高可用可扩展数据层架构探讨

——基于 MySQL 的分布式低成本数据层

引言

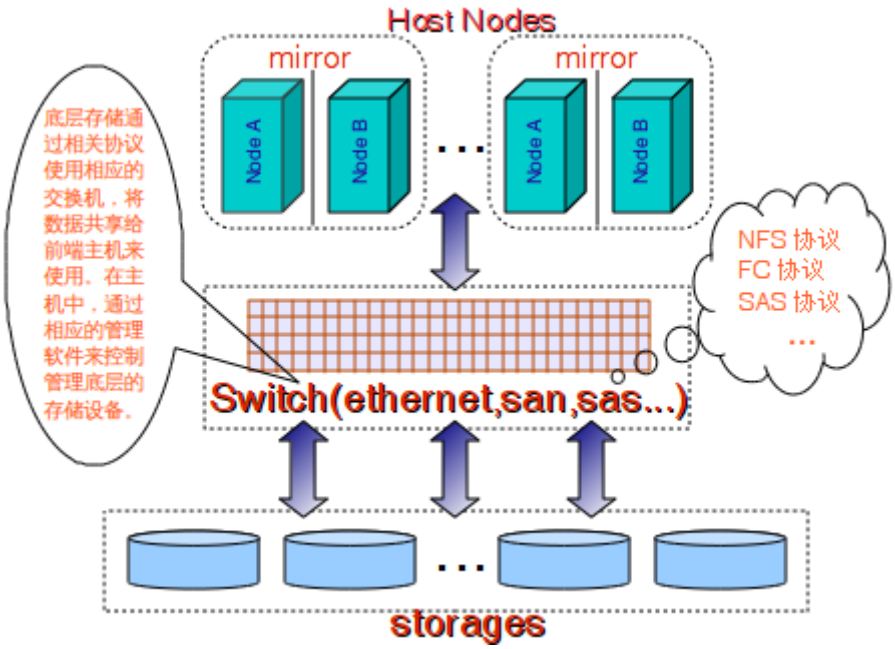
随着信息时代的高速发展，信息量的飞速增长，信息的存储自然也成为了这个时代中至关重要的一项技术。如何来保证数据存储技术能够适应信息量的增长速度以及对信息的高度依赖，也成为了现在一个非常重要的课题。本文将从数据库架构的层面，通过以开源的数据存储软件来构建分布式数据层的思路，期望实现一个低成本的高可用可扩展的数据层架构。

传统数据库架构

纵观各传统的商业数据库软件，大多以集中式架构为主，鲜有以分布式为设计理念的架构。这类数据库架构最大特点就是将所有的数据都集中在同一个数据库中，依赖大型高端设备来提供高处理能力和扩展性。并且由于数据库实例较少，可以较好的控制维护成本。

传统的集中式数据库架构在扩展性方面主要依赖于所运行的主机和存放数据的存储设备的扩展能力，也就是说依赖硬件本身的纵向扩展能力（Scale Up），很难做到较好的横向扩展（Scale Out）。而其可靠性也同样是以硬件设备为依托，主要通过 Share Storage 的方式来实现。如通过 IBM 提供的 HACMP，HP 提供的 ServiceGuard 等，SUN 提供的 Sun Cluster 等 OS 厂商提供的专用软件，或者是如 Veritas 这样的专业存储解决方案厂商提供的通用软件，在相应的平台上实现 Storage 的共享来实现可靠性保障。再如大家所熟知的传统商业数据库代表厂商 Oracle 的 RAC（Real Application Cluster），就更是一个非常典型的 Share Eerything 的集中式架构了。

我们可以通过下图来简单的描绘一下传统的数据库的典型架构：



从图中可以看出，传统的数据库架构在数据库软件所在的主机端大多都通过两台主机共享存储设备，平时其中一台主机使用存储设备上的数据通过数据库软件提供服务。这样的架构只能有一台主机（RAC 除外）上的数据库能够提供服务，另一台主机主要是用来做为热备冗余，但是不能启动数据库实例来提供服务。也就是说，其处理器的处理能力以及整个

