# hbase在淘宝的应用和优化小结

作者：邓明鉴(taobao.com)　2012.3.5

# 1 前言

hbase是从hadoop中分离出来的apache顶级开源项目。由于它很好地用java实现了google的bigtable系统大部分特性，因此在数据量猛增的今天非常受到欢迎。对于淘宝而言，随着市场规模的扩大，产品与技术的发展，业务数据量越来越大，对海量数据的高效插入和读取变得越来越重要。由于淘宝拥有也许是国内最大的单一hadoop集群(云梯)，因此对hadoop系列的产品有比较深入的了解，也就自然希望使用hbase来做这样一种海量数据读写服务。Facebook曾经详细公布过内部使用hbase的情况，本文也将响应开源的号召，出于对社区的反馈及让更多的人了解hbase的实际应用，将hbase部署于生产环境，对淘宝近一年来在online应用上使用和优化hbase的经验做一次小结。

# 2 原因

为什么要使用hbase

淘宝在2011年之前所有的后端持久化存储基本上都是在mysql上进行的(不排除少量oracle/bdb/tair/mongdb等)，mysql由于开源，并且生态系统良好，本身拥有分库分表等多种解决方案，因此很长一段时间内都满足淘宝大量业务的需求。

但是由于业务的多样化发展，有越来越多的业务系统的需求开始发生了变化。一般来说有以下几类变化：

a) 数据量变得越来越多，事实上现在淘宝几乎任何一个与用户相关的在线业务的数据量都在亿级别，每日系统调用次数从亿到百亿都有，且历史数据不能轻易删除。这需要有一个海量分布式文件系统，能对TB级甚至PB级别的数据提供在线服务

b) 数据量的增长很快且不一定能准确预计，大多数应用系统从上线起在一段时间内数据量都呈很快的上升趋势，因此从成本的角度考虑对系统水平扩展能力有比较强烈的需求，且不希望存在单点制约

c) 只需要简单的kv读取，没有复杂的join等需求。但对系统的并发能力以及吞吐量、响应延时有非常高的需求，并且希望系统能够保持强一致性

d) 通常系统的写入非常频繁，尤其是大量系统依赖于实时的日志分析

e) 希望能够快速读取批量数据

f ) schema灵活多变，可能经常更新列属性或新增列

g) 希望能够方便使用，有良好且语义清晰的java接口

以上需求综合在一起，我们认为hbase是一种比较适合的选择。首先它的数据由hdfs天然地做了数据冗余，云梯三年的稳定运行，数据100%可靠已经证明了hdfs集群的安全性，以及服务于海量数据的能力。其次hbase本身的数据读写服务没有单点的限制，服务能力可以随服务器的增长而线性增长，达到几十上百台的规模。LSM-Tree模式的设计让hbase的写入性能非常良好，单次写入通常在1-3ms内即可响

应完成，且性能不随数据量的增长而下降。region（相当于数据库的分表）可以ms级动态的切分和移动，保证了负载均衡性。由于hbase上的数据模型是按rowkey排序存储的，而读取时会一次读取连续的整块数据做为cache，因此良好的rowkey设计可以让批量读取变得十分容易，甚至只需要 1 次io就能获取几十上百条用户想要的数据。最后，淘宝大部分工程师是java背景的同学，因此hbase的api对于他们来说非常容易上手，培训成本相对较低。

当然也必须指出，在大数据量的背景下银弹是不存在的，hbase本身也有不适合的场景。比如，索引只支持主索引（或看成主组合索引），又比如服务是单点的，单台机器宕机后在master恢复它期间它所负责的部分数据将无法服务等。这就要求在选型上需要对自己的应用系统有足够了解。

# 3 应用情况

我们从2011年3月开始研究hbase如何用于在线服务。尽管之前在一淘搜索中已经有了几十节点的离线服务。这是因为hbase早期版本的目标就是一个海量数据中的离线服务。2009年9月发布的0.20.0版本是一个里程碑，online应用正式成为了hbase的目标，为此hbase引入了zookeeper来做为backupmaster以及regionserver的管理。2011年1月0.90.0版本是另一个里程碑，基本上我们今天看到的各大网站，如facebook/ebay/yahoo内所使用于生产的hbase都是基于这一个版本(fb所采用的0.89版本结构与0.90.x相近)。bloomfilter等诸多属性加入了进来，性能也有极大提升。基于此，淘宝也选用了0.90.x分支作为线上版本的基础。

