软件复用-讨论（II）

范亮_1352899

一、分布式：高可用性，高吞吐量

异步化方式（SEDA）：针对高并发的分布式系统，NIO 方式很早提出来就是针对这个问题的，包括两部分，一部分是后端系统之间的异步，典型的是 Netty、Mina、xsocket 等成熟的框架，另外一部分是前端接入 http 的异步 servlet(都有相应的规范)， 在 Http1.1 中支持，对客户端 LCP 来讲是同步调用的的。典型是有 Jetty6、jetty7、tomcat6、tomcat7 以及其他商业服务器都有实现。基本思想都是采用非阻塞方式，接受线程和工作线程分离，提高整个系统的吞吐量。好处是，整个系统的可用性大大增加，系统之间耦合度大大降低，并且有比较高的吞吐量，各个系统能高速运转，不会相互等待资源，可以充分利用资源。同时整个系统也可以有很好的伸缩性。弊端是，中肯的讲，对 CPU 的要求还是比较高的。

并行计算（Parallel Computing）是指同时使用多种计算资源解决计算问题的过程，是提高计算机系统计算速度和处理能力的一种有效手段。它的基本思想是用多个处理器/进程/线程来协同求解同一问题，即将被求解的问题分解成若干个部分，各部分均由一个独立的处理机来并行计算。
和 MR 的区别在于，它是基于问题分解的，而不是基于数据分解。

二、数据分区，分区，一致性，缓存问题

分布式文件系统(Cloudfs)：反复研究 HDFS，TFS，Gridfs(Mongodb)，FastDFS 基础上研发出来的分布式文件系统。存储架构与 FastDFS 相似，包括数据结点(Node)，数据组（Group），分区(Region)三级；数据结点(Node)为最终物理存储结点，相同数据组（Group）的不同结点会进行实时的数据同步，即同组的结点数据最终一致；?相同分区(Region)内的文件会自动去重，即相同内容的文件在同一个分区(Region)只会有一份。Cloudfs 通过使用消息队列(CloudMQ)进行同组结点间的数据序列同步，从而保证最终一致性，这一点跟 FastDFS 的 binlog 双向同步最为相似。Cloudfs 使用 Zookeeper 来监测存储结点状态维护和变更，使用 CloudRedis 存储文件索引信息，使用 Nginx 作为文件下载服务器。性能优秀，单台 Server 线上监测得到的数据，文件上传的 IOPS 可以达到 1200，上传速率达到25MBps,复制文件TPS达到9000以上,创建和删除文件的TPS达到30000以上。支持存储结点的动态加入和退出，支持线上扩容，数据多备份，结点动态负载均衡，最大理论可支持文件数量达千亿以上，支持结点数 3000 以上。
对于分布式系统的一致性，尽量满足可用性，一致性可以通过校对来达到最终一致的状态。

在一些高并发高性能的场景中，使用 cache 可以减少对后端系统的负载，承担可大部分读的压力，可以大大提高系统的吞吐量，比如通常在数据库存储之前增加 cache 缓存。

但是引入 cache 架构不可避免的带来一些问题，cache 命中率的问题， cache 失效引起的抖动，cache 和存储的一致性。

Cache 中的数据相对于存储来讲，毕竟是有限的，比较理想的情况是存储系统的热点数据，这里可以用一些常见的算法 LRU 等等淘汰老的数据；随着系统规模的增加，单个节点 cache 不能满足要求，就需要搭建分布式 Cache；为了解决单个节点失效引起的抖动 ，分布式 cache 一般采用一致性 hash 的解决方案，大大减少因单个节点失效引起的抖动范围；而对于可用性要求比较高的场景，每个节点都是需要有备份的。数据在 cache 和存储上都存有同一份备份，必然有一致性的问题，一致性比较强的，在更新数据库的同时，更新数据库 cache。对于一致性要求不高的，可以去设置缓存失效时间的策略。

Memcached 作为高速的分布式缓存服务器，协议比较简单，基于 libevent 的事件处理机制。

Cache 系统在平台中用在 router 系统的客户端中，热点的数据会缓存在客户端，当数据访问失效时，才去访问 router 系统。

当然目前更多的利用内存型的数据库做 cache，比如 redis、mongodb；redis 比 memcache 有丰富的数据操作的 API；redis 和 mongodb 都对数据进行了持久化，而 memcache 没有这个功能，因此 memcache 更加适合在关系型数据库之上的数据的缓存。

三、负载均衡

随着平台并发量的增大，需要扩容节点进行集群，利用负载均衡设备进行请求的分发；负载均衡设备通常在提供负载均衡的同时，也提供失效检测功能；同时为了提高可用性，需要有容灾备份，以防止节点宕机失效带来的不可用问题；备份有在线的和离线备份，可以根据失效性要求的不同，进行选择不同的备份策略。

