——除了文件系统“监狱”之外，重要的是容器不能“看到”进程、网络接口和它们自己语境之外的其他对象。使用命名空间允许应用程序彼此隔离以及与主机系统隔离运行。一个安全的容器化平台，如 JARVICE 或 Kubernetes，将允许在每个主机上运行多个容器，而这些容器化应用程序甚至不知道彼此存在，也不知道那些可能直接在底层主机操作系统上运行的应用程序。

3. 访问控制和资源限制——在 Linux 中，实现这一点的主要机制被称为“cgroups”。容器与虚拟机相比的主要优势之一是系统设备可以轻松从主机“传递”到容器。与虚拟机不同，不需要复杂的半虚拟化或总线级仿真方法，因为应用程序共享相同的主机内核。因此，可以连接计算加速器等设备（FPGA、GPU 等），轻松且无开销地传输到容器化应用程序。但这也必须非常小心地管理，因为它可能导致容器化应用程序不受限制地访问主机系统上的任何资源。 Cgroups 允许容器平台以非常细粒度的控制和最小的开销来限制对设备以及一般系统资源（如内存和 CPU 周期）的访问。

2. 容器生态系统介绍

2.1 容器注册表

容器镜像（或静止容器）的远程访问对象存储；通常以 RESTful API 为前端，以便从客户端轻松访问。流行的容器注册表包括 Docker Hub 和 gcr.io。此外，Nimbix HyperHub 为各种容器注册表提供工作流级元数据和授权，作为自动跨集群同步应用程序的统一接口。

2.2 容器格式

容器化应用程序打包的实际标准是 Docker，但也存在其他格式（例如 Singularity，见下文）。格式化的镜像被“推送”到容器注册表并由容器运行时“拉取”。

2.3 容器运行时

Docker 再次成为最受欢迎的容器运行时，它实际上拉取并运行容器化应用。但是开放容器计划 (OCI) 允许其他引擎，例如“containerd”，无需修改即可运行 Docker 样式的容器。考虑到这种格式的流行和广泛采用，在平台上运行何种容器引擎与它是否可以支持 Docker 样式的容器无关。

2.4 容器平台

如果容器是容器原生环境中的应用程序，那么容器平台就是运行事物的操作系统。容器平台为最终用户和 API 提供接口以运行和管理容器化应用程序。Nimbix 云中的 JARVICE 是 HPC 的事实上的容器原生平台，而 Kubernetes 正日益成为 Web 服务应用程序事实上的容器原生平台。JARVICE XE 实际上与 Kubernetes 交互，以在融合的 IT 基础架构上运行 HPC 应用程序。

3. Linux容器的替代方法：Singularity

虽然 Docker 样式的容器越来越普遍且通用，另一种格式 Singularity 在 HPC 中取得了一些进展。两者的主要区别在于架构理念。Docker 格式（和 Docker 运行时）旨在完全隔离地工作，为每个容器提供自己的网络和系统语境。它还允许容器化应用程序在其“监狱”和命名空间（见上文）中获得管理访问权限（也称为“root”）。容器平台必须确保应用程序得到适当的资源管理以避免安全问题，这已获得广泛理解。在 Singularity 学派中，容器实际上使用主机进行网络和互连，且不允许容器化应用程序获得 root 权限，即使是在其隔离的运行时环境中。设计理念是在已经在主机上安装所有内容，例如 MPI 库（一个关键的 HPC 组件）等的整体式传统 HPC 环境中更好地支持工作负载可移植性。

Docker 的理念是，主机只为容器化应用程序提供运行时环境设置。

因此，Docker 样式的容器化应用程序必须自带所有依赖项。虽然这会导致镜像略大，但它确实简化了在同一系统上同时运行不同代码的过程，并且更适合于多租户环境，例如 JARVICE。

