高可用性 IM 服务器架构设计

仇文超

（焦点科技股份有限公司）

摘要：阐述系统可用性的含义与当前业界技术状况。对于 C/S 中典型的 IM(即时通讯软件)服务器，以高可用性为研究重点，提出一种可行的整体架构设计方案。
关键词：高可用性；容错；服务器设计；高并发。

1 引言

1.1 背景

IM 软件(即时通讯软件)是一种常见的 C/S 结构的软件系统，不少公司在做，如著名的腾迅公司的 QQ，阿里巴巴旗下的阿里旺旺以及焦点公司的麦通等。

IM 的思路很简单，基本就是解决登录和消息转发工作。但是，需要较多的设计经验才能开发出稳定高效的 IM 服务器系统。

对于服务器设计通常对可用性都有要求，只是要求的标准不同。对于重要场合的 IM 系统，保证其稳定和高效是相当重要的。比如阿里旺旺及麦通这类用于客户商务洽谈的软件，如果消息不能稳定的收到，那么很可能会延误商机。

服务器端系统通常要求 7x24 小时不间断运行。这不仅要求硬件系统不能出现中断服务的宕机，而且要求软件系统不能出现无法提供服务的故障。 一般来说，硬件是无法保证其一直无故障且正常工作的，软件一般也不能保证不出现崩溃。 一个庞大的系统，通常的解决方案是保证每一个子系统的硬件都有备份系统，每一个软件也有不中断服务的能力。这对于系统架构者提出了相当高的要求[1]。

据报道，高铁列车上的同类功能计算机通常是配置三台。三台同时工作，一台出错了没关系，还有两台能提供服务。根据概率学知识可以得知，三台计算机同时出错的概率远小于一台计算机，这就大大提高了系统的可靠性。

1.2 术语含义

可用性

计算机系统的可用性(availability)是通过系统的可靠性(reliability)和可维护性(maintainability)来度量的。工程上通常用平均无故障时间(MTTF)来度量系统的可靠性，用平均维修时间（MTTR）来度量系统的可维护性。于是可用性被定义为 MTTF/(MTTF+MTTR) * 100% 。

业界根据可用性把计算机系统分为如下几类[2]：

可用比例 (Percent Availability)	年停机时间 (downtime/year)	可用性分类
99.5	3.7 天	常规系统(Conventional)
99.9	8.8 小时	可用系统(Available)
99.99	52.6 分钟	高可用系统(Highly Available)
99.999	5.3 分钟	故障可恢复系统(Fault Resilient)
99.9999	32 秒	容错系统(Fault Tolerant)

表 1.1 系统可用性分类

对于关键业务，停机通常是灾难性的。因为停机带来的损失也是巨大的。下面的统计数字列举了不同类型企业应用系统停机所带来的损失。

应用系统	每分钟损失(美元)
呼叫中心(Call Center)	27000
企业资源计划(ERP)系统	13000
供应链管理(SCM)系统	11000
电子商务(eCommerce)系统	10000
客户服务(Customer Service Center)系统	27000

表 1.2 停机给企业带来的损失[2]

坊间有传言，4 个 9 的可用性实际上是很难实现的目，至于 5 个 9 的 Web 站点，一半靠内功，另一半恐怕是要靠点运气。据说，使用 Erlang 的电信关键产品，达到了传说中的 99.9999999%可用性(即 9 个 9 的最高可用性标准)。正所谓是”没有最高可用，只有更高可用性”[3]。但是我们也不应该过度盲目的追求高可用性，提高可用性的同时一般会不同程度的提高成本。

容错

所谓容错，是指在故障存在的情况下计算机系统不失效，仍然能够正常工作的特性。容错即是 Fault Tolerance，确切地说是容故障（Fault），而并非容错误（Error）[4]。

计算机系统的容错性是指软件检测应用程序所运行的软件或硬件中发生的错误并从错误中恢复的能力，通常可以从系统的可靠性、可用性、可测性等几个方面来衡量。可靠性对于火箭发射之类关键性应用领域来说尤为重要。而对于通用计算机来说，一个重要的指标就是系统的可用性。可用性是一年的时间中确保系统不失效的时间比率。可测性在容错系统的设计过程中也是一个非常重要的指标，如果我们无法对某个系统进行测试，又如何能保证它不出问题呢？此外 MTTR 的大小直接影响着系统的可用性，而平均故障间隔（MTBF）则反映了系统的可靠性[5]。

