HBase架构中RPC客户端的通信性能优化

胡 波[1] 谭 良[1,2]

(四川师范大学计算机学院 成都 610101)[1] (中国科学院计算技术研究所 北京 100190)[2]

摘 要 HBase已成为大数据存储、分析和处理的关键部件,对其进行性能优化是当前产业界和学术界的一个研究热点。HBase架构包括多个子系统,子系统之间的通信采用远程过程调用(Remote Procedure Call,RPC)通信机制,但这些子系统的RPC客户端采用的是阻塞通信模式,这种模式在客户端数据请求密集的情况下会引起线程的阻塞,影响了子系统之间的通信效率,降低了HBase的性能。首先分析了HBaseRPC客户端与服务端的通信机制,然后提出了一种HBaseRPC客户端非阻塞的通信模型,并通过Java NIO技术实现。实验结果表明,该模型有效降低了阻塞模式对通信性能的影响,提高了HBaseRPC客户端的通信性能。

关键词 HBase,大数据,远程过程调用,非阻塞

中图法分类号 TP393 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.4.019

Optimization of Communication Performance of RPC Client in HBase Architecture

HU Bo[1] TAN Liang[1,2]

(College of Computer Science,Sichuan Normal University,Chengdu 610101,China)[1]

(Institute of Computing Technology,Chinese Academy of Sciences,Beijing 100190,China)[2]

Abstract HBase has become a key component of big data storage,analysis and processing. Its performance optimization is a hot research topic in nowaday industry and academia. HBase architecture consists of multiple subsystems,among which communication is realized via remote procedure call (RPC) communication mechanism. But the RPC client of subsystem adopts the blocking communication patterns,which will cause threads to block when there are intensive client data requests,thus affecting communication efficiency among subsystems,and reducing HBase performance. We first analyzed communication mechanism of HBase RPC client side and its server side. Then we proposed a nonblocking communication pattern of HBase RPC client side via Java NIO. Experiment results show that this communication pattern can minimize the effect of blocking patterns on communication performance,and improve communication performance of HBase RPC client.

Keywords HBase,Big data,RPC,Nonblocking

1 引言

随着电子商务、社交网络等互联网技术的快速发展,网络数据量正在以极快的速度增长,数据灵活多变,数据类型呈多样化。分析和利用这些庞大的数据资源,必须依赖有效的数据管理技术。传统的关系型数据库技术在数据处理、系统扩展性等方面遇到了巨大的障碍,无法胜任大规模数据存储的工作。与此同时,NoSQL(Not Only SQL)[1]技术迅速发展,因其具有非关系型、分布式、开源及水平扩展等特性,是解决大数据存储和分析问题的有效方案。

HBase[2]是一种基于列存储的NoSQL数据库,它是Apache开源组织模仿Google的BigTabel特性,利用Java编程实现的开源数据库,它主要用于HDFS(Hadoop Distributed File System)文件系统上,为Hadoop提供类似BigTable的数据库服务,能够实时地对大数据进行随机访问和存储操作。HBase凭借对大数据存储的良好特性和优势,在产业界得到了广泛应用,如Facebook、阿里巴巴、京东、小米等。HBase由多个软件子系统组成,主要包括客户端、HMaster、HRegionServer、Zookeeper等,这些子系统共同组成一个分布式应用系统。随着当前网络数据量的飞速增加,基于HBase的应用系统处理数据、管理数据的任务越来越繁忙,如何提高HBase的系统性能便成了一个研究热点[5-15]。值得注意的是,HBase数据量的增加以及密集的I/O请求,使得其子系统间的通信会越来越频繁,因此提高HBase各子系统间的通信性能就成为了优化HBase的重要途径。

HBase各子系统之间的通信采用了定制HBase RPC,且0.96版本后引入了Google Protocol Buffers作为数据交换格式,并在Protocol Buffers提供的RPC接口之上实现了基于

到稿日期:2015-03-13 返修日期:2015-06-29 本文受国家自然科学基金(61373162),四川省科技厅科技支撑项目(2014GZ007),四川省可视化计算与虚拟现实项目(KJ201402)资助。

胡 波(1989—),男,硕士生,主要研究方向为云计算、大数据处理,E-mail:hubo609@foxmail.com;谭 良(1972—),男,博士,教授,CCF高级会员,主要研究方向为可信计算、云安全、大数据处理,E-mail:tanliang2008cn@126.com。

