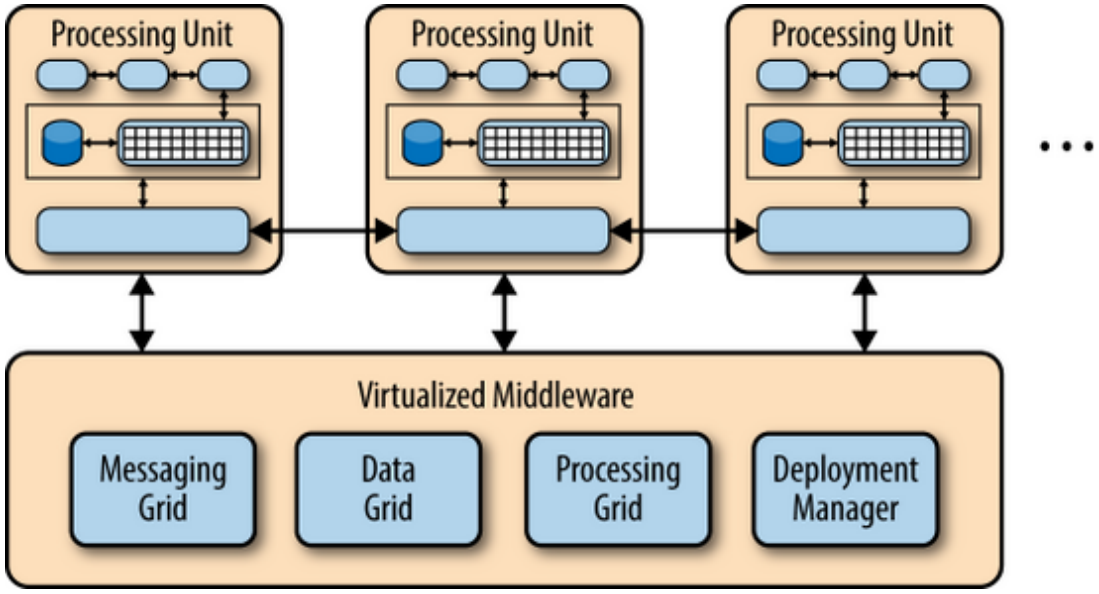
基于空间的架构浅析与实践

大部分Web应用由Web服务器、应用服务器和数据库服务器的分层架构设计，用户的请求依次经过各个服务器。当用户访问量增加到一定程度后，各个服务器均可能出现性能瓶颈，扩展某一层的服务会将瓶颈转移到下一层，系统瓶颈在不同情况下可能出现在不同层次，系统自动按需扩展比较困难且水平扩展服务层比较复杂，代价也更高，虽然可以通过缓存或者其他辅助扩展工具来解决这些问题，但是扩展出可承受极端负载的应用会比较难。面对大量不可预测的并发用户量的访问负载，从架构上进行优化是更好的方式，基于空间的架构模式被设计用来解决扩展性和并发的问题。