单点故障

单点故障（single point of failure，简称 SPOF），从英文字面上可以看到是单个点发生的故障，通常应用于计算机系统及网络，这也是在设计 IT 基础设施时应避免的。

对于承担关键业务的核心系统而言，是不能存在单一节点故障的，否则该设计就有严重

缺陷。借助集群软件可比较容易的实现系统容灾，故障发生时，接替故障主机，保持业务连续性。但对于双机系统而言，进行容灾是没有办法做到无缝切换的，需要一个故障的恢复时间，可能长，也可能短。原因在于，当故障出现时系统宕机，这些系统的 CPU 中正在处理的数据、内存中驻留的重要数据以及 I/O 中正在等待写入/写出的数据就会丢失，对于业务系统而言，这意味交易不完整，正在进行的交易失败。当我们用备份系统接替工作时，备份系统会对相关数据进行检测，对数据进行回滚，然后接替故障系统工作，实现平台切换[6]。

集群

“集群（cluster）”[7] 在不同的地方可能会意味着不同的意义。通常集群可分成三种类型：

1) 故障迁移集群

最简单的故障迁移集群有两个节点：一个节点是活动的，另外一个节点是备用的，不过它会一直对活动节点进行监视。一旦活动节点出现故障，备用节点就会接管它的工作，这样就能使得关键的系统能够持续工作。

2) 高可用集群[8]

高可用集群通常会在非常繁忙的 Web 站点上采用，它们有多个节点来承担相同站点的工作，每个获取 Web 页面的新请求都被动态路由到一个负载较低的节点上。

3) 高性能计算集群[9]

高性能计算集群用来运行那些对时间敏感的并行程序，它们对于科学计算来说具有特殊的意义。

本文架构设计中所涉及的集群属于高可用集群。

2 关键技术

服务器端系统的设计有很多要求。限于篇幅有限，本文将以高可用性设计为探讨重点，研究 C/S 系统服务器架构设计中的多种技术。本文论述涵盖服务器端整体架构设计，包括应用层，缓存层及数据层。 如果要提高可用性则必须保证整个系统的每个子系统都不能有单点故障。

在高可用性系统中一个关键的思想就是冗余,不管是软件、硬件还是数据,都需要采取冗余的策略,从而在故障时可以及时恢复,否则一切高可用性的措施和方法都将失去意义。

高可用性系统有三种工作方式[4]，

1) 主从方式 （非对称方式）

工作原理：主机工作，备机处于监控准备状况(stand by)；当主机宕机时，备机接管主机的一切工作，待主机恢复正常后，按使用者的设定以自动或手动方式将服务切换到主机上运行，数据的一致性通过共享存储系统解决。

2) 双机双工方式（互备互援）

工作原理：两台主机同时运行各自的服务工作且相互监测情况，当任一台主机宕机时，另一台主机立即接管它的一切工作，保证工作实时，应用服务系统的关键数据存放在共享存储系统中。

3) 集群工作方式（多服务器互备方式）

工作原理：多台主机一起工作，各自运行一个或几个服务，各为服务定义一个或多个备用主机，当某个主机故障时，运行在其上的服务就可以被其它主机接管。

3 业内现状

3.1 提高可用性的一般途径

在 UNIX 系统上创建高可用性计算机系统，业界的通行做法，也是非常有效的做法， 就是采用集群系统（Cluster），将各个主机系统通过网络或其他手段有机地组成一个群体，共同对外提供服务。创建集群系统，通过实现高可用性的软件和冗余的高可用性的硬件组合起来，消除单点故障[10]。

通常我们构建的集群属于高可用性集群。集群提高系统可用性的同时，也大大提高了系统的并发能力，从而提高了整体系统对于巨大业务量的处理性能。

3.2 商业解决方案

商业方案解决高可用性通常是通过硬件来进行 HA。常见的硬件有比较昂贵的 NetScaler、F5、Radware 和 Array 等商用的负载均衡器，它的优点就是有专业的维护团队来对这些服务进行维护，缺点就是花销太大，所以对于规模较小的网络服务来说暂时还没有必要使用。