服务的 RPC。在 HBase 中，RPC 服务端采用非阻塞编程技术实现对客户端请求的处理，而客户端使用阻塞编程技术对服务端发送请求。客户端的这种实现方式，在密集的数据请求下会影响 HBase 的整体性能，很难满足客户端与服务端之间的数据交互质量要求。例如客户端需要发送大量的写数据请求，如果发送的请求遇到网络问题或是序列化时间较长等，客户端线程便会阻塞直到收到响应才会进行下一次调用，这在一定程度上降低了客户端的请求效率，影响了系统的通信性能。阻塞式 IO 虽然可以通过多线程的机制减小对处理性能的影响，但也存在局限性：1）随着线程的增多，Java 虚拟机会为每个线程分配独立的堆栈空间，系统开销大，增加了节点的负担；2）Java 虚拟机需要频繁地转让 CPU 的使用权，在进程切换间浪费了大量时间[3]。因此，改进客户端的通信方式是提高 HBase 系统通信性能进而提高 HBase 整体性能的重要途径。

本文针对这一问题提出了改进 HBase RPC 客户端处理模型，以非阻塞式通信机制代替阻塞式，提高了 RPC 客户端的通信性能。第 2 节介绍相关工作；第 3 节通过分析 HBase 的系统架构及其 RPC 通信模型指出客户端阻塞对系统性能造成的影响；第 4 节提出 HBaseRPC 通信模型的改进模型及实现；第 5 节通过实验对比改进前后系统的 RPC 性能，验证改进后 HBase 通信性能的提升。

2 相关工作

Google 成功实现了分布式存储系统 Bigtable[4]，为谷歌的大数据管理提供了优秀的解决方案。HBase 参照 Bigtable，实现了 Bigtable 的大部分特性。HBase 虽然与其存在一定差异，但作为一个开源的数据存储系统，得到了学术界和产业界的广泛关注，许多 HBase 的优化方案被提出。

在学术界，文献[5]提出一种 HBase 的扩展设计方案，其在保留 Java Socket 接口的基础上，使 HBase 客户端能够绕过 Java 本地接口利用远程直接内存访问（RDMA）网络访问服务端，性能评估显示 Get 操作时间大大减少。文献[6]指出了 Hadoop RPC 在发送和接收数据上存在的瓶颈，使用 RDMA 技术扩展了 Hadoop 通信方式，提出以一种绕过 JVM 堆内存的数据传输策略及基于历史的两级缓存池策略，很大程度上提高了 Hadoop RPC 的数据传输性能。文献[7]分析得出 HBase 的默认配置文件会引起性能的下降，提出了一种半自动化的配置管理器 Hconfig，实验表明 Hconfig 显著地提高了 HBase 的批量任务加载性能。针对现有的 HBase 提供的 MapReduce 访问接口存在数据读取速度较慢的问题，文献[8]提出以 HBase 的物理存储单元 Block 作为任务分配的基本单位，克服了原有接口无法保证数据本地性的问题，减少了网络开销，提高了访问速率。根据列存储数据库按列存储、列属性值相似度高这一依据，文献[9]提出了一种基于贝叶斯分类的压缩算法动态选择策略，即对压缩数据集中的不同数据块，根据数据的特点选择不同的压缩算法，从而提高了压缩率，同时加快了压缩/解压和查找数据的速度。文献[10]设计了以 Key 作为字典的压缩方法去压缩所有 Key，使用可变字节码去压缩 Value，利用倒排索引表的键作为字典的压缩方法的高即时性，有效提升了 HBase 的存储空间利用率。文献[11]指出 HBase 在存储非结构化的数据等大对象的过程中，Region 会频繁进行分片和合并，进而提出了一种定制的大数据对象存储方案，其在插入时减少了分片和合并次数，提高了 HBase 的大对象存储效率。

在产业界，Facbook 信息存储服务分析出了降低 HBase 在 HDFS 上的 I/O 性能的因素，通过增加缓存、减少读写次数以及绕过 HDFS 的复制层，减少了网络 I/O 的负载[12]。淘宝对 HBase 的系统中 Zookeeper 的宕机的发现与恢复效率进行了改进[13]，首先将 Zookeeper 的宕机发现时间尽量缩短到一分钟，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，解决了 HBase 在宕机恢复时有可能发生十几分钟甚至半小时无法重启的问题；同时在 HDFS 上缩短了 socket. timeout 时间以及重试时间，减少了 DataNode宕机引起的长时间阻塞现象。京东在 HBase 上实现了流量控制模块[14]，解决了平常几台甚至数十台的压力突然压在一台节点上可能引起的宕机问题；通过限流机制来控制一台节点上承受的流量和并发连接数，在实现流量控制的同时，保证了客户端不会超时且阻塞的请求不会占用大量 HBase 堆内存。小米云平台对 HBase 的 Hlog 的写模型和块索引键进行了改进[15]，改进后写吞吐性能的上限提高了3.4倍，节省了块索引的存储空间，间接提高了读性能。