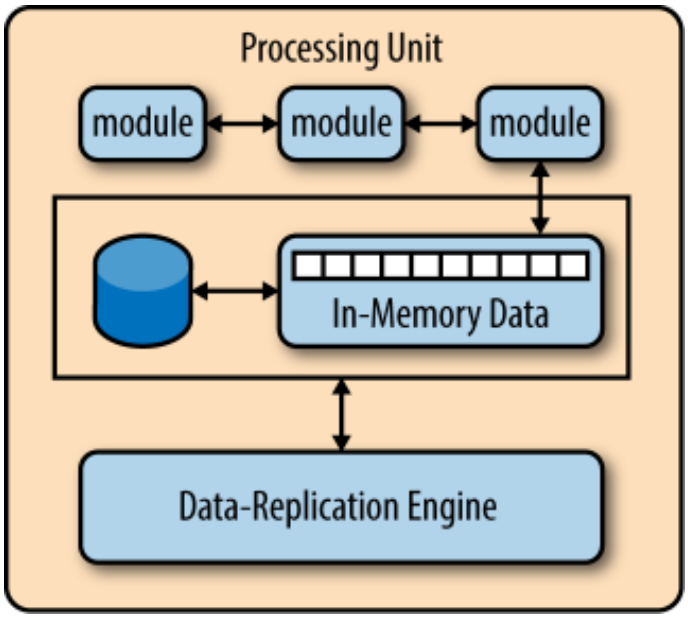
1.基本框架与概念

> 基于空间的架构模型（The space-based pattern，也称为云架构模型），旨在减少限制应用伸缩的因素。The Space-Based Pattern来源于分布式共享内存中的元组空间（Tuple Space）的概念，其通过数据网格来替代数据库来实现高伸缩性，应用需要的数据保存并复制给所有运行的进程，并且这些进程动态的随着请求数量的增减而启动或结束，通过这种方式可以避免中心数据库的瓶颈问题，同时提供近乎无限的扩展能力

这种架构模式有两个主要组件：处理单元（processing unit）和虚拟化中间件（virtualized middleware），如下图所示：



- **1）处理单元**
  - 应用组件或者部分应用组件，包括Web、后端业务逻辑
  - 内存中的数据网格
  - 异步持久化转移模块（可选）
  - 复制引擎，提供给虚拟中间件使用来同步各单元之间的数据变化

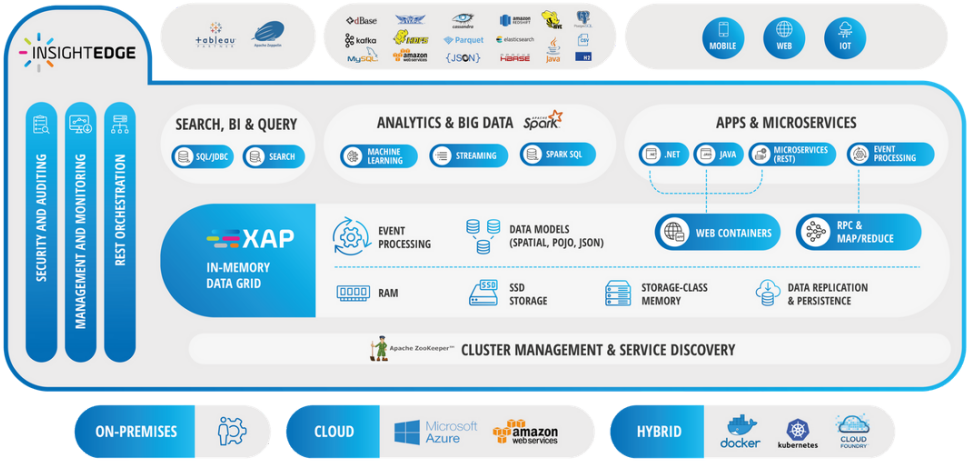


- **2）虚拟化中间件**，控制数据同步和请求处理
  - 消息网格，管理输入的请求和会话信息，当一个请求到达虚拟化中间件后，消息网格根据处理单元的状态进行请求的分配，支持多种算法如轮询、优先级等
  - 数据网格，在数据变化时和处理原因的数据复制引擎交互，进行数据的同步。消息网格将请求指派给处理单元后，每个处理单元内存中的数据是一致的，数据的复制基本上在较短时间内异步完成的
  - 处理网格（可选组件），协调和编排各个处理组件间的分布式请求
  - 配置管理器，决定应用可伸缩性的重要组件，持续监控处理单元的响应事件和用户负载，动态地开启或关闭处理单元

基于空间的架构比较复杂，不太适合使用传统的大规模关系型数据库应用，常用于负载不太稳定的Web应用，如社交网站等，通常在架构中会建立数据仓库来初始化内存数据，并异步地将变化的数据持久化。这种架构还被称为云架构，处理组件和虚拟化中间件可以通过部署在云服务或者PaaS上来实现扩展性，当然也可以部署在本地服务器上。目前很多产品实现了该框架，如数据管理平台Apache Geode、GigaSpaces等