3.3 开源方案

开源集群方案较多[11]，较为著名的有 LVS[12]、HAProxy 及 Nginx。

HAProxy 及 Nginx 工作在网络的 7 层之上，有点类似于 LVS 的 VS/NAT 模式。而 LVS 可以工作在 4 层，效率较高。

Nginx 仅能支持 http 和 Email，这样就在适用范围上面小很多，这个它的弱势。它跟 LVS 一样，本身仅仅就只是一款负载均衡软件；单纯从效率上来讲 HAProxy 更会比 Nginx 有更出色的负载均衡速度，在并发处理上也是优于 Nginx 的。HAProxy 可以对 Mysql 读进行负载均衡，对后端的 MySQL 节点进行检测和负载均衡，不过在后端的 MySQL slaves 数量超过 10 台时性能不如 LVS，所以推荐优先使用 LVS。

LVS 是 Linux Virtual Server 的简写，意即 Linux 虚拟服务器，是一个虚拟的服务器集群系统。本项目在 1998 年 5 月由章文嵩博士成立，是中国国内最早出现的自由软件项目之一。

3.4 合适的选择

LVS 抗负载能力强、是工作在网络 4 层之上仅作分发之用，没有流量的产生。所以使用 LVS 来构建高可用集群是比较好的选择。本文的很多设计思想借鉴于 LVS 的架构。

4 架构方案

即时通讯程序是常见的 C/S 结构系统。 根据上述讨论、多方面研究及实践，作者提出了一种即时通讯系统的服务器端架构设计方案。

4.1 系统整体架构

高可用性 IM 服务器系统(本节下文简称“系统”)整体架构见下图，

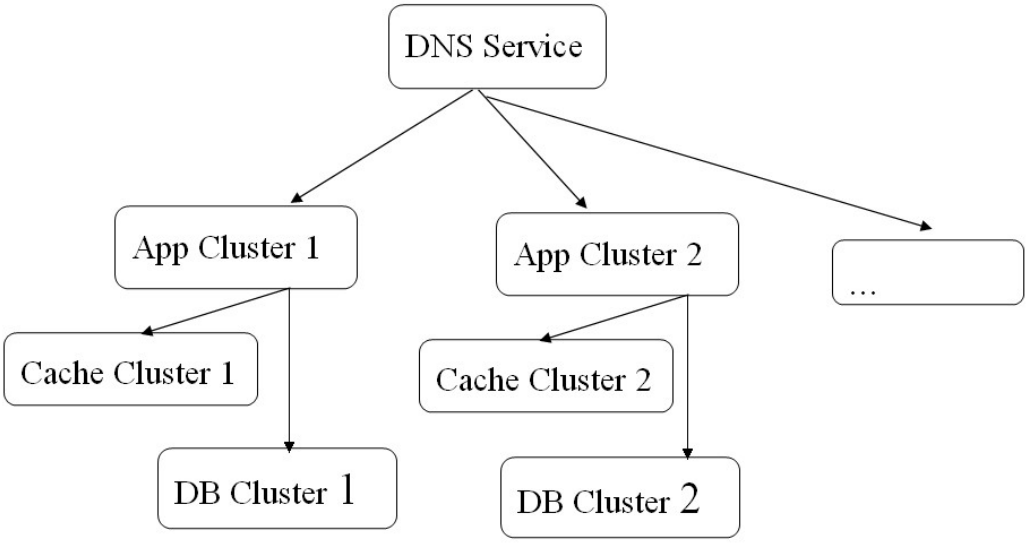

图 4.1 高可用性 IM 服务器系统整体架构图

系统主要由 App Cluster(应用集群)、Cache Cluster(缓存集群)及 DB Cluster(数据库集群)三个子系统组成。系统设计充分考虑了以下原则[13][14][15]：

1) 子系统必须不能有 SPOF。设计中使用多台服务器组成的集群来避免 SPOF，所有服务器都有备份机器。

2) 使用的集群技术借鉴于成熟的架构，而不是自己创造。我们是在快速开发商用系统，而不是在搞科研。所以，有现成的成熟的软件模块或者技术可以直接拿过来用，没必要自己重新制造“轮子”。本文的集群设计中主要使用了 LVS 技术。