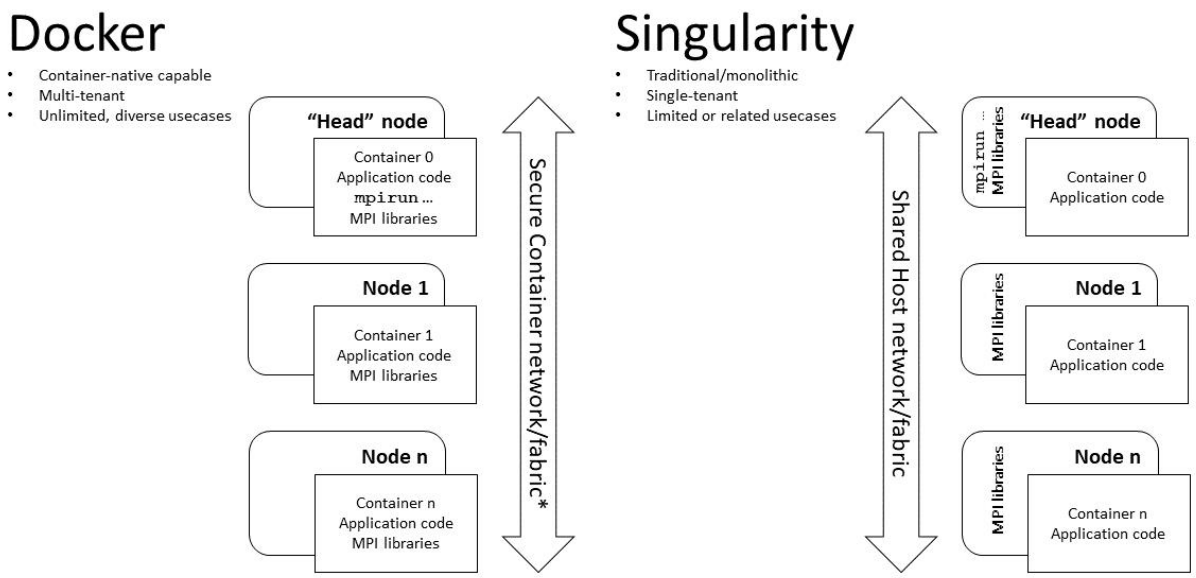

图1：容器化 HPC 应用对比

虽然 Singularity 正在传统 HPC 中取得进展，但它不太可能挑战通用应用程序的 Docker 式容器。因此，它可能与在融合 IT 基础设施上统一 HPC 和商品应用程序无关。

4. 容器原生与使用Docker的容器化应用程序对比

理想情况下，在容器中运行的应用程序是容器原生的。这通常意味着几件事，但一般归结为：

1. 最小化依赖——只打包容器特定用途所需的那些库和配置文件；事实上，越来越多的静态链接二进制文件（例如由 Go 语言生成的二进制文件）正在取代应用程序堆栈，进一步简化和使容器映像比以往任何时候都更精简。

2. 简化操作——由于容器旨在在容器平台上运行，因此无需打包可扩展的 HTTP(S) 服务器、防火墙软件等大型框架。容器平台通常提供此功能，只需通过标准化的服务端口将请求代理到容器化的应用程序中。横向扩展也由容器平台处理，进一步降低复杂性。在 Kubernetes 环境中，应用程序必须遵循简单的服务发现规则，由平台负责其余的。在 JARVICE 环境中，HPC 应用程序在运行时环境中自动配置，有利于无缝 MPI 和其他跨节点的并行分发方法。

虽然容器原生应用程序当然是一种容器化应用程序，但反之则不然。例如，可以（使用易于理解的方法）容器化用于分发和移动的传统应用程序，无论它们是开源还是商业位。

多年来，鉴于现有 HPC 应用程序的广度和复杂性以及无需修改即可将它们容器化的需要，Nimbix 开发了一种方法来实现这一点：

1. 使用应用程序的安装程序构建容器镜像。在 Docker 术语中，这意味着在 Docker 文件本身中以“静默”模式（没有用户界面，因为在构建容器镜像时没有用户输入的机会）运行安装脚本。多年的研究和发展带来高度优化的层和支持，即使是最复杂和最广泛的应用程序套件。再次， ISV 代码不是容器原生的，不能分解为微服务，并且不知道任何类型的容器运行时。

2.添加了额外的层来扩展容器功能，例如图形用户界面（例如基于 Web 的桌面或 shell）、第三方集成（例如用于跨不同供应商代码的“协同模拟”）和便利性（例如用户通常与其 HPC 代码结合运行的桌面应用程序，如文本编辑器等）。 Nimbix 在一个名为“image-common”的 GitHub 包中开源了容器的图形桌面环境。

