企业级分布式系统架构

1352961 秦博

分布式系统

分布式系统最大的特点是可扩展性，它能够适应需求变化而扩展。企业级应用需求经常随时间而不断变化，这也对企业级应用平台提出了很高的要求。企业级应用平台必须要能适应需求的变化，即具有可扩展性。分布式系统有良好的可扩展性，可以通过增加服务器数量来增强分布式系统整体的处理能力，以应对企业的业务增长带来的计算需求。

分布式系统由独立的服务器通过网络松散耦合组成的。每个服务器都是一台独立的 PC 机，服务器之间通过内部网络连接，内部网络速度一般比较快。分布式系统在设计上尽可能减少节点间通讯。

此外，因为网络传输瓶颈，**单个节点的性能高低对分布式系统整体性能影响不大。**比如，对分布式应用来说，采用不同编程语言开发带来的单个应用服务的性能差异，跟网络开销比起来都可以忽略不计。

分布式系统对服务器硬件要求很低。对服务器硬件可靠性不做要求，允许服务器硬件发生故障，硬件的故障由软件来容错。所以分布式系统的高可靠性是由软件来保证。

分布式系统强调横向可扩展性。横向可扩展性（Scale Out）是指通过增加服务器数量来提升集群整体性能。纵向可扩展性（Scale Up）是指提升每台服务器性能进而提升集群整体性能。纵向可扩展性的上限非常明显，单台服务器的性能不可能无限提升，而且跟服务器性能相比，网络开销才是分布式系统最大的瓶颈。横向可扩展性的上限空间比较大，集群总能很方便地增加服务器。

数据存储

通常，我们会通过两种手段来扩展我们的数据服务：

(1)数据分区：就是把数据分块放在不同的服务器上(如：uid % 16，一致性哈希等)。

(2)数据镜像：让所有的服务器都有相同的数据，提供相当的服务。

对于第一种情况，我们无法解决数据丢失的问题，单台服务器出问题时，会有部分数据丢失。所以，数据服务的高可用性只能通过第二种方法来完成——数据的冗余存储(一般工业界认为比较安全的备份数应该是 3 份，如：Hadoop 和 Dynamo)。

同时，我们还要考虑性能的因素，如果不考虑性能的话，事务得到保证并不困难，系统慢一点就行了。除了考虑性能外，我们还要考虑可用性，也就是说，一台机器没了，数据不丢失，服务可由别的机器继续提供。于是，我们需要重点考虑下面的这么几个情况：

1)容灾：数据不丢

2)数据的一致性：事务处理

3)性能：吞吐量，响应时间

一个数据存储系统不可能同时满足上述三个特性，只能同时满足其两个特性，也就是： CA、CP、AP。可以这么说，当前所有的数据存储解决方案，都可以归类的上述三种类型。

【CA】 满足数据的一致性和高可用性，但没有可扩展性，如传统的关系型数据，基本上满足是这个解决方案，如 ORACLE , MYSQL 的单节点，满足数据的一致性和高可用性。

【CP】 满足数据的一致性和分区性，如 Oracle RAC ，Sybase 集群。虽然 Oracle RAC 具备一点的扩展性，但当节点达到一定数目时，性能（也即可用性）就会下降很快，并且节点之间的网络开销很在在，需要实时同步各节点之间的数据。

【AP】 在性能和可扩展性方面表现不错，但在数据一致性方面会用牺牲，各节点的之间数据同步没有哪么快，但能保存数据的最终一致性。当前热炒的 NOSQL 大多类是典型的 AP 类型数据库。

综合上述，架构师不要企图设计一套同是满足 CAP 三方面的数据库。只能在根据业务场景，对数据存储要求有所折衷。

分布式缓存与 NoSQL

NoSQL 又称为 Not Only Sql,主要是指非关系型、分布式、支持水平扩展的数据库设计模式.NoSQL 放弃了传统关系型数据库严格的事务一致性和范式约束,采用弱一致性模型.相对于 NoSQL 系统,传统数据库难以满足云环境下应用数据的存储需求,具体体现在以下 3 个方面:

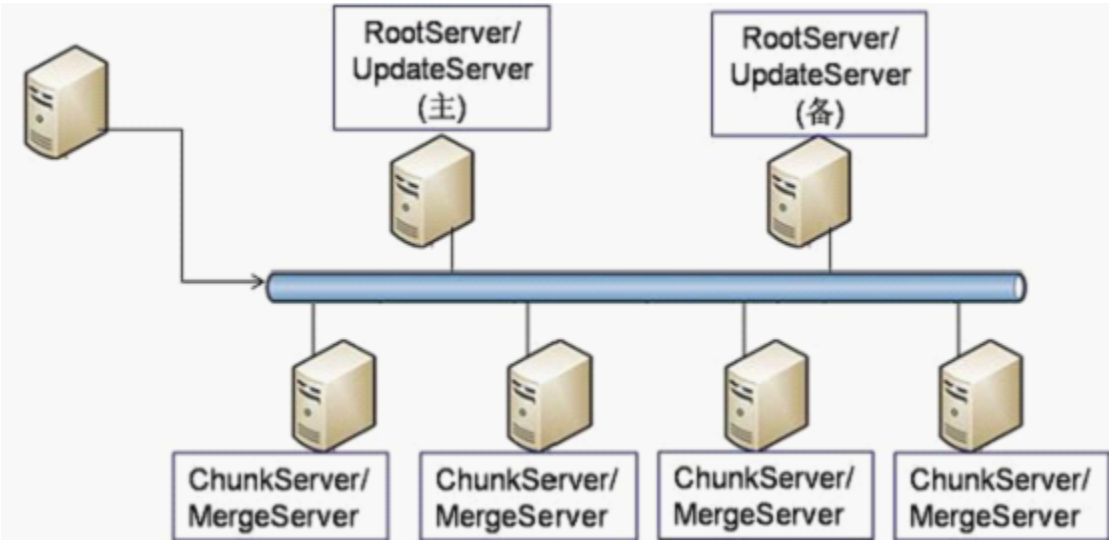
1) 根据 CAP 理论,一致性(consistency)、可用性(availability)和分区容错(partition tolerance)这 3 个要素最多同时满足两个,不可能三者兼顾.对云平台中部署的大量 Web 应用而言,数据可用性与分区容错的优先级通常更高,所以一般会选择适当放松一致性约束.传统数据库的事务一致性需求制约了其横向伸缩与高可用技术的实现;

2) 传统数据库难以适应新的数据存储访问模式.Web 2.0 站点以及云平台中存在大量半结构化数据,如用户 Session 数据、时间敏感的事务型数据、计算密集型任务数据等,这些状态数据更适合以 Key/Value 形式存储,不需要 RDBMS 提供的复杂的查询与管理功能;

3) NoSQL 提供低延时的读写速度,支持水平扩展,这些特性对拥有海量数据访问请求的云平台而言是至关重要的.传统关系型数据无法提供同样的性能,而内存数据库容量有限且不具备扩展能力.分布式缓存作为 NoSQL 的一种重要实现形式,可为云平台提供高可用的状态存储与可伸缩的应用加速服务,与其他 NoSQL 系统间并无清晰的界限.平台中应用访问与系统故障均具有不可预知性,为了更好地应对这些挑战,应用软件在架构时通常采用无状态设计,大量状态信息不再由组件、容器或平台来管理,而是直接交 付给后端的分布式缓存服务或 NoSQL 系统.

负载均衡

在分布式系统中存在着著名的“短板理论”，一个集群如果出现了负载不均衡问题，那么负载最大的机器往往将成为影响系统整体表现的瓶颈和短板。为了避免这种情况的发生，需要动态负载均衡机制，以达到实时的最大化资源利用率，从而提升系统整体的吞吐。结合 OceanBase 的实际应用分享一个去年淘宝双十一前期的准备工作中遇到负载均衡相关案例。



OceanBase 是一个具有自治功能的分布式存储系统，由中心节点 RootServer、静态数据节点 ChunkServer、动态数据节点 UpdateServer 以及数据合并节点 MergeServer 四个 Server 构成

- Tablet：分片数据，最基本的存储单元，一般会存储多份，一个 Table 由多个 tablet 构成；
- RootServer：负责集群机器的管理、Tablet 定位、数据负载均衡、Schema 等元数据管理等。
- UpdateServer：负责存储动态更新数据，存储介质为内存和 SSD，对外提供写服务；
- ChunkServer：负责存储静态 Tablet 数据，存储介质为普通磁盘或者 SSD。
- MergeServer：负责对查询中涉及多个 Tablet 数据进行合并，对外提供读服务；

在一个集群中，Tablet 的多个副本分别存储在不同的 ChunkServer，每个 ChunkServer 负责一部分 Tablet 分片数据，MergeServer 和 ChunkServer 一般会一起部署。