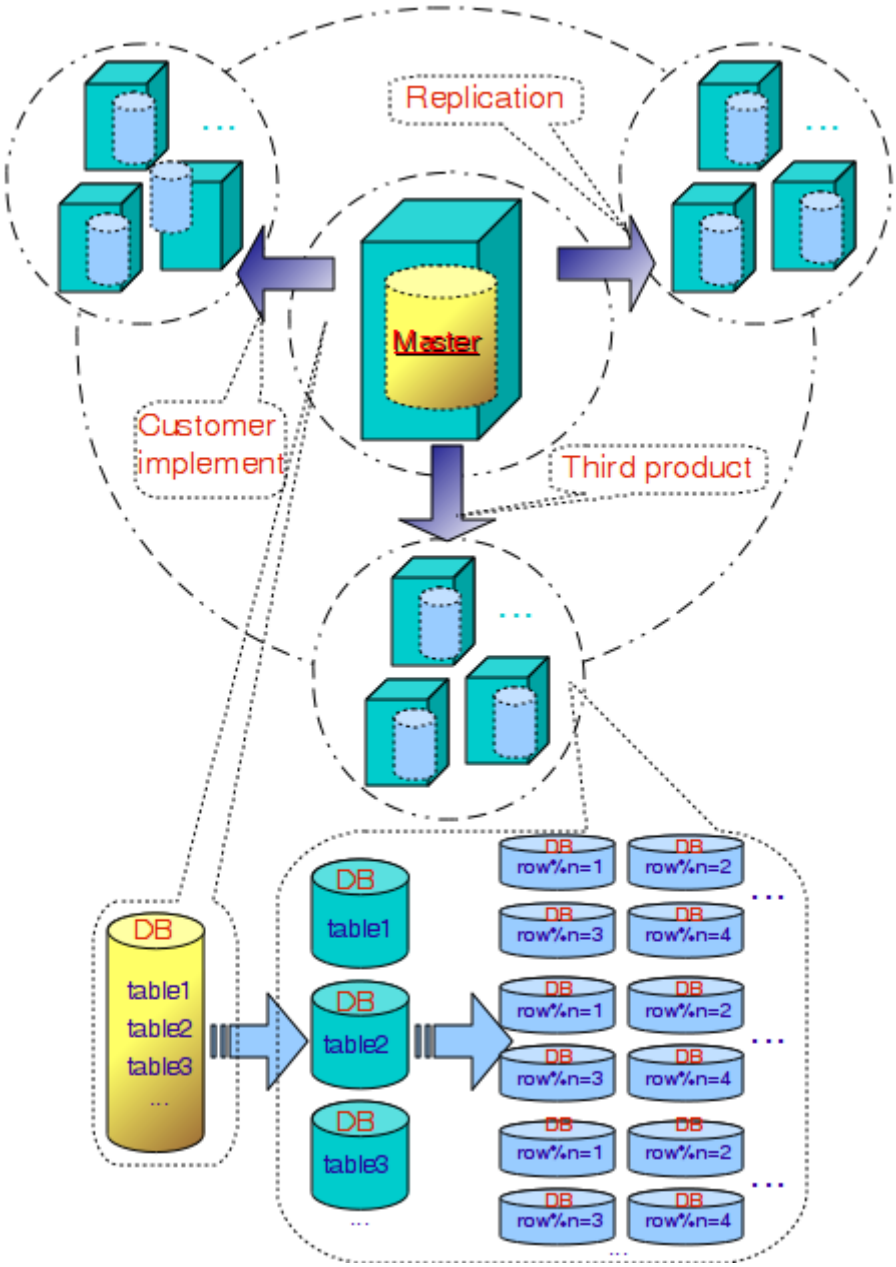
系统的内存容量就完全处决于这台主机的最大扩展能力，也就是我们常说的 Scale Up，很难通过增加主机数量来增加处理能力。而每个人都清楚，单台主机的扩展能力毕竟是有限的，即使是某些厂商的大型机，同样也有其扩展限制。
当然，可能大家会说 Oracle 的 RAC 可以通过增加节点的方式来突破单机限制。这个我确实承认，但是随着节点数的增加，每增加一个节点后处理能力的增加量，我个人持保留意见。
此外，传统架构对高端设备的依赖，无疑将直接导致系统成本的大幅度增加，甚至可能会导致系统被主机和硬件厂商所“绑架”，不得不持续的增加成本投入。

基于 MySQL 的可扩展和高可靠

MySQL 作为开源数据库的佼佼者，无论是软件本身的设计思想，还是推崇给广大 使用者们常用的架构思路都和传统的商业数据库软件背道而驰，弃用了传统的 Share Eerything 的思想，而采用了 Share Nothing 的思想。MySQL 的 Replication 实现机制，以及 MySQL Cluster 的架构设计，都体现了这一思想。也正是这一思想，给 MySQL 在可扩展性和高可靠性方面带来了非常灵活的架构设计思路。