3) 所有软件都有监测系统或备份系统，任何进程都可以在运行过程中被重启。

4) 使用成熟的心跳软件来进行服务器间的健康检测(如 keepalived, heartbeat 等)。

5) 使用 VRRP 等协议来避免路由及交换机等设备的 SPOF[16][17]。

如图 4.1 所示，系统使用地理分布式服务器集群。

DNS 负载均衡技术的实现原理是，在 DNS 服务器中为同一个主机名配置多个 IP 地址，应答 DNS 查询时，DNS 服务器对每个查询根据 DNS 文件中主机记录的 IP 地址按顺序返回不同的解析结果，将客户端的访问引导到不同的机器上去，使得不同的客户端访问不同的服务器，从而达到负载均衡的目的。

DNS 负载均衡技术既有优点也有缺点。优点就是技术实现比较灵活、方便，简单易行，成本低，缺点主要是不能够按照服务器的处理能力分配负载。据说，现在有些 DNS 服务提供按照地理位置就近解析，也就是将 DNS 查询与用户 IP 信息挂钩，当用户查询域名时，首先根据用户 IP 地址，确定其地理位置，然后根据就近原则，查找最近的服务器并将其 IP 返回给请求者。作者目前对此类 DNS 没有详细探究。

如果不使用 DNS 负载均衡，那么有可能要自己处理不同集群间的负载均衡，这方面难度较大。

基于 BGP 的地理分布服务器集群调度是一种较成熟的方法。利用 BGP 协议在 Internet 的 BGP 路由器中插入 Virtual IPAddress 的路由信息。在不同区域的 LVS 集群向它附近的 BGP 路由器广播到 Virtual IP Address 的路由信息，这样就存在多条到 Virtual IP Address 的路径，Internet 的 BGP 路由器会根据评价函数选出最近的一条路径。这样，我们可以使得用户访问离他们最近的 LVS 集群。当一个 LVS 集群系统失效时，它的路由信息自然不会在 Internet 的 BGP 路由器中交换，BGP 路由器会选择其他到 Virtual IP Address 的路径。这样，可以做到抗灾害性（Disaster Tolerance）[18]。

图 4.1 中 Cluster1 及 Cluster2 代表了位于两个不同地理位置的集群。每个集群包含了三个子系统，分别是 App Cluster、Cache Cluster 及 DB Cluster。App Cluster 表示应用服务器集群，Cache Cluster 表示缓存服务器集群，DB Cluster 表示持久化服务器(数据库服务器)集群。其实，每个子系统都是一个集群。每个子系统都设计成集群的直接后果就是，每个子系统都有较高的可用性。不会因为其中的一台机器宕机而导致该子系统不能提供服务，进而降低整个系统的可用性。

由于 Cache Cluster 及 DB Cluster 会保存数据，所以在不同的地理集群间必然存在着数据同步的问题。这无论如何是无法避免的。解决方法无非就是复制，即不断在不同地理集群间传递更新的数据集。

但是，复制的策略非常复杂。本文对此不作深入讨论。读者可以参考多个数据中心数据同步技术的开源实现。其中，著名的谷歌(google) 公司旗下有一个系统叫 Google Spanner，是 Google 的全球级的分布式数据库 (Globally-Distributed Database) 。Spanner 的扩展性达到了令人咋舌的全球级，可以扩展到数百万的机器，数已百计的数据中心，上万亿的行。更惊人的是，除了夸张的扩展性之外，他还能同时通过同步复制和多版本来满足外部一致性，可用性也是很好的，冲破 CAP 的枷锁，在三者之间完美平衡[19]。

下面针对单个集群中的不同子系统进行讨论。

4.2 应用层架构[20]

应用层用于处理业务，如 IM 消息的记录及转发、音频视频数据转发等。实际中应用层可能的做法是把普通业务处理、文件传送、语音视频等不同的业务放到不同的集群中，但这并不影响我们在此对于通用的架构设计进行讨论。因为从更高层次来说，这些不同的业务只是处理逻辑及算法不同而已，都是属于应用层系统。