一个大型的平台包括很多个业务域，不同的业务域有不同的集群，可以用 DNS 做域名解析的分发或轮询，DNS 方式实现简单，但是因存在 cache 而缺乏灵活性；一般基于商用的硬件 F5、NetScaler 或者开源的软负载 lvs 在 4 层做分发，当然会采用做冗余(比如 lvs+keepalived)的考虑，采取主备方式。
4 层分发到业务集群上后，会经过 web 服务器如 nginx 或者 HAProxy 在 7 层做负载均衡或者反向代理分发到集群中的应用节点。
选择哪种负载，需要综合考虑各种因素（是否满足高并发高性能，Session 保持如何解决，负载均衡的算法如何，支持压缩，缓存的内存消耗）；下面基于几种常用的负载均衡软件做个介绍。

LVS，工作在 4 层，Linux 实现的高性能高并发、可伸缩性、可靠的的负载均衡器，支持多种转发方式(NAT、DR、IP Tunneling)，其中 DR 模式支持通过广域网进行负载均衡。支持双机热备(Keepalived 或者 Heartbeat)。对网络环境的依赖性比较高。
Nginx 工作在 7 层，事件驱动的、异步非阻塞的架构、支持多进程的高并发的负载均衡器/反向代理软件。可以针对域名、目录结构、正则规则针对 http 做一些分流。通过端口检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点，不过其中缺点就是不支持 url 来检测。对于 session sticky，可以基于 ip hash 的算法来实现，通过基于 cookie 的扩展 nginx-sticky-module 支持 session sticky。
HAProxy 支持 4 层和 7 层做负载均衡，支持 session 的会话保持，cookie 的引导；支持后端 url 方式的检测；负载均衡的算法比较丰富，有 RR、权重等。
对于图片，需要有单独的域名，独立或者分布式的图片服务器或者如 mogileFS，可以图片服务器之上加 varnish 做图片缓存。

四、系统监控

大型分布式系统涉及各种设备，比如网络交换机，普通 PC 机，各种型号的网卡，硬盘，内存等等，还有应用业务层次的监控，数量非常多的时候，出现错误的概率也会变大，并且有些监控的时效性要求比较高，有些达到秒级别；在大量的数据流中需要过滤异常的数据，有时候也对数据会进行上下文相关的复杂计算，进而决定是否需要告警。因此监控平台的性能、吞吐量、已经可用性就比较重要，需要规划统一的一体化的监控平台对系统进行各个层次的监控。

平台的数据分类

应用业务级别：应用事件、业务日志、审计日志、请求日志、异常、请求业务 metrics、性能度量

系统级别：CPU、内存、网络、IO

时效性要求

阀值，告警：

实时计算：

近实时分钟计算

按小时、天的离线分析

实时查询

架构

节点中 Agent 代理可以接收日志、应用的事件以及通过探针的方式采集数据，agent 采集数据的一个原则是和业务应用的流程是异步隔离的，不影响交易流程。

数据统一通过 collector 集群进行收集，按照数据的不同类型分发到不同的计算集群进行处理；有些数据时效性不是那么高，比如按小时进行统计，放入 hadoop 集群；有些数据是请求流转的跟踪数据，需要可以查询的，那么就可以放入 solr 集群进行索引；有些数据需要进行实时计算的进而告警的，需要放到 storm 集群中进行处理。