第一个上线的应用是数据魔方中的prom。prom原先是基于redis构建的，因为数据量持续增大以及需求的变化，因此我们用hbase重构了它的存储层。准确的说prom更适合0.92版本的hbase，因为它不仅需要高速的在线读写，更需要count/group by等复杂应用。但由于当时0.92版本尚未成熟，因此我们自己单独实现了coprocessor。prom的数据导入是来源于云梯，因此我们每天晚上花半个小时将数据从云梯上写入hbase所在的hdfs，然后在web层做了一个client转发。经过一个月的数据比对，确认了速度比之redis并未有明显下降，以及数据的准确性，因此得以顺利上线。

第二个上线的应用是TimeTunnel，TimeTunnel是一个高效的、可靠的、可扩展的实时数据传输平台，广泛应用于实时日志收集、数据实时监控、广告效果实时反馈、数据库实时同步等领域。它与prom相比的特点是增加了在线写。动态的数据增加使hbase上compact/balance/split/recovery等诸多特性受到了极大的挑战。TT的写入量大约一天20TB，读的量约为此的1.5倍，我们为此准备了20台regionserver的集群，当然底层的hdfs是公用的，数量更为庞大（下文会提到）。每天TT会为不同的业务在hbase上建不同的表，然后往该表上写入数据，即使我们将region的大小上限设为1GB，最大的几个业务也会达到数千个region这样的规模，可以说每一分钟都会有数次split。在TT的上线过程中，我们修复了hbase很多关于split方面的bug，有好几个commit到了hbase社区，同时也将社区一些最新的patch打在了我们的版本上。split相关的bug应该说是hbase中会导致数据丢失最大的风险之一，这一点对于每个想使用hbase的开发者来说必须牢记。hbase由于采用了LSM-Tree模型，从架构原理上来说数据几乎没有丢失的可能，但是在实际使用中不小心谨慎就有丢

失风险。原因后面会单独强调。TT在预发过程中我们分别因为Meta表损坏以及split方面的bug曾经丢失过数据，因此也单独写了meta表恢复工具，确保今后不发生类似问题(hbase-0.90.5以后的版本都增加了类似工具)。另外，由于我们存放TT的机房并不稳定，发生过很多次宕机事故，甚至发生过假死现象。因此我们也着手修改了一些patch，以提高宕机恢复时间，以及增强了监控的强度。

CTU以及会员中心项目是两个对在线要求比较高的项目，在这两个项目中我们特别对hbase的慢响应问题进行了研究。hbase的慢响应现在一般归纳为四类原因：网络原因、gc问题、命中率以及client的反序列化问题。我们现在对它们做了一些解决方案(后面会有介绍)，以更好地对慢响应有控制力。

和Facebook类似，我们也使用了hbase做为实时计算类项目的存储层。目前对内部已经上线了部分实时项目，比如实时页面点击系统，galaxy实时交易推荐以及直播间等内部项目，用户则是散布到公司内各部门的运营小二们。与facebook的puma不同的是淘宝使用了多种方式做实时计算层，比如galaxy是使用类似affa的actor模式处理交易数据，同时关联商品表等维度表计算排行(TopN)，而实时页面点击系统则是基于twitter开源的storm进行开发，后台通过TT获取实时的日志数据，计算流将中间结果以及动态维表持久化到hbase上，比如我们将rowkey设计为url+userid，并读出实时的数据，从而实现实时计算各个维度上的uv。

最后要特别提一下历史交易订单项目。这个项目实际上也是一个重构项目，目的是从以前的方案上迁移到hbase上来。由于它关系到已买到页面，用户使用频率非常高，重要程度接近核心应用，对数据丢失以及服务中断是零容忍。它对compact做了优化，避免大数据量的compact在服务时间内发生。新增了定制的filter来实现分页查询，rowkey上对应用进行了巧妙的设计以避免了冗余数据的传输以及90%以上的读转化成了顺序读。目前该集群存储了超过百亿的订单数据以及数千亿的索引数据，线上故障率为0。