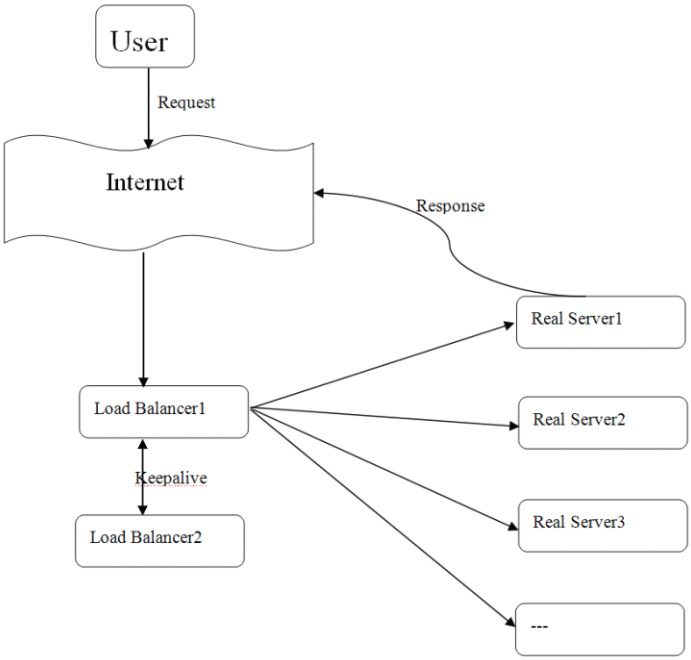

图 4.2 高可用性 IM 服务器应用层架构图

如图 4.2 所示，应用层架构是典型的 LVS 集群架构。User(用户)发出请求，Loader Balancer 收到请求数据并按照一定的负载均衡算法，转发给某台 Real Server 处理，Real Server 处理完成后数据(即图 4.2 中 Response)不经过 Load Balancer 直接返回给 User (假定使用 VS/DR 或 VS/TUN 方式)。

两台 Load Balancer 相互监控，一台 Load Balancer 宕机，另一台会立即接替其工作。Real Server 的健康状况由 Load Balancer 监控。对于应用的部署和处理来说，这些 Real Server 是相互平等的，Load Balancer 把收到的数据转发给任何一台 Real Sever 处理都是可以的。任何一台 Real Server 宕机都没关系。如果某台 Real Server 不能正常工作，短时间内 Load Balancer 会监测到并更新其维护的活跃的 Real Server 表，后续的数据就不会发给那台 Real Server。

应用软件运行在 Real Server 上，这些应用处理所有业务，如查询缓存、转发消息及查询数据库等。为了提高系统的可用性，对于应用软件的设计也是非常重要的[21]。

一般来说，我们要求每个进程运行的过程中都可以重启，而且重启之后可以继续正常工作。并且，进程应该可以平滑升级。也就是说，可以在不停止服务的情况下，更新服务器软件系统或者模块。这对于应用软件设计来说，提出了相当高的要求。但是，很多情况下，这是可以做到的。作者研究过不中断服务升级服务器软件的途径，并写过类似的文章介绍某些情况下可以不终止服务平滑升级服务器软件的设计方法。

另外，对于应用软件的设计，作者提倡多采用多进程异步模式的设计方法。一是因为通常情况下多进程结构的程序容错能力较好。另外，多进程软件更适用于分布式环境，可以在

同一服务器上启动多个不同的进程实例。 软件如果适应于分布式环境将是大有好处的，因为单机系统性能提升空间有限且不易拓展，分布式系统必将是大势所趋。 如果采用多进程设计时应该尽量使用异步 IO，网络事件、缓存读写及数据库读写都应该使用异步接口，否则就会严重拉低进程的业务处理能力。如何设计异步服务器软件是个比较大的话题，不在本文的论述范围。