3.工作流脚本，用于自动运行应用程序和管道许可服务器连接，自动配置并行规模参数等。这些脚本假设底层有一个 JARVICE 平台，但可以很容易地在本地模拟，以用于小规模的单元测试应用程序工作流。

4. 工作流自动化和“贸易包”的元数据——简单的声明文件被分层，这有助于 JARVICE 平台生成用户界面并向工程师和科学家展示高级工作流。这有助于“构建”命令和参数以在运行时在容器镜像内执行。

“贸易包”包括描述性信息、屏幕截图和可选的 EULA 语言，用于为 HyperHub 生成服务目录。此元数据是隔离的，不会以任何方式干扰非 JARVICE 平台——维护人员可以在没有 JARVICE 的情况下继续在通用 Docker 运行时上运行他们的代码，但也没有工作流自动化的最终用户好处。

实际上，容器化传统应用程序所需的只是自动化安装脚本、执行必要的安装后修复，并开发包装机制

（例如工作流脚本）来参数化和动态编辑配置以适应运行时的动态环境。一个主要的挑战是容器的短暂性——与工作站或服务器不同，除非文件存储在显式卷中，否则没有持久性。因此在某些情况下，需要进行大量修复。凭借这十年的大部分经验，Nimbix 既在许多流行的应用程序堆栈上实现了这一壮举，又为无数开发人员和 ISV 提供了以自助方式容器化自己代码的最佳实践的建议。JARVICE 平台甚至提供了一种称为 PushToCompute™的 CI/CD 管道机制，它可以进一步帮助在异构平台和架构上启用和部署传统应用程序。

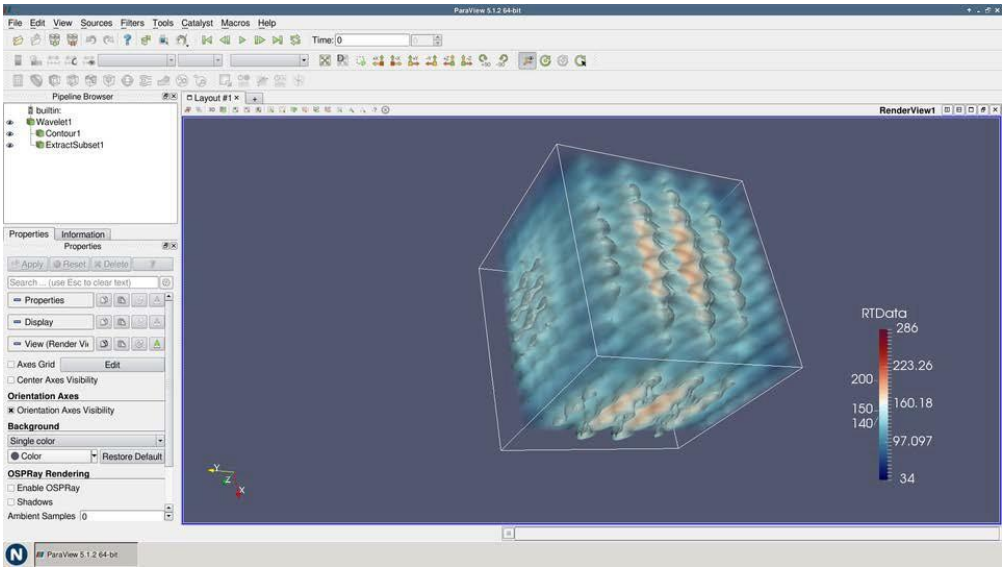

图2：在 JARVICE 平台上运行的示例容器化（非容器原生）应用程序

示例和参考资料

Nimbix 在 GitHub 上提供了容器化传统和容器原生工作流的各种示例。这些源可以用作为各种类型的 HPC 应用程序生成 Dockerfile 和元数据的模式。

此外，JARVICE 开发人员文档包括完整的参考。请注意，虽然所有这些都假设一个容器运行时环境可以设置和模拟动态 HPC 集群，但对使用 JARVICE 平台本身并没有严格要求。

5. Kubernetes上的HPC

以下部分将检查在 Kubernetes 上运行 HPC 工作流的选项。

JARVICE 与 Kubernetes 应用程序模式支持的对比

为了进行比较，下表说明了 Kubernetes 和支持 Nimbix 云的原始 JARVICE 平台之间对各种常见应用程序模式的支持级别。 作为参考，原始 JARVICE 平台早于 Kubernetes。