- 可扩展性

在提升扩展性方面，我们最为常用的是通常通过 MySQL 自身的 Replication 功能，将同一份 MySQL 的数据以异步的方式同时复制到另外的一台或者多台 MySQL 主机上，并让这些 MySQL 主机同时对外提供查询服务。每增加一个复制的节点，查询处理能力也就得到了相应的增加，新增节点的处理能力就是整个系统增加的处理能力。由于其 Replication 主要是逻辑方式，同一个集群中可以有多家厂商的硬件，也可以使用多种不同的 OS，所以可以做到完全不受任何软/硬件平台限制，摆脱对单一平台的依赖。
当然，完们可能会对 MySQL Replication 的功能特性不满意，比如无法做到实时同步，单个 Slave 只能有一个 Master 等，进而通过通过第三方的开源软件（如 Continuent Tungsten 等），甚至是通过解析其开源的通信协议自行开发出来的复制软件来进行数据实时（或者异步）复制来达到和 Replication 完全相同甚至更好的效果。
在这一点上，传统数据库可能也同样具有某项功能能实现数据的复制，但是和 MySQL 来比较，由于只能依靠数据库本身的某项特性来完成，在架构的灵活性和可控性方面存在一些不足。
最后，在提升扩展性方面不得不说的数据切分（横向/纵向）的思想，同样可以在 MySQL 数据库上得到灵活的发挥，无论是以功能模块的方式和进行纵向的切分，还是以某个特定键（字段）的类 HASH 分段的方式进行横向的切分，抑或者是通过数据库本身的 Partition 功能进行切分，都可以在 MySQL 上得得到很好的实现。当然，无论是切分前的数据分散，还是切分后的数据路由和合并都离不开应用层的协作与配合，除非我们通过 MySQL Cluster 来实现。
对于提升扩展性的架构，通过下图会有一个更为直观的展现：



图中以 “Master” 为核心，可以通过不同的方式以三条“路线”将数据复制到相应的 MySQL 集群中对外提供服务。在实际的架构中， Master 就是一个数据写入点，复制出去的集群则可以对外提供相应的查询请求。在常见的 Web 应用系统中，查询请求远远大于写入请求，所以非常适合通过 MySQL 数据库使用类似的架构思想来解决实际的扩展性问题。

● 高可靠性

在保证高可靠性方面，我们同样可以通过 MySQL 的 Replication 为基础，加以应用层架构的简单配合或以及一些开源的第三方 HA 管理软件（如 heartbeat，MMM）来设计出多种非常灵活的高可靠架构。

对于数据写入点（Master），我们可以通过 MySQL 的 Replication 功能，将两台 MySQL 主机设置成双 A（Dual Master）的状态，并通过 HA 管理软件，通过对 MySQL 的状态检测，来判定 MySQL 的状态，并对外提供供单一的服务 IP 地址，以确保在任何时候有一台

MySQL 崩溃后可以马上马上切换到另外一台 MySQL 上提供服务。这样即可以自动的方式非常方便的让我们的写入点拥有高可靠性。当然，如果我们的应用程序也自己可以自动判断当一个点失效后马上自动切换到另外一个点就更好，基本上可以做到对外部完全透明的切换，对可用性的影响之小，比现在一些知名厂商的专业 HA 管理都要好很多。

对于数据查询点（Slave），高可靠性的实现就更容易了，我们可以从双 A 的两台 Master 端中的任意一台上通过数据复制搭建多个具有完全相同数据拷贝的 MySQL 节点，来保证任何时刻都可以有多台 MySQL 可以对外提供数据查询服务。当有一台 MySQL 崩溃之后，让系统马上将该节点从可以服务的节点中剔除出去。这通过应用架构的帮助，是非常容易做到的事情。