4.3 缓存层架构

缓存层的构架设计图如下所示，

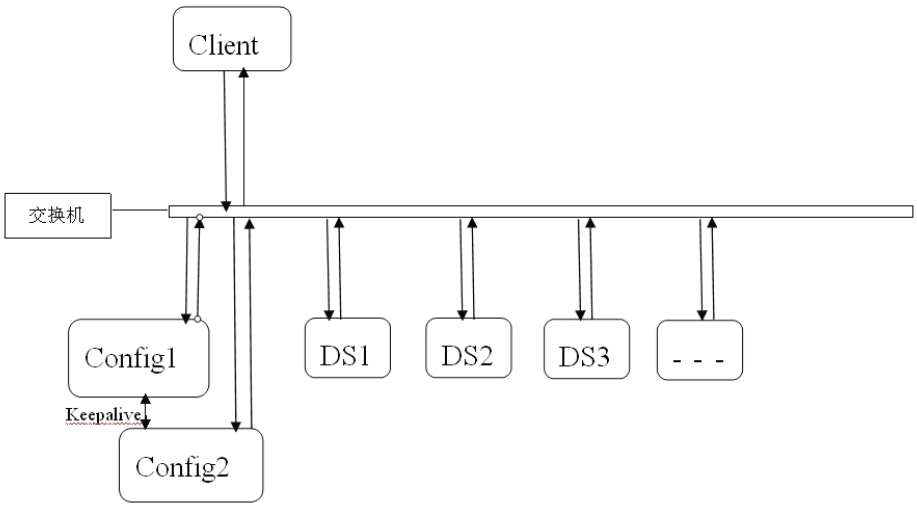

图 4.3 高可用性 IM 服务器应用层架构图

缓存系统[22]也是一个集群，其中 Config1 及 Config2 是配置信息服务器，DS1、DS2...表示不同的缓存数据集，但是每个 DS(即 Data Set)并不需要独占一台服务器，也就是可以把多个 DS 放在一台服务器上。

Config 服务器是轻量级的，在大部分时候，Config 服务器不可用对集群的服务是不造成影响的[23]。Config1 与 Config2 通过心跳来监控对方，同一时刻只有一台工作，一台宕机另一台立即接替其工作。DS 的健康状况由 Config 服务器监控，一旦某台存放缓存数据的服务器(Cache 服务器)宕机那么多个 DS 会丢失，其它 Cache 服务器上缓存同样数据的 DS 会保证继续提供同样的数据。

Config 服务器上会保存一个缓存数据映射表。该表通常使用一致性 Hash 算法来映射 Key 到 Cache 服务器及 DS。另外，客户会缓存该映射表。

数据的缓存策略很重要，缓存数据的缓存映射关系由 Config 服务器完成。它会保证同一份缓存数据在不同的 Cache 服务器上备份。

Client(客户端)发出请求时，通过查询映射表获知应该从哪一个 DS 上获取数据。Client 直接向 DS 发出数据请求，DS 通过高效的算法(Hash 表或树等)查询出请求结果并直接返回给 Client。中间不经过其它服务器中转，效率非常高。

4.4 持久层架构

持久层[24][25][26]的构架设计原理如图 4.4 所示，

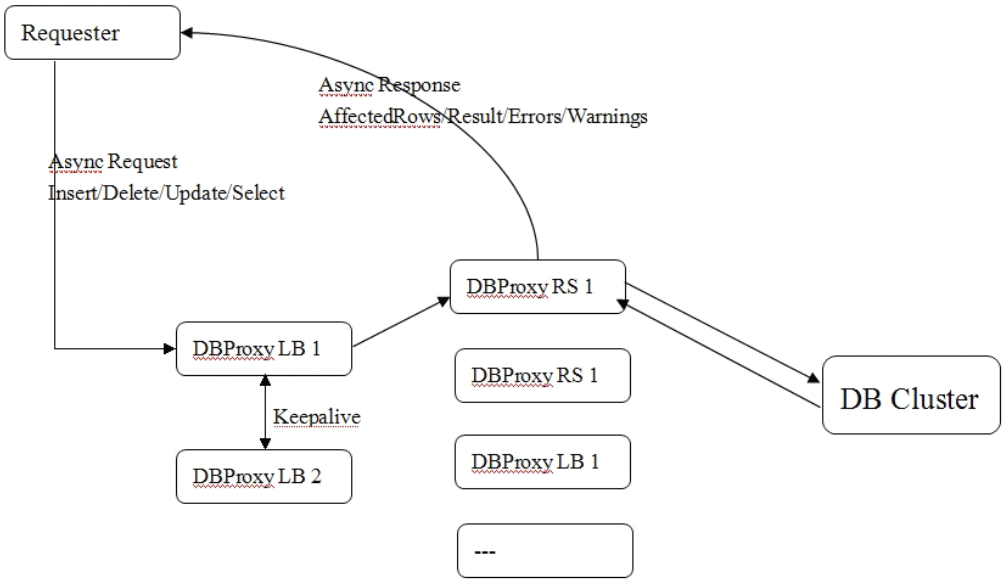

图 4.4 高可用性 IM 服务器持久层架构图

Requester 表示请求数据库的客户端(即 Client)，DBProxy LB1 及 DBProxy LB2 表示负载均衡器，DBProxy RS 表示数据库访问服务器，DB Cluster 表示数据库集群。