模式	示例	JARVICE 支持	Kubernetes 支持
面向服务的应用程序（SOA）	横向扩展 MVC（模型/视图/控制器）应用程序，例如：内容管理系统	基础：可以按预定规模运行单个或多个容器化镜像，但不提供自动负载平衡或 HA。	完整：可以作为多容器镜像运行并单独扩展服务，以及提供服务发现和负载平衡。
HPC：“令人尴尬的并行”或“完美并行”的应用程序	Monte Carlo 仿真	完整：容器环境设置自动跨越多个节点，在启动时确定规模；“主”节点可以设置和分片数据等。	部分：Kubernetes“部署”确实可以并行启动多个容器，但在启动时规模大小是“尽力而为的”，因为不可能实现“组调度”；此外，应用程序不可能在“主”容器上作为“从属”执行不同的功能，因此必须提前或手动执行分片和设置。这在架构上与大多数现有应用程序不兼容。
HPC：紧耦合并行求解器	计算流体动力学（CFD）	完整：准备运行 MPI 启动的求解器，无论是直接通过 mpirun 还是间接通过应用程序前端来设置数据和处理。 JARVICE 提供了一个完整的动态集群，包括结构设置、节点（并行容器）之间的 SSH 信任以及为 MPI 生成的机器文件等。	无：Kubernetes“部署”不适合这种机制，因为它不能保证不在启动时或在运行时扩展，不支持在所有并行资源可用之前将作业排入队列进行“组调度”，不自动选择和运行“主”容器的代码，不会自动为应用程序配置结构。
AI: 加速并行训练	分布式深度学习 (DDL)	完整：与其他并行求解器类似，支持 HPC 样式深度学习，在架构上是与分布式训练框架（如 Horovod）兼容。	无：出于与并行求解器相同的原因；当需要规模时，必须使用替代培训工作流程（假设框架支持）。

AI：加速实时分析	推理	完整：当使用 FPGA 和新颖的推理硬件等技术时，JARVICE 可以提供和托管预定规模的单节点或多节点服务。	完整：假设存在用于加速硬件的 Kubernetes 插件，可以像支持 SOA 一样支持这些堆栈（见上文）。

6. 基础Kubernete上的HPC

由于平台本身不提供足够的 HPC 支持（如上所述），这里有一些选项和解决方法。

单 Pod 求解器

如果需要多核/多线程而不是多节点，则可以将基于 MPI 的求解器配置为单个 Pod 中的单个容器，并绑定到单个节点。这消除了对结构设置和“主/从”类型配置的需要。求解器可以简单地启动以在配置的 CPU 内核和线程上使用共享内存互连。根据底层节点容量，这可能足以满足某些形式的 HPC 求解，但显然会将规模限制为适合单个节点的。

外部控制

对于多个 Pod，一种可能的解决方法是在“待机模式”下启动所有容器——对于大多数应用程序，这意味着简单地执行初始化或显式启动 SSH 服务器。然后，外部进程可以发现给定部署实际可用的 pod、容器地址是什么，然后继续执行构建机器文件并启动基于 MPI 的求解器来分发工作。明显的缺点就是其复杂性，因为这需要拆开应用程序或运行不同的阶段，以及明确管理单个容器。更重要的是，它仍然不能保证启动规模，因 Kubernetes 调度器不支持组调度，所以比起将一组 pod 排队直到全部容量可用时，它会简单地绑定任何可以绑定的东西，并持续这样做，直到所有 pod 都被绑定。

外部控制也可以手动执行，用户可以查看可用的 Pod，并在绑定所需的比例后继续进行设置。

无论如何，仍然需要在容器之间建立 SSH 信任——这可以在构建时通过通用信任来完成（简单但不安全），或者在启动后明确（更复杂且难以在应用程序层自动化）。

尴尬的并行求解器

虽然扩展有些简单，但尴尬（或“完美”）的并行求解器仍然需要设置，并且可能需要在启动之前对数据进行分片。如果平台不能协调一组有保证的容器并在开始工作之前自动将控制权移交给设置过程，即使求解器架构更适合于无状态复制样式的系统，也存在类似的问题。