下面的图中很好的描绘了通过 MySQL Replication 来搭建提高可靠性的架构方式：

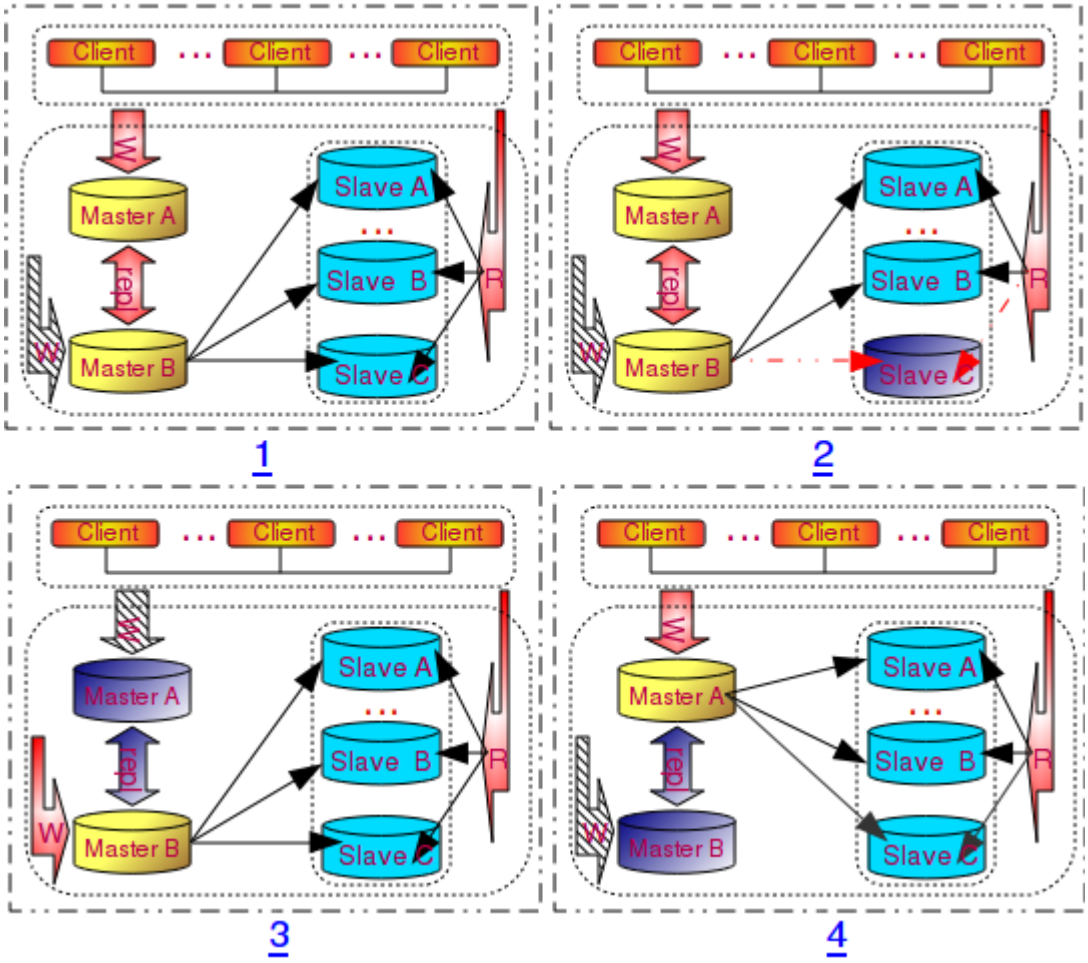


图 1 描绘了基本架构，图 2，图 3 和图 4 分别描绘了当读节点失效以及写节点中的任何一个失效后的高可靠性实现。

在高可靠性方面，除了我们自行控制将数据复制到多个 MySQL 主机上的方式之外，MySQL 还给我们提供了更为高级的方案，那就是 MySQL Cluster，一个完全的分布式数据库集群，而且是一个非常典型的 Share Nothing 的分布式数据库架构。数据层和 SQL 层分离，每份数据都以 2（或者更多）份拷贝的方式存放在不同的数据节点上，整个数据层的所有节点以分布式计算的思想共同处理整个数据库的数据，在保证高并发的处理能力的同时，以数据冗余的方式同时保证了高可靠性。希望进一步深入了解 MySQL Cluster 的朋友可以通过以下链接获取更为详细的信息 ：http://www.mysql.com/products/database/cluster