持久层使用了 DBProxy(数据库代理)的思想及 LVS 架构。Requester 请求数据时，DBProxy LB 接收数据并按照一定的负载均衡算法转发给某台 DBProxy RS，DBProxy RS 根据特定协议(如图 4.4 所示,Insert/Delete/Update/Select 对应不同的 SQL 操作) 拼接出 SQL 语句并同步访问 DB Cluster，DB Cluster 将结果同步返回给 DBProxy RS，DBProxy RS 将结果异步返回给 Requester。原理类似于 UMP(Unified MySQL Platform)系统[27]，不同点是引入了异步请求机制。阿里巴巴的 UMP 系统基本上实现 Mysql 的通信协议，但是这个工作量非常大。对于类似 IM 这样 DB 操作不太复杂的应用，作者建议使用如图 4.4 所示较为简单的协议。

DBProxy 服务器(下文称 proxy 服务器)向用户提供访问 MySQL 数据库的服务，proxy 服务器通过用户名获取到用户的认证信息、资源配额的限制（例如最大连接数、QPS、IOPS 等），以及后台 MySQL 实例的地址（列表），再将用户的 SQL 查询请求转发到正确的 MySQL 实例上。

除了数据路由的基本功能外，proxy 服务器中还可实现了资源限制、屏蔽 MySQL 实例故障、读写分离、分库分表、记录用户访问日志的功能。Proxy 服务器是无状态的，服务器宕机不会对系统中其他服务器造成影响，只会造成连接到该 proxy 的用户连接断开。多台 proxy 服务器采用 LVS HA 方案实现负载均衡，用户应用重连后会被 LVS 定向到其他的 proxy 服务器上。

持久层架构设计中充分考虑到了高可用性，DBProxy 集群保证了整个数据库访问的高可用性，而 DBCluster 保证了数据库集群的高可用性。DB Cluster 可以采用现成的方案，如一主多从的模式。整体设计中创新的提出了异步请求数据库的思想，这样的设计导致的直接后果就是，访问者不用再等待请求返回就可以处理其它业务。这样就释放了线程的 IO 请求时间，可以大大降低程序运行时所需的线程数量。

作者认为，服务器端软件设计思想中，进程线程的结构设计方式大体上分为三个时期。即，八九十年代的多进程方式，后来的多线程方式，以及目前流行的异步方式。多进程方式是早期 Unix 环境下服务器设计的常用模式，由于进程比线程重量得多，后来发展成为多线程。多线程程序比比皆是，这得益于大名鼎鼎的 NPTL(Next-Generation POSIX Threads)。多线程程序可以更高效的创建、销毁线程及线程间切换。但是，多线程程序带来了同步问题，而且大量的线程间频繁切换带来的时间消耗不可忽视，尤其对于核心应用系统更为明显。如果你对 Unix 环境比较熟悉，你不难注意到， Unix 环境下有很多古老的著名软件，如 apache 等，这些 Unix 早期软件多半是多进程的，甚至是 Oracle 也是多进程设计；而近现代的软件以多线程设计居多，比如 Mysql；如果留心最近一两年流行的开源软件，你会发现它们有些采用比较前卫的异步结构设计。 所以，目前异步模式的服务器程序设计更占优势，这可以说是把服务器的最后一滴 CPU 都“毫不留情”的榨干，让它一点“偷懒”的时间都没有。

4.5 小结

本节向读者详细介绍 IM 服务器架构设计思路。从前端业务，到中间缓存，再到后端数据库，每一个环节都是集群，这样就可以大大提高整个系统的可用性。 同时集群设计的副产品就是可以提高系统的并发，从而可以为更多的用户同时提供服务。

系统具有很高的扩展性，不同的地理区域集群是可以随时添加的，同一地理区域的集群中应用集群、缓存集群及数据库集群都是可以随时扩展集群中单机数量的。 使用强大的 LVS 技术，使得单一集群中机器数量可以达到上百台甚至更多，这样的系统其扩展性可想而知。