2.GiGaSpaces浅析

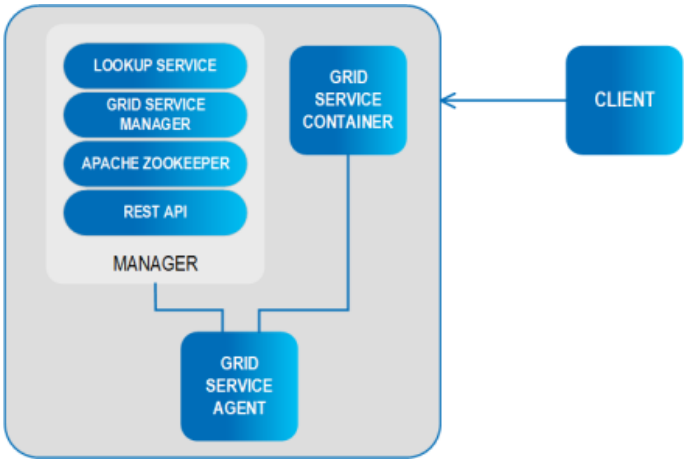
GigaSpaces为采用基于空间的架构的应用平台，其提供了分布式应用、数据缓存、消息传递、内存数据网格及分布式事务等功能，具有类似于OSGI的部署模型，并同时独立部署，私有云KubeGrid及Amazon EC2、Microsoft Azure等云平台等多种灵活方式，支持弹性扩展等高级特性。其系统架构图如下所示：



根据前面章节中基于空间架构的概念，下面介绍GigaSpace中核心层次的组件与基本流程

- 1). 服务网格层（Service Grid）

在数据网格中基本的部署单元是处理单元（Processing Unit，以容器的形式启动），业务程序打包后以处理单元的形式部署在GigaSpaces运行时环境中，这个运行时环境称为Service Grid，其负责处理单元的配置管理、与资源基础层交互获取物理资源来启动实例并且启动业务的管理，其架构如下所示：



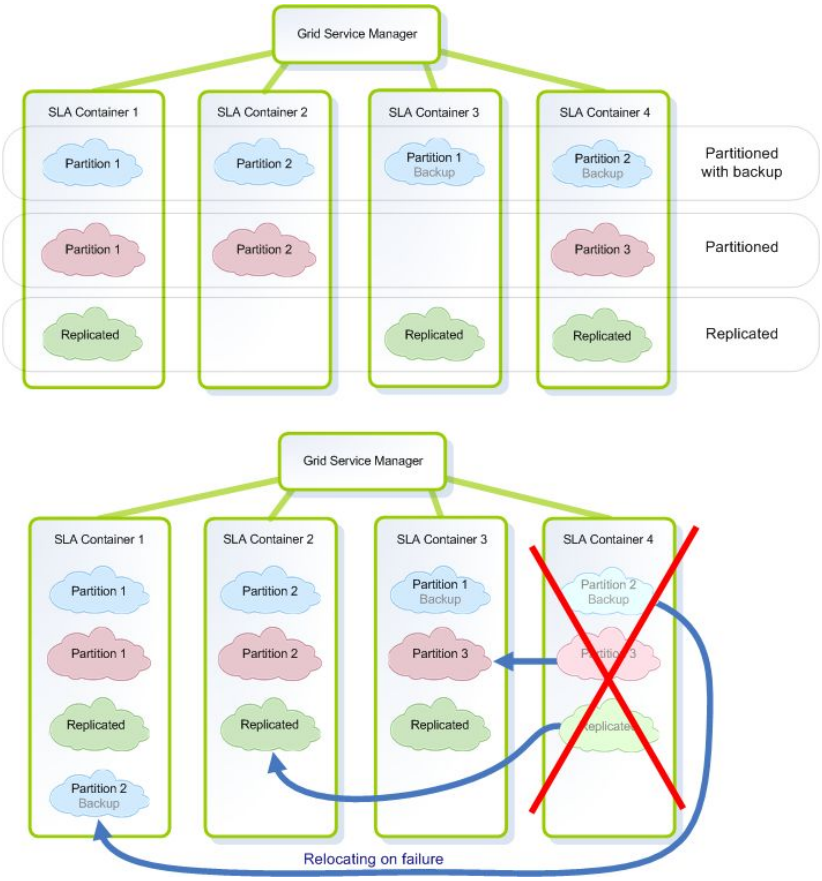
用户通过GigaSpace提供的部署工具（UI、CLI及API）等将Processing Unit提交给GSM Grid Service Manager（GSM），GSM进行PU的部署和生命周期管理，分析部署需要的资源及部署PU使用的容器，并将业务代码上传到Grid Service Container（GSC）中。为了实现PU的管理及调度，GigaSpace系统使用Lookup Service(LUS)来实现，GSM和GSC启动后注册到LUS中。最后Grid Service Agent负责启动这些组件