如果算法永不需要协调（即使在设置时也不需要），这可能不是任何类型的 HPC 求解器，但它可以很好地在标准 Kubernetes 平台上运行。

7. 使用JARVICE XE在 Kubernetes上运行HPC

JARVICE XE 通过两个主要改进弥补了在 Kubernetes 上运行 HPC 代码的差距：

7.1 两级 HPC 调度器

调度器提供 2 个级别，一个将传统的 HPC 作业请求转换为一组 Kubernetes Pod，一个将 Pod 绑定到节点的组调度器，如果请求的规模不可用，则将整个作业排队。

在 Kubernetes 术语中，绑定一个 pod 是指将一个 pod（及其封装的容器）放置在一个工作节点上进行处理； 这相当于作业调度程序选择一个节点来运行工作，然后依次运行。

此外，组调度器提供以下重要功能：

1.“最适合”，异构部署的理想选择——JARVICE XE pod 调度器将始终尝试将 pod 放置在总资源最少的节点上，包括加速器等。这与考虑负载不同。假设没有尝试真正的超额订阅，会导致更好的资源利用。

例如，在其中一部分节点具有 GPU 的集群中，除非所有仅 CPU 节点都在使用中，否则在这些节点上调度仅 CPU 工作是没有意义的。如果只考虑当前负载，这很容易导致具有更稀缺和更新颖设备的节点上的周期浪费。该调度器吸取了 Nimbix 云（一个多租户异构部署）上资源管理的经验教训。

2. 可配置的资源权重——当涉及到更有价值的东西时，不同的服务供应商可能有不同的经济学——例如大内存节点与 GPU 节点的对比。JARVICE XE pod 调度器可以通过配置，适当地权衡这两点。

3. 建议限制支持——在多租户甚至多用户环境中，有时需要限制某些用户和团队可以使用的资源数量和类型。JARVICE XE 支持自助服务和管理员强加的限制，即使在调度作业时有更多资源可用。这与 Kubernetes 已经支持的命名空间限制不同，因为在这种情况下，尝试超过限制会导致对象创建失败。在 JARVICE XE 情况下，当达到限制时，作业会排队，这更符合最终用户的期望。在调度批处理 HPC 作业时，这仍然适用于“排队并遗忘”的操作，用户在调度工作时无需担心资源管理。

4.租户隔离——在多租户环境中，出于安全或合规性防止节点运行属于多个租户的工作。即使在单租户环境中，用户团队之间仍然存在监管限制。标记节点并定义针对它们的资源类型是实现这一目标的明显方法，但这是非常静态的，在不限制确切机器的情况下可能无法产生最佳利用率。相反，JARVICE XE 可以动态地确保没有两个租户或团队同时共享相同的节点进行工作。

调度器的上层部分还将资源集合作为“机器类型”呈现给用户，这是最终用户在运行工作时选择规模和能力更自然的方式。机器类型当然可以请求非常细粒度的资源，但这种复杂性可以从最终用户那里抽象出来，例如，他们只是决定使用 MPI 在 32/16 个核心节点(共 512 个核上运行一个作业。JARVICE XE 在

将此请求移交给 pod 调度程序进行绑定之前，将其转换为适当的 pod 副本计数和资源请求。整个机制对最终用户来说是不透明的，使系统操作更加简单。

调度程序的下层部分（也称为 pod 调度程序）实际上可以与默认的 Kubernetes pod 调度程序并行运行，但如果竞争相同的资源，当然可能存在竞争条件。Kubernetes 在绑定 Pod 时不使用全局临界区，因此如果默认和 JARVICE XE Pod 调度程序都试图将 Pod 绑定到相同的节点，则可能会过度订阅资源（导致 Pod 驱逐）。最佳实践是使用标签和污点来确保 JARVICE 独占专门用于 HPC 的硬件上的调度。JARVICE pod 调度器确实支持多个命名空间，因此实际上可以在同一个集群调度工作上同时部署多个 JARVICE XE，而不会出现竞争条件的风险。

调度程序可通过 API 或点击式 Web 门户访问。**如上所述，JARVICE XE 使用来自 HyperHub™目录中应用程序的元数据为最终用户定义工作流，而不是要求用户编写 PBS 或 Slurm 脚本来启动工作。**