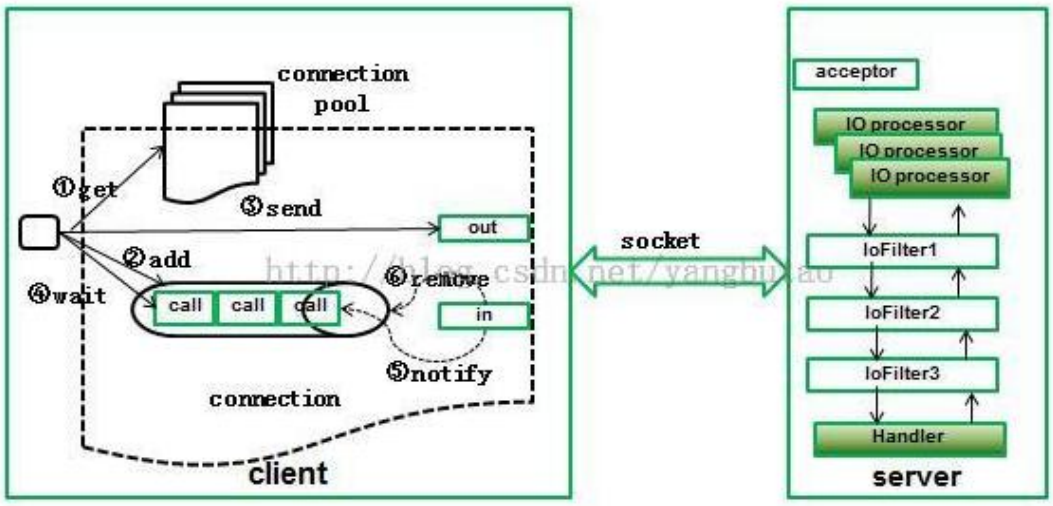
数据经过计算集群处理后，结果存储到 Mysql 或者 HBase 中。

监控的 web 应用可以把监控的实时结果推送到浏览器中，也可以提供 API 供结果的展现和搜索。

五、ID 分配

六、通信可靠高效

通信组件用于业务系统内部服务之间的调用，在大并发的电商平台中，需要满足高并发高吞吐量的要求。
整个通信组件包括客户端和服务端两部分。
客户端和服务器端维护的是长连接，可以减少每次请求建立连接的开销，在客户端对于每个服务器定义一个连接池，初始化连接后，可以并发连接服务端进行 rpc 操作，连接池中的长连接需要心跳维护，设置请求超时时间。
对于长连接的维护过程可以分两个阶段，一个是发送请求过程，另外一个是接收响应过程。在发送请求过程中，若发生 IOException，则把该连接标记失效。接收响应时，服务端返回 SocketTimeoutException，如果设置了超时时间，那么就直接返回异常，清除当前连接中那些超时的请求。否则继续发送心跳包(因为可能是丢包，超过 pingInterval 间隔时间就发送 ping 操作)，若 ping 不通(发送 IOException)，则说明当前连接是有问题的，那么就把当前连接标记成已经失效；若 ping 通，则说明当前连接是可靠的，继续进行读操作。失效的连接会从连接池中清除掉。
每个连接对于接收响应来说都以单独的线程运行，客户端可以通过同步(wait,notify)方式或者异步进行 rpc 调用，
序列化采用更高效的 hession 序列化方式。
服务端采用事件驱动的 NIO 的 MINA 框架，支撑高并发高吞吐量的请求。



七、协议

协议接口，HTTP、JSON

八、消息队列

对于平台各个系统之间的异步交互，是通过 MQ 组件进行的。

在设计消息服务组件时，需要考虑消息一致性、持久化、可用性、以及完善的监控体系。

业界开源的消息中间件主要 RabbitMQ、kafka 有两种，

RabbitMQ,遵循 AMQP 协议，由内在高并发的 erlanng 语言开发;kafka 是 Linkedin 于 2010 年 12 月份开源的消息发布订阅系统,它主要用于处理活跃的流式数据,大数据量的数据处理上。

对消息一致性要求比较高的场合需要有应答确认机制，包括生产消息和消费消息的过程；不过因网络等原理导致的应答缺失，可能会导致消息的重复，这个可以在业务层次根据幂等性进行判断过滤；RabbitMQ 采用的是这种方式。还有一种机制是消费端从 broker 拉取消息时带上 LSN 号，从 broker 中某个 LSN 点批量拉取消息，这样无须应答机制，kafka 分布式消息中间件就是这种方式。

消息的在 broker 中的存储，根据消息的可靠性的要求以及性能方面的综合衡量，可以在内存中，可以持久化到存储上。

对于可用性和高吞吐量的要求，集群和主备模式都可以在实际的场景应用的到。RabbitMQ 解决方案中有普通的集群和可用性更高的 mirror queue 方式。 kafka 采用 zookeeper 对集群中的 broker、consumer 进行管理，可以注册 topic 到 zookeeper 上;通过 zookeeper 的协调机制,producer 保存对应 topic 的 broker 信息，可以随机或者轮询发送到 broker 上；并且 producer 可以基于语义指定分片，消息发送到 broker 的某分片上。

总体来讲，RabbitMQ 用在实时的对可靠性要求比较高的消息传递上。kafka 主要用于处理活跃的流式数据,大数据量的数据处理上。

九、垃圾消息过滤

Mollom：实际上，每台服务器都能处理所有请求，但它们都有完整的故障切换机制。工作负载在多台机器之间分配。如果一台机器出了故障，那么工作负载会转移到另一台机器上。

十、安全问题

1.防火墙：
（1）包过滤型：根据包中信息的数据源地址、目标地址、TCP/UDP 源端口、目标端口判断是否来源于可信任的安全站点；
（2）网络地址转换-NAT：在内部网络通过安全网卡访问外部网络时，将产生一个映射记录。 外部网络通过非安全网络网卡与外部网络连接，这样对外就隐藏了真实的内部网络地址、外部网络通过非安全网卡访问内部网络时并不知道内部的网络连接状况，而只是通过一个开放的 IP 地址和端口来请求访问；
（3）代理型：位于客户机、服务器之间，完全阻挡了二者间的交流；
（4）监测型：对各层数据主动、实时监测；分析基础上也能有效的判断各层 的非法入侵。分布式的探测器安置在各种应用服务器和其他网络 的节点中，能检测网络外部的攻击和内部的恶意破坏。
2.应用网关
3.虚拟专用网络

引用：

1.高性能高可用的分布式架构体系

http://www.williamlong.info/archives/3664.html

2.构建高并发高可用的电商平台架构实践

http://blog.csdn.net/yangbutao/article/details/12242441/

3.《浅析分布式系统的安全性问题》

http://www.docin.com/p-62695971.html

4.《一个垃圾信息过滤系统的进化之道》

http://networking.ctocio.com.cn/96/11816596.shtml