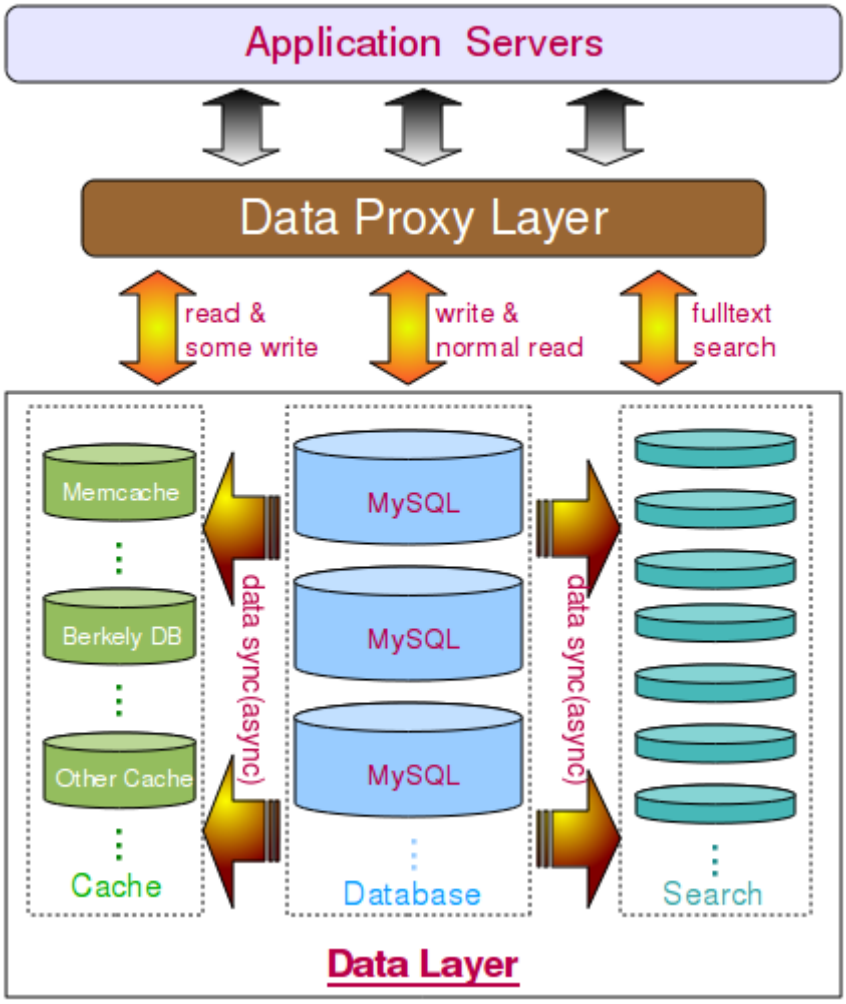
结合 MySQL、Cache 和 Search 的数据层

随着 Web2.0 的不断成熟，Web 应用系统的负载的增速越来越块，经常会让我们的系统不堪重压，尤其是数据库系统。而随着大家对用户体验的关注度的提高，响应速度和使用的便利性则成为了不可避免的话题。

在响应速度方面，由于数据库这种基于磁盘 IO 系统，不论我们如何优化，都不可能避免磁盘物理 IO。而磁盘上的物理 IO 的响应速度和内存中的 IO 响应速度完全不是一个数量级，所以我们很自然的想到了通过内存 Cache 的方式来提高响应速度。

而在使用的便利性方面最为典型的就是让用户以最简单方便的方式查询到自己想要的数据，也就是我们常说的数据搜索。关系型数据库的特性，决定了他很难提供类似于 Google 那样的可以全文检索的搜索系统。同样的，我们再次以借助“外力”的方式来完善服务。我们可以将数据库中的数据通过利用 Lucene 或者 Egothor 等类似的搜索引擎系统，或者是利用 Sphinx 之类的软件和数据库集成，为数据库增加全文检索的能力。

通过这样的思路，我们可以设计出一个以数据库（MySQL）来完成持久化和常规的数据访问功能，以分布式内存 Cache 系统（Memcached，BerkelyDB...）来提供对响应速度和并发能力极高的数据的访问，以第三方或者自行研发的分布式全文搜索系统来提供全文检索服务的全方位的分布式数据服务层，中间则通过应用架构的帮助实现一个可以统一的数据访问层，来控制数据的读取写入检索等操作，如下图：



很多朋友可能会问如何让 MySQL 中的数据同步（或异步）写入到 Cache 和 Search 系统中呢？其实实现方式真的很多，如果对实时性要求不是很高，也不希望Data Proxy Layer 中有太多的控制逻辑，我们完全可以通过队列的方式来以异步的方式实现；如果是对实时性要求较高，我们也可以通过 Data Proxy Layer 这一层应用来控制Cache 和 Search 中的数据更新；甚至我们还可以通过 MySQL 的用户自定义函数，以 Trigger 的方式，将数据实时的更新到Cache 集群。类似的方式我们可以想出很多很多，关键是要根据我们的应用场景，来选择一个最合适，最简单的实现方式来实现。

个人一直认为架构无所谓最好的，只有最合适的。任何一个架构都有其适用的场景，也有其相应的生命周期。随着应用场景的变化，或者是业务量的变化，都可能导致架构不足以应对的现象。引用一句电影台词:“出来混，迟早要还的!”