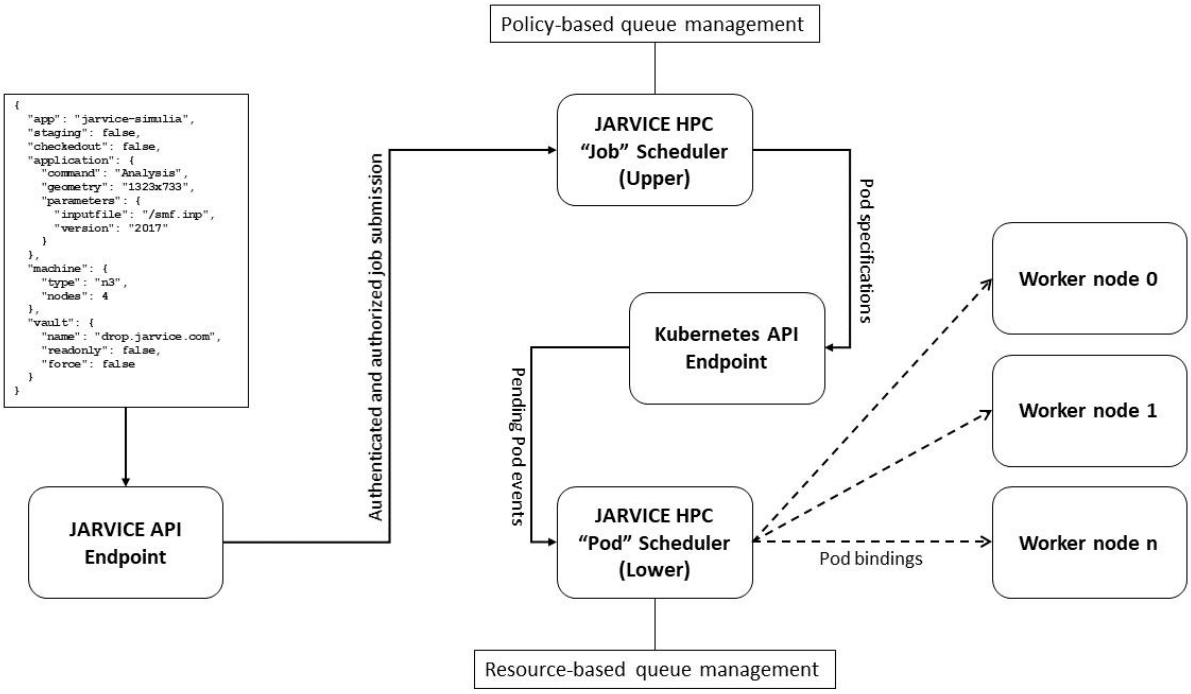

图 3：Kubernetes 上的 JARVICE HPC 调度程序架构：示例 64 核批处理模拟的提交流程

7.2 HPC 运行时环境

JARVICE XE 为 Kubernetes 带来的第二个主要改进是 HPC 运行时环境，它在作业启动时动态创建。该环境执行以下功能：

1. 根据来自调度程序的工作流请求的参数配置批处理运行或交互界面

2. 确保工作流的有限完成，无论求解器成功还是失败（默认情况下，使用 Kubernetes “Job” API，失败的求解器将自动重启）；由于故障是 HPC 作业处理的正常部分，因此重要的是要捕获这些故障并将其视为自行完成，以便用户有机会查看工作流参数并通过适当的调整重试。使用相同参数自动重新启动将导致最终用户可能永远不会检测到无休止的故障循环。

3. 建立清晰的“主/从”拓扑，采用“头节点”设计，启动求解器将工作分散到工作节点（从）。这意味着传统的 HPC 代码可以不加修改地在 JARVICE 上运行，而不是在执行任何工作之前实现复杂的主选举逻辑。

4. 自动生成可以直接传入 mpirun 命令行的机器文件，例如，通过建立的 SSH 信任，结构配置等确保从属准备好接收请求。每个作业在自己的动态集群环境中运行，自然地向传统 HPC 软件展示自己。

5. 支持任何类型的 Kubernetes PersistentVolumeClaims，包括 ReadWriteOnce，并在作业中的所有节点之间动态共享它们。这意味着用户可以使用块存储卷启动多节点作业，并且求解器将自动从所有节点获取共享文件系统访问权限，而不会因存储争用而导致数据损坏或无限排队。如果需要，JARVICE XE 还直接支持 NFS 和 CephFS 共享文件系统，无需管理下面的 Kubernetes 卷。