随着业务的发展，目前我们定制的hbase集群已经应用到了线上超过二十个应用，数百台服务器上。总在线数据量接近100TB，线上每秒操作数在10万级别。包括淘宝首页的商品实时推荐、广泛用于卖家的实时量子统计等应用，并且还有继续增多以及向核心应用靠近的趋势。

# 4 部署、运维和监控

Facebook之前曾经透露过Facebook的hbase架构，可以说是非常不错的。如他们将message服务的hbase集群按用户分为数个集群，每个集群100台服务器，拥有一台namenode以及分为 5 个机架，每个机架上一台zookeeper。可以说对于大数据量的服务这是一种优良的架构。对于淘宝来说，由于数据量远没有那么大，应用也没有那么核心，因此我们采用公用hdfs以及zookeeper集群的架构。每个hdfs集群尽量不超过100台规模（这是为了尽量限制namenode单点问题）。在其上架设数个hbase集群，每个集群一个master以及一个backupmaster。公用hdfs的好处是可以尽量减少compact的影响，以及均摊掉硬盘的成本，因为总有集群对磁盘空间要求高，也总有集群对磁盘空间要求低，混合在一起用从成本上是比较合算的。zookeeper集群公用，每个hbase集群在zk上分属不同的根节点。通过zk的权限机制来保证hbase集群的相互独立。zk的公用原因则仅仅是为了运维方便。

由于是在线应用，运维和监控就变得更加重要，由于之前的经验接近0，因此很难招到专门的hbase运维人员。我们的开发团队和运维团队从一开始就很重视该问题，很早就开始自行培养。以下讲一些我们的运维和监控经验。

我们定制的hbase很重要的一部分功能就是增加监控。hbase本身可以发送ganglia监控数据，只是监控项远远不够，并且ganglia的展示方式并不直观和突出。因此一方面我们在代码中侵入式地增加了很多监控点，比如compact/split/balance/flush队列以及各个阶段的耗时、读写各个阶段的响应时间、读写次数、region的open/close，以及具体到表和region级别的读写次数等等。仍然将它们通过socket的方式发送到ganglia中，ganglia会把它们记录到rrd文件中，rrd文件的特点是历史数据的精度会越来越低，因此我们自己编写程序从rrd中读出相应的数据并持久化到其它地方，然后自己用js实现了一套监控界面，将我们关心的数据以趋势图、饼图等各种方式重点汇总和显示出来，并且可以无精度损失地查看任意历史数据。在显示的同时会把部分非常重要的数据，如读写次数、响应时间等写入数据库，实现波动报警等自定义的报警。经过以上措施，保证了我们总是能先于用户发现集群的问题并及时修复。我们利用redis高效的排序算法实时地将每个region的读写次数进行排序，能够在高负载的情况下找到具体请求次数排名较高的那些region，并把它们移到空闲的regionserver上去。在高峰期我们能对上百台机器的数十万个region进行实时排序。

为了隔离应用的影响，我们在代码层面实现了可以检查不同client过来的连接，并且切断某些client的连接，以在发生故障时，将故障隔离在某个应用内部而不扩大化。mapreduce的应用也会控制在低峰期运行，比如在白天我们会关闭jobtracker等。

此外，为了保障服务从结果上的可用，我们也会定期跑读写测试、建表测试、hbck等命令。hbck是一个非常有用的工具，不过要注意它也是一个很重的操作，因此尽量减少hbck的调用次数，尽量不要并行运行hbck服务。在0.90.4以前的hbck会有一些机率使hbase宕机。另外为了确保hdfs的安全性，需要定期运行fsck等以检查hdfs的状态，如block的replica数量等。

我们会每天根踪所有线上服务器的日志，将错误日志全部找出来并且邮件给开发人员，以查明每一次error以上的问题原因和fix。直至错误降低为0。另外每一次的hbck结果如果有问题也会邮件给开发人员以处理掉。尽管并不是每一次error都会引发问题，甚至大部分error都只是分布式系统中的正常现象，但明白它们问题的原因是非常重要的。