本文设计中提出了异步数据库请求等思想，把应用从直接请求数据库的逻辑中解放出来，让它几乎不用额外的 IO 线程就可以非常容易的访问数据库。

5 结束语

本文简要介绍了系统高可用性的含义以及高可用性系统设计方法，针对 IM 服务器，作者提出了一套硬件、软件架构方案。限于篇幅有限，众多细节本文并未涉及，如集群中缓存层 Config 服务器与 Cache 服务器通信实现细节，集群中所用的软件模块选择及开发等。但是，忽略众多细节并不影响读者理解 IM 服务器整体架构以及如何从硬件到软件综合考虑提高系统可用性的思想。

作者参照众多现有系统架构，提出一套 IM 服务器的系统架构设计方案，该设计具有相当高的可扩展性。 其中部分软件模块实现起来需要一定的精力，但是有的模块可以直接使用开源软件，或者稍作修改即可。

6 参考文献

[1] Gray, J., Why Do Computers Stop and What Can We Do About It, 6th International Conference on Reliability and Distributed Databases, IEEE Press, 1987.

[2] Linux 高性能计算集群 [DB/OL] http://www.ibm.com/developerworks/cn/linux/cluster/hpc/part1/index.html

[3] Joe Armstrong . Programming Erlang: Software for a Concurrent World[M]. 2007-07-18

[4] Baidu baike [DB/OL] http://baike.baidu.cn/view/2700299.htm

[5] Laprie, J. C., Dependable Computing and Fault Tolerance: Concepts and Terminology, Proc. 15th FTCS, IEEE Press, pp. 2-11, 1985.

[6] Pradhan, D.K., Fault Tolerant Computing: Theory and Techniques, Vol I and II, Prentice Hall, Englewood Cliffs, 1986.

[7] 蔡学森,戴金波,宗国仕. 现代操作系统下的高可用集群系统概述[J]. 长春师范学院学报. 2006,08.

[8] 刘彦锋. 基于 Linux 下的高可用集群系统[D]. 吉林大学. 2004.

[9] 胡章平. 集群系统与分布式计算[J]. 电脑知识与技术. 2006,05.

[10] 高可用性软件的设计与实现[J]. 计算机工程. 2004,08.

[11] 杨帆. Hadoop 平台高可用性方案的设计与实现[D]. 北京邮电大学. 2012.

[12] Linux virtual server [DB/OL] http://www.linuxvirtualserver.org/

[13] Johnson, B.W., Design and Analysis of Fault Tolerant Digital Systems, Addison Wesley, Reading, 1989.

[14] Avizienis, A., Software Fault Tolerance, Proc. 1989 IFIP World Computer Conference, IFIP Press., 1989.

[15] Mourad, J., The Reliability of the IBM/XA Operating System, Proc. 15th FTCS, IEEE Press, 1985.

[16] Virtual Router Redundancy Protocol (VRRP) [DB/OL] http://tools.ietf.org/html/rfc3768

[17] 李秀剑. 基于 VRRP 的防火墙高可用性设计与实现[D]. 西安建筑科技大学. 2007.

[18] 章文嵩. 可伸缩网络服务的设计与实现 [DB/OL] http://zh.linuxvirtualserver.org/node/7

[19] Google, Inc. Spanner: Google’s Globally-Distributed Database[J].

[20] 熊乐. Google 集群系统技术综述[J]. 中国科技信息. 2009,06.

[21] 吴雨彬. 一个高可用性容灾系统的设计与实现[D]. 复旦大学. 2011.

[22] 杨艳,李炜,王纯. 内存数据库在高速缓存方面的应用[J]. 现代电信科技. 2011,12.

[23] 淘宝开源 Key/Value 结构数据存储系统 Tair 技术剖析 [DB/OL] http://www.infoq.com/cn/articles/taobao-tair

[24] 龚卫华. 数据库集群系统的关键技术研究[D]. 华中科技大学. 2006.

[25] 万春. 基于 Linux 的数据库集群系统的研究[D]. 华中科技大学. 2004.

[26] 韩毅. 面向集群体系结构的海量数据库管理集成技术研究与实现[D]. 国防科学技术大学. 2006.

[27] 低成本和高性能 MySQL 云数据的架构探索 [DB/OL] http://blog.yufeng.info/archives/2349