此外，JARVICE XE 的运行时环境可以通过将主机级别的挂入点定义为机器定义的一部分来附加 Kubernetes 原生不支持的存储接口。这支持并行存储系统，例如 WekaIO2。

6. 支持基于用户登录的网络安装主目录，包括用于从登录名和网络安装主目录中的文件自动导出用户身份 ID（UID 和 GID）；这消除了在容器中进行复杂配置以访问 ActiveDirectory 等的需要。

7. 在每个作业的基础上自动管理机密、配置、服务和入口。例如，如果用户运行交互式作业，JARVICE XE 将自动为其配置入口，以便最终用户可以通过 Web 浏览器远程访问它。

如上所述，Nimbix 提供基于 Web 的桌面环境作为任何容器的开源层，因此这是为传统 HPC 应用程序提供图形用户界面的一种非常有效的方式。

JARVICE XE 使用安全令牌来确保用户无法访问彼此的桌面，除非他们明确共享链接或冒充对方（JARVICE XE 的可选功能）。

8. 高规模与高吞吐量作业调度对比

作业调度程序要求经常混淆用例，了解差异很重要。

JARVICE XE 在平台层面提供了一个大规模的作业调度器，这意味着用户可以立刻提交在多台机器上运行的作业，但可能需要几秒钟到几分钟（取决于工作流复杂性、容器缓存状态等）才能开始处理作业。平台级调度程序本身适用于运行需要有限但大量计算时间来执行的大型作业。这再次受到 Nimbix 云在过去十年的大部分时间里提供按需 HPC 的启发。

大多数工程/模拟或分布式深度学习作业需要几分钟、几小时或几天才能运行，即使是大规模运行也是如此。

对于高吞吐量调度，最好不要依赖平台本身来运行单个作业。这是将单个大型作业调度为动态集群，然后在其中嵌入作业调度程序的合适之处。 Nimbix 在 HyperHub 上为“HPC 测试环境”（GitHub 上可用的容器源）提供了一个简单的示例模式，它实际上动态部署了一个 Slurm 集群，一旦启动就准备好调度工作。虽然基础镜像通常用于测试或向最终用户提供按需学术风格的集群，但也非常适合某些用例，例如许多倾向于在测序仪数据上运行阵列作业的生物信息学代码等。一旦作业开始并且嵌入式调度器已完全配置，动态集群内的作业提交延迟与物理静态系统上没有什么不同。之后，运行许多短期作业变得高效，而不会产生在基础设施上配置容器和其他对象的平台开销。明显的缺点是在按需环境中，比起作业结束并在大规模平台调度模型中自动放弃资源，用户在动态集群的生命周期内消耗所有资源。用户或应用程序本身必须实际终止嵌入式调度程序模型中的动态集群，以便将资源返回给系统。

9. 与Kubernetes和JARVICE XE融合的IT基础架构

JARVICE XE 通常作为 Kubernetes 集群上的一组服务运行，通过 Helm chart 部署。对于集群管理员来说，像系统上的任何其他框架一样，应用更新和管理 JARVICE XE 本身是微不足道的。JARVICE XE 处理上面的整个用户界面，包括用户 HPC 应用程序、身份、身份验证、授权、作业控制、会计和审计。它支持多个租户（或团队），而无需部署单独的实例。简而言之，它使在大型 Kubernetes 部署中启用 HPC 与部署任何其他类型的应用程序一样简单。

融合部署通过以下方式使组织受益：

1. 减少专门的培训——管理 IT 基础设施的同一团队现在可以通过最少的培训来管理 HPC；原因在于 JARVICE XE 提供来自 HyperHub 的策划工作流，包括 HPC 应用程序工程，因为工作流已经准备好运行。

2. 提高利用率——由于最佳实践是在同一集群上分离 HPC 和商品工作负载，不需要冗余的控制平面、存储或网络；这提高了集群的规模经济，并且可以更轻松地插入专用硬件，而无需“步骤功能”和复杂的管理界面。