# 5 测试与发布

因为是未知的系统，我们从一开始就非常注重测试。测试从一开始就分为性能测试和功能测试。性能测试主要是注意基准测试，分很多场景，比如不同混合读写比例，不同k/v大小，不同列族数，不同命中率，是否做presharding等等。每次运行都会持续数小时以得到准确的结果。因此我们写了一套自动化系统，从web上选择不同的场景，后台会自动将测试参数传到各台服务器上去执行。由于是测试分布式系统，因此client也必须是分布式的。

我们判断测试是否准确的依据是同一个场景跑多次，是否数据，以及运行曲线达到99%以上的重合度，这个工作非常烦琐，以至于消耗了很多时间，但后来的事实证明它非常有意义。因为我们对它建立了100%的信任，这非常重要，比如后期我们的改进哪怕只提高2%的性能也能被准确捕捉到，又比如某次代码修改使compact队列曲线有了一些起伏而被我们看到，从而找出了程序的bug，等等。

功能测试上则主要是接口测试和异常测试。接口测试一般作用不是很明显，因为hbase本身的单元测试已经使这部分被覆盖到了。但异常测试非常重要，我们绝大部分bug修改都是在异常测试中发现的，这帮助我们去掉了很多生产环境中可能存在的不稳定因素，我们也提交了十几个相应的patch到社区，并受到了重视和commit。分布式系统设计的难点和复杂度都在异常处理上，我们必须认为系统在通讯的任何时候都是不可靠的。某些难以复现的问题我们会通过查看代码大体定位到问题以后，在代码层面强行抛出异常来复现它。事实证明这非常有用。

为了方便和快速定位问题，我们设计了一套日志收集和处理的程序，以方便地从每台服务器上抓取相应的日志并按一定规律汇总。这非常重要，避免浪费大量的时间到登录不同的服务器以寻找一个bug的线索。

由于hbase社区在不停发展，以及线上或测试环境发现的新的bug，我们需要制定一套有规律的发布模式。它既要避免频繁的发布引起的不稳定，又要避免长期不发布导致生产版本离开发版本越来越远或是隐藏的bug爆发。我们强行规定每两周从内部trunk上release一个版本，该版本必须通过所有的测试包括回归测试，并且在release后在一个小型的集群上24小时不受甘扰不停地运行。每个月会有一次发布，发布时采用最新release的版本，并且将现有的集群按重要性分级发布，以确保重要应用不受新版本的潜在bug影响。事实证明自从我们引入这套发布机制后，由发布带来的不稳定因素大大下降了，并且线上版本也能保持不落后太多。

# 6 改进和优化

Facebook是一家非常值得尊敬的公司，他们毫无保留地对外公布了对hbase的所有改造，并且将他们内部实际使用的版本开源到了社区。facebook线上应用的一个重要特点是他们关闭了split，以降低split带来的风险。与facebook不同，淘宝的业务数据量相对没有如此庞大，并且由于应用类型非常丰富，我们并们并没有要求用户强行选择关闭split，而是尽量去修改split中可能存在的bug。到目前为止，虽然我们并不能说完全解决了这个问题，但是从0.90.2中暴露出来的诸多跟split以及宕机相关的可能引发的bug我们的测试环境上已经被修复到接近了0，也为社区提交了10数个稳定性相关的patch，比较重要的有以下几个：

https://issues.apache.org/jira/browse/HBASE-4562
https://issues.apache.org/jira/browse/HBASE-4563
https://issues.apache.org/jira/browse/HBASE-5152
https://issues.apache.org/jira/browse/HBASE-5100
https://issues.apache.org/jira/browse/HBASE-4880
https://issues.apache.org/jira/browse/HBASE-4878
https://issues.apache.org/jira/browse/HBASE-4899

还有其它一些，我们主要将patch提交到0.92版本，社区会有commitor帮助我们backport回0.90版本。所以社区从0.90.2一直到0.90.6一共发布了5个bugfix版本后，0.90.6版本其实已经比较稳定了。建议生产环境可以考虑这个版本。

split这是一个很重的事务，它有一个严重的问题就是会修改meta表（当然宕机恢复时也有这个问题）。如果在此期间发生异常，很有可能meta表、rs内存、master内存以及hdfs上的文件会发生不一致，导致之后region重新分配时发生错误。其中一个错误就是有可能同一个region被两个以上的regionserver所服务，那么就可能出现这一个region所服务的数据会随机分别写到多台rs上，读取的时候也会分别读取，导致数据丢失。想要恢复原状，必须删除掉其中一个rs上的region，这就导致了不得不主动删掉数据，从而引发数据丢失。