- Grid Service Manager(GSM)，处理单元的部署和生命周期管理，实时监控其运行状况，当其fails时根据策略进行恢复或者异常处理等，在系统中通常会启动两个GSM实例来保证高可用
- Grid Service Container(GSC), 提供处理单元实例运行的隔离Runtime，并将运行状态汇报给GSM，在GSC中可以根据启动物理主机的内存和CPU等情况启动同时启动多个PU，隔离技术可以使用Java中Class Loader或者.NET中的AppDomains
- Lookup Service(LUS)，用于服务间的组件发现，组件启动后注册到LUS中，LUS通过租约来保证组件的可用性，组件注册后周期性的更新租约的时间。LUS服务的可以通过multicast(默认)或者unicast机制来发现组件，组件在运行时从LUS中查询需要到的服务，例如GSM从LUS中查询可用的GSC。
- Grid Service Agent(GSA)，以上系统进程的启动及管理，通常一个物理机上启动一个GSA

还有其他组件如Apache Load Balancer Agent，用于部署Web应用，Transaction Manager(TXM)，实现事务管理等，用户根据需要进行选择性使用。

- 2). 内存网格层（In-Memory Data Grid Layer）

整个系统运行的核心，为基于空间的实例提供统一的数据存储层，其启动和使用方式与服务组件类似，支持嵌入或者独立组件方式。内存网格的组件启动后将数据加载到内存中，通常数据量比较大，数据以分区及副本的形式维护在内存网格组件中，在数据变化时进行内存中数据的更新并保证内存网格各组件中的数据一致。其架构如下图所示：



从运行角度来看，内存网格集群由运行在物理集群上的实例组成，运行的实例由GSM来管理，数据切分后维护在启动的组件中，这些组件之间通过网络进行交互形成数据的统一视图，客户端与任何组件交互均可以获取到网格中的所有数据。GigaSpace数据网格支持不同种类的缓存，如Map、Java POJO及提供JDBC API来进行缓存数据的查询。

服务网格和内存网格是GigaSpace的核心功能，之外还有消息网格提供事件驱动框架来支持组件进行的消息传递；分布式处理框架，提供分布式程序的编程接口来支持用户自定义程序的执行，方式包括远程执行（Executors及Remoting），来简化分布式程序的开发等；事务处理，支持本地事务、分布式事务；监控与管理等。

参考链接

> https://www.oreilly.com/library/view/software-architecture-patterns
> https://www.tuicool.com/articles/M73URjr
> https://docs.gigaspaces.com/latest/overview/space-based-architecture.html
> https://yq.aliyun.com/articles/1709
> http://blogxin.cn/2017/05/14/Space-Based-Architecture/