3. 工作负载之间的数据共享——一个常见的用例是将 SOA 应用程序与同一基础架构上的 HPC 应用程序相结合。例如，完整的深度学习管道可以具有基于 SOA 的推理服务，该服务从基于 HPC 的 DDL 训练工作流按需或连续生成的同一存储中读取训练模型。SOA 可以在有新数据或用于强化目的时通过 Web 服务 API 触发训练，而 JARVICE 可确保 DDL 训练架构正确运行。

4. 迁移和恢复简化——当需要移动、复制或恢复集群时，集群管理员可以依靠相同的工具和技术来处理商品与 HPC 工作负载。

10. 使用JARVICE XE的多云和混合云

由于 JARVICE XE 提供了一个处理 API 和一个通用的、同步的服务目录（来自 HyperHub），工作负载移动性是无缝的。相同的工作负载在任意提供 JARVICE XE 的集群上运行，包括保持兼容性的 Nimbix 云。

即使在不同的云基础设施提供不同类型资源的情况下，JARVICE XE 应用程序仍然保持兼容性，针对机器类型和资源请求的明确定义的模式确保了可移植性。

JARVICE XE 目前不管理跨多个云的数据移动和迁移，但支持底层采用的任何机制来实现这一点。在用户从未在多个云上运行工作的情况下（但不同的用户团队会），这通常不是问题，因为数据将始终驻留在计算中。

JARVICE XE 提供了一些附加功能来简化多云部署：

1. Kubernetes 集群上的 Helm Chart 部署使得在多个基础设施上管理 JARVICE XE 变得无缝，包括部署更新和更改部署参数。

2. 对 Amazon EKS 和 Google GKE 等公共云上管理 Kubernetes 端点的脚本化支持；这是比 Helm 图表本身更高级别的抽象，在几分钟内支持 JARVICE XE 的单个命令部署。

3. 跨集群的公私应用同步；不仅可以同步上游的 HyperHub 应用程序，还可以选择私有和自定义应用程序，进一步提高工作负载的移动性。

4. 在有联盟和在无联盟的情况下单点登录；JARVICE XE 支持 Active Directory 和 SAML2，允许用户身份在多个部署中保持一致。

5. **“单个管理界面”**——相同的 JARVICE XE 界面操作任何集群

a. 2018 年第四季度可用：基于 URL 的联盟 - 用户通过定位其用户门户入口 URL 来访问他们希望在其上运行工作的集群或区域。

b. 2020 年第二季度可用：来自同一门户 URL 的管理员和团队控制的联盟——用户从单个 Web 门户 URL 启动和管理多个集群上的作业。

c. 2020 年第二季度可用：根据站点配置的策略和规则，将作业从一个集群突增到另一个集群；在

JARVICE 多云模型中，机器类型固定到集群，因此当最初请求的机器繁忙时，突增策略定义了 可以利用哪些机器（前提是满足所有其他约束）。

由于多云部署通常是一种安全性和合规性技术，**因此 JARVICE XE 可用于根据登录凭据和组员身份限制租户、团队和个人用户对计算位置的访问，包括对应用程序（例如强制出口控制）、数据集甚至硬件的访问。**

在所有情况下，底层管理，无论是单集群还是多集群，对应用程序都是完全透明的。真正为 HyperHub 应用程序容器启用了“一次发布，随处运行”的模型。

11. 结论

- Kubernetes 正在成为标准的企业容器部署平台，但不支持 HPC
- Nimbix 云中的 JARVICE 是领先的容器化应用按需 HPC 平台
- JARVICE XE 将 HPC 引入 Kubernetes 管理的基础架构，包括丰富（且不断扩展）的可立即运行的开源和商业工作流目录
- JARVICE XE 为传统应用程序提供运行环境，无需修改即可运行，无需转换为容器原生架构；这在部署源代码不可用或架构配置不足以发生巨大变化的商业堆栈时尤其有价值
- JARVICE XE 支持私有、混合、公共和多云部署
- JARVICE XE 使企业 IT 部门可以轻松地将 HPC 添加到他们的服务组合中，而无需额外的实践、工具或专业技能

虹科电子科技有限公司

www.hongcloudtech.com
hongcloudtech@hkaco.com

广州市黄埔区神舟路18号润慧科技园C栋6层

T (+86) 400-999-3848
M (+86)15528663362

联系我们　微信公众号　行业交流群　hongcloudtech.com