消息队列

kafka 是 LinkedIn 开发并开源的一个分布式 MQ 系统。在它的主页描述 kafka 为一个高吞吐量的分布式（能将消息分散到不同的节点上）MQ。

企业集成的基本特点是把企业中现存的本不相干的各种应用进行集成。例如：

一个企业可能想把财务系统和仓管系统进行集成，减少部门间结算和流通的成本和时间，并能更好的支持上层决策。但这两个系统是由不同的厂家做的，不能修改。另外企业集成是一个持续渐进的过程，需求变化非常频繁。这对 MQ 系统的要求是要非常灵活，可定制性要求高。所以常见的 MQ 系统通常都可以通过复杂的 xml 配置或插件开发进行定制以适应不同企业的业务流程的需要。他们大多数都能通过配置不同程度的支持 EIP 中定义一些模式。但设计目标并没有很重视扩展性和性能，因为通常企业级应用的数据流和规模都不会非常大。即使有的比较大，使用高配置的服务器或做一个简单几个节点的集群就可以满足了。

大规模的 service 是指面向公众的向 facebook，google，linkedin 和 taobao 这样级别或有可能成长到这个级别的应用。相对企业集成来讲，这些应用的业务流程相对比较稳定。子系统间集成的业务复杂度也相对较低，因为子系统通常也是经过精心选择和设计的并能做一定的调整。所以对 MQ 系统的可定制性及定制的复杂性要求并不高。但由于数据量会非常巨大，不是几台 Server 能满足的，可能需要几十甚至几百台，且对性能要求较高以降低成本，所以 MQ 系统需要有很好的扩展性。

kafka 正是一个满足 SaaS 要求的 MQ 系统，它通过降低 MQ 系统的复杂度来提高性能和扩展性。

- Producer （P）：就是往 kafka 发消息的客户端
- Consumer （C）：从 kafka 取消息的客户端
- Topic （T）：可以理解为一个队列
- Consumer Group （CG）：这是 kafka 用来实现一个 topic 消息的广播（发给所有的 consumer）和单播（发给任意一个 consumer）的手段。一个 topic 可以有多个 CG。topic 的消息会复制（不是真的复制，是概念上的）到所有的 CG，但每个 CG 只会把消息发给该 CG 中的一个 consumer。如果需要实现广播，只要每个 consumer 有一个独立的 CG 就可以了。要实现单播只要所有的 consumer 在同一个 CG。用 CG 还可以将 consumer 进行自由的分组而不需要多次发送消息到不同的 topic。
- Broker （B）：一台 kafka 服务器就是一个 broker。一个集群由多个 broker 组成。一个 broker 可以容纳多个 topic。

- Partition（P）：为了实现扩展性，一个非常大的topic可以分布到多个broker（即服务器）上。kafka 只保证按一个 partition 中的顺序将消息发给 consumer，不保证一个 topic 的整体（多个 partition 间）的顺序。

可靠性（一致性）

MQ 要实现从 producer 到 consumer 之间的可靠的消息传送和分发。传统的 MQ 系统通常都是通过 broker 和 consumer 间的确认（ack）机制实现的，并在 broker 保存消息分发的状态。kafka 的做法是由 consumer 自己保存状态，也不要任何确认。不管 consumer 上任何原因导致需要重新处理消息，都可以再次从 broker 获得。

kafka 的 producer 有一种异步发送的操作。这是为提高性能提供的。producer 先将消息放在内存中，就返回。内存中的消息会在后台批量的发送到 broker。

扩展性

kafka 使用 zookeeper 来实现动态的集群扩展，不需要更改客户端（producer 和 consumer）的配置。broker 会在 zookeeper 注册并保持相关的元数据（topic，partition 信息等）更新。而客户端会在 zookeeper 上注册相关的 watcher。一旦 zookeeper 发生变化，客户端能及时感知并作出相应调整。这样就保证了添加或去除 broker 时，各 broker 间仍能自动实现负载均衡。

负载均衡

负载均衡可以分为两个部分：producer 发消息的负载均衡和 consumer 读消息的负载均衡。

producer 有一个到当前所有 broker 的连接池，当一个消息需要发送时，需要决定发到哪个 broker（即 partition）。这是由 partitioner 实现的，partitioner 是由应用程序实现的。应用程序可以实现任意的分区机制。要实现均衡的负载均衡同时考虑到消息顺序的问题（只有一个 partition/broker 上的消息能保证按顺序投递），partitioner 的实现并不容易。

consumer 读取消息时，除了考虑当前的 broker 情况外，还要考虑其他 consumer 的情况，才能决定从哪个 partition 读取消息。