总结起来，当前对 HBase 系统通信性能的优化工作还很少。因此本文提出一种基于非阻塞机制的 HBaseRPC 客户端通信模型，对其通信性能进行优化。

3 HBase 架构及其 RPC 通信机制分析

HBase 的数据存储架构[16]如图 1 所示，它由 HMaster 服务器（HBase Master Server）和 HRegion 服务器（HBase Region Server）构成，所有服务器由 Zookeeper 提供协调服务。HDFS 及其 DataNode 属于 Hadoop 平台，HBase 架构中包括 Client、HMaster、HRegionServer、Zookeeper 等子系统，各个子系统之间大多通过 RPC 来完成进程间通信。HBase 系统中的 RPC 是基于 Java Socket 的 TCP 机制实现的。

在图 1 所示的架构中，Client、HMaster 和 HRegionServer 之间实现了多个 HBaseRPC 协议[17]来完成管理和应用逻辑。在 Client 与 HMaster 之间，使用 HMasterMonitorProtocol 和 HMasterAdminProtocol，Client 为 RPC 客户端，HMaster 是 RPC 服务器端，通过前者实现客户端对集群状态的获取，后者主要实现了对用户表的管理。在 Client 与 HRegionServer 之间，使用 AdminProtocol 和 ClientProtocol，Client 为 RPC 客户端，HRegionServer 是 RPC 服务器端，通过前者实现对 region、WAL 的管理等，通过后者完成客户端数据的读写请求。在 HMaster 和 HRegionServer 之间，使用 HRegionServerStatusProtocol，HRegionServer 为 RPC 客户端，HMaster 是 RPC 服务器端，客户端 HRegionServer 向 HMaster 发送状态信息。特别是对于客户端的请求，假定客户端应用比较复杂，频繁的客户端请求在任务调度队列中一个接一个完成；或者是服务端返回的数据量过大，物理读写和传输延长了一次调用的时间；或者客户端一次远程调用因为服务端系统繁忙没有得到及时处理等。这几种情况都会使客户端线程一直处于等待状态，直到收到服务端的响应或是等待超时才会进行下一次调用。HBase 在内部采用 Google ProtocolBuffer 作为数据

组织格式完成各个子系统之间的 RPC 通信。客户端内实现了 Protocol Buffers 提供的 BlockingRpcChannel 接口方法 callBlockingMethod，被调用时会转换成 RpcClient 的 callBlockingMethod，Protocol Buffers 使用 MethodDescriptor 封装了不同 rpc 函数，使用 Message 基类可以接收基于 Message 的不同的 Request 和 Response 对象。