前面说到慢响应的问题归纳为网络原因、gc问题、命中率以及client的反序列化问题。网络原因一般是网络不稳定引起的，不过也有可能是tcp参数设置问题，必须保证尽量减少包的延迟，如nodelay需要设置为true等，这些问题我们通过tcpdump等一系列工具专门定位过，证明tcp参数对包的组装确实会造成慢连接。gc要根据应用的类型来，一般在读比较多的应用中新生代不能设置得太小。命中率极大影响了响应的时间，我们会尽量将version数设为 1 以增加缓存的容量，良好的balance也能帮助充分应用好每台机器的命中率。我们为此设计了表级别的balance。

由于hbase服务是单点的，即宕机一台，则该台机器所服务的数据在恢复前是无法读写的。宕机恢复速度决定了我们服务的可用率。为此主要做了几点优化。首先是将zk的宕机发现时间尽量缩短到1分钟，其次改进了master恢复日志为并行恢复，大大提高了master恢复日志的速度，然后我们修改了openhandler中可能出现的一些超时异常，以及死锁，去掉了日志中可能发生的open…too long等异常。原生的hbase在宕机恢复时有可能发生10几分钟甚至半小时无法重启的问题已经被修复掉了。另外，hdfs层面我们将socket.timeout时间以及重试时间也缩短了，以降低datanode宕机引起的长时间block现象。

hbase本身读写层面的优化我们目前并没有做太多的工作，唯一打的patch是region增加时写性能严重下降的问题。因为由于hbase本身良好的性能，我们通过大量测试找到了各种应用场景中比较优良的参数并应用于生产环境后，都基本满足需求。不过这是我们接下来的重要工作。

# 7 将来计划

我们目前维护着淘宝内基于社区0.90.x而定制的hbase版本。接下来除继续fix它的bug外，会维护基于0.92.x修改的版本。之所以这样，是因为0.92.x和0.90.x的兼容性并不是非常好，而且0.92.x修改掉的代码非常多，粗略统计会超过30%。0.92中有我们非常看重的一些特性。

- 0.92版本改进了hfile为hfileV2，v2版本的特点是将索引以及bloomfilter进行了大幅改造，以支持单个大hfile文件。现有的HFile在文件大到一定程度时，index会占用大量的内存，并且加载文件的速度会因此下降非常多。而如果HFile不增大的话，region就无法扩大，从而导致region数量非常多。这是我们想尽量避免的事。
- 0.92版本改进了通讯层协议，在通讯层中增加了length，这非常重要，它让我们可以写出nio的客户端，使反序列化不再成为影响client性能的地方。

- 0.92版本增加了coprocessor特性，这支持了少量想要在rs上进行count等的应用。
- 还有其它很多优化，比如改进了balance算法、改进了compact算法、改进了scan算法、compact变为CF级别、动态做ddl等等特性。

除了0.92版本外，0.94版本以及最新的trunk(0.96)也有很多不错的特性，0.94是一个性能优化版本。它做了很多革命性工作，比如去掉root表，比如HLog进行压缩，replication上支持多个slave集群，等等。

我们自己也有一些优化，比如自行实现的二级索引、backup策略等都会在内部版本上实现。

另外值得一提的是hdfs层面的优化也非常重要，hadoop-1.0.0以及cloudera-3u3的改进对hbase非常有帮助，比如本地化读、checksum的改进、datanode的keepalive设置、namenode的HA策略等。我们有一支优秀的hdfs团队来支持我们的hdfs层面工作，比如定位以及fix一些hdfs层面的bug,帮助提供一些hdfs上参数的建议，以及帮助实现namenode的HA等。最新的测试表明，3u3的checksum+本地化读可以将随机读性能提升至少一倍。

我们正在做的一件有意义的事是实时监控和调整regionserver的负载，能够动态地将负载不足的集群上的服务器挪到负载较高的集群中，而整个过程对用户完全透明。

总的来说，我们的策略是尽量和社区合作，以推动hbase在整个apache生态链以及业界的发展，使其能更稳定地部署到更多的应用中去，以降低使用门槛以及使用成本。