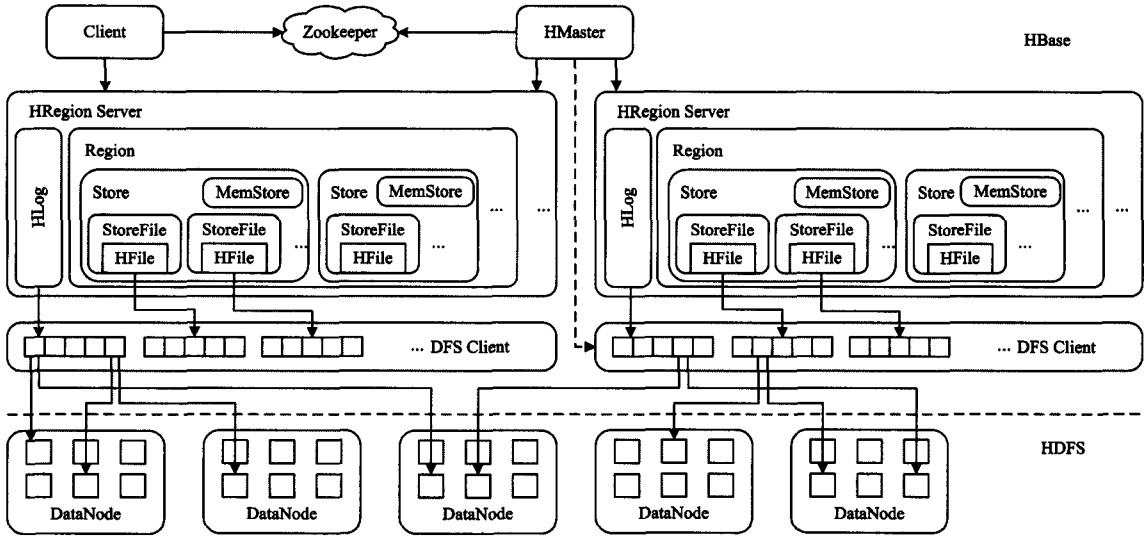

图 1　HBase 系统存储架构

当前 HBase Client 的 RPC 主要功能是发送远程过程调用信息并接收服务器响应。现有 HBase 客户端通信流程如图 2 所示。一个 Rpc 请求的函数调用过程如下：RPC 客户端首先会收到具体的请求，并将请求组成 Call 实例；RpcClient 创建一个 BlockingRpcChannel，实际内部的 BlockingChannelImplementation 调用 callBlockingMethod()，然后通过 getConnection()获取客户端与服务端的连接对象，若连接不存在，则通过 createConnection()新建一个 Connection 实例，获取连接后将 Call 实例放入自身的成员变量 calls 中并调用 writeRequest()向服务端发送请求，同时连接对象作为守护进程等待服务器的响应；客户端请求的数据通过 OutputStream 对象写入服务端，接着进行简单的 while 循环一直调用 wait(3000)等直到连接对象调用 Call 实例的 callComplete()唤醒阻塞的客户端线程或者是等待超时。最后根据结果类型对其进行处理并删除 calls 中对应的 Call 实例完成此次调用。

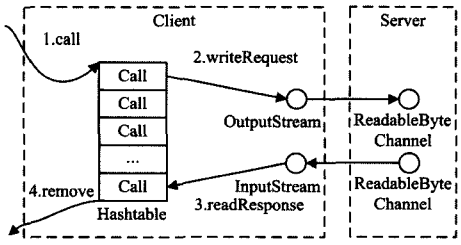

图 2　HBase RPC 客户端通信模型

对 HBase 的源码进行分析可以发现，客户端通过阻塞模式连接服务器，连接线程会发生阻塞直到连接成功或出现 I/O 错误。同时从图 2 可以看出，HBaseRPC 客户端中使用 OutputStream 类和 InputStream 类作为数据传输类，而在 OutputStream 上的写操作将会面临阻塞问题直到数据被成功写入，同样 InputStream 上的读操作也会阻塞，直到读完服务端响应的数据。由此可见，客户端的连接以及工作线程的 I/O 操作都可能进入阻塞状态。这些因素都导致了客户端资源的浪费，影响了系统的性能。在只有较少的客户端线程和服务端保留有连接的情况下，客户端阻塞的工作机制也无法充分利用服务端的处理能力。

HBase RPC 服务器端的主要功能是处理来自客户端的 RPC 请求，通过调用相应的函数将处理后的结果返回给客户端。图 3 是 RPC 服务端的通信流程，具体过程如下：服务端首先使用 Listener 作为连接监听守护进程，当有客户端连接请求时，选择器便会选择一个可连接的通道并为其分配读线程；然后 Reader 将读取到的数据重组为 Call，交由 RpcScheduler 进行处理并为其分配 RunCaller，RunCaller 负责执行 Call 并将处理后的结果放入响应队列中，最后由 Responder 将处理好的结果返回给客户端。

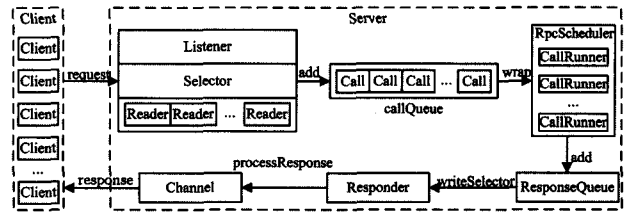

图 3　HBase RPC 服务端通信模型

HBase RPC 的服务端与客户端不同，服务端主要用 Java NIO(新 I/O)来实现。如果 Server 端也采用阻塞式通信，当连接进来的 Client 端很多时，一旦连接阻塞必然会影响 Server 端的性能。服务端采用了提高服务端并发处理能力的技术，包括线程池、基于事件驱动机制的 Reactor 设计模式。线程池能够有效地管理系统中线程的数量，用户可以根据系统的运行情况设置线程数以达到更好的运行性能，减少系统资源的浪费。事件驱动机制就是利用事件到的时刻触发，而不是同步地等待事件的发生，将线程调度交给另一个线程去完成，提高了线程间调度的效率，服务端利用 Java NIO 很好地实现了事件驱动机制，基于 Reactor 设计模式通过派发、分离 IO 操作事件提高了系统并发性能。Reactor 实现原理如图 4 所示。Reactor 是事件的派发者。Acceptor 接收事件处理请求，建立并向 Reactor 注册与此事件对应的 Handler。Handler 是与服务器通信的实体，按一定的过程处理业务流程，例如数据的解码、计算、编码等。read 和 write 是从 Handler 分离出的读事

件和写事件，由单独的读线程和写线程对相应的操作进行处理。

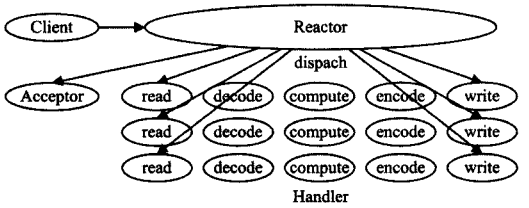

图 4　服务端基于事件驱动的 Reactor 设计模式

4　HBaseRPC 客户端通信性能的优化设计与实现

4.1　HBaseRPC 客户端优化模型

从上述的 HBase RPC 客户端和服务端的分析可以看出，如果将 RPC 客户端改进成类似服务端的非阻塞式通信，充分利用与服务器连接的通道，就能有效降低线程阻塞所引起的时间消耗，提高客户端的请求效率。基于这一思路，提出了 HBase RPC 客户端的优化模型，如图 5 所示。

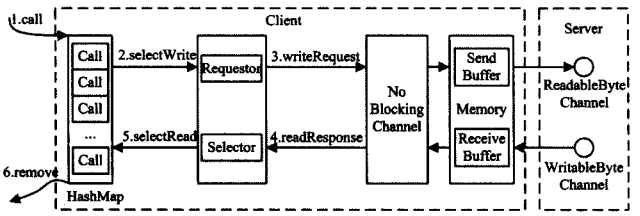

图 5　HBase RPC 客户端通信优化模型

图 5 中，采用了 JavaNIO 来实现客户端的非阻塞通信。非阻塞机制主要由 java. nio 包中的 SocketChannel、ByteBuffer、Selector、SelectionKey 等类实现，即去除客户端 OutputStream 和 InputStream 输入输出类、Socket 以及阻塞式的调用等，由缓冲区 Buffer 和支持非 SocketChanel 阻塞通信的 SocketChannel 以及非阻塞式调用等代替。SocketChannel 是 Socket 的代替类，能够支持阻塞通信和非阻塞通信，通过调用其方法 configureBlocking(false)将其设置为非阻塞模式便可支持非阻塞通信。数据的发送和接收都会经过 ByteBuffer，缓冲区在客户端一直被重用，减少了动态内存分配次数，同时减少了实际物理读写次数。Selector 负责监控 SocketChannel 上的读就绪和写就绪事件，后者向前者注册生成事件句柄如 OP_READ 和 OP_WRITE，一旦 Selctor 的事件集合大于零，说明事件发生并立刻执行相应的读或写处理。采用基于事件驱动的非阻塞机制，只要客户端写可操作便可将请求提交给服务端，等待服务端的响应。这种方式能够有效地提高请求效率，充分利用服务端的处理能力，提高客户端的数据请求效率。

HBase RPC 客户端通信性能优化模型的执行流程如下。首先，客户端采用非阻塞的方式连接服务端，如果连接建立成功，该方法返回 true；如果不能立即连接成功，便返回 false，稍后会重新进行连接，若连接仍然失败，则抛出相应的异常，说明客户端连接出现了问题，这种连接不会引发连接线程的阻塞，节约了系统资源。对于客户端请求操作，在请求 Call 实例构建成功之后，将其放入 LinkList 类型的 Call 队列中，同时请求者(Requestor)检查连接的通道是否可写，一旦通道可写，将请求对象发送至服务端。对于结果响应操作，Requestor 中的 Selector 会对请求的连接进行监听，若通道可写，说明有响应数据传送回来，根据返回结果的 callId，将结果给到对应的 Call 实例，这样一次请求操作完成。整个过程中 Requestor 会对连接通道进行轮询监听操作，只有在没有请求或者响应时才会发生阻塞，直到有请求数据和响应数据时工作，当连接失效时 Requestor 也会停止对该通道的监听。一个具体 RPC 请求的函数调用过程如下：RPC 客户端首先会收到具体的请求，并将请求组成 Call 对象；RpcClient 创建一个 NoBlockingRpcChannel，实际内部的 NoBlockingChannelImplementation 实现了调用 callNoBlockingMethod()，然后通过 getConnection()获取客户端与服务端的连接，若连接不存在，则通过 createConnection()新建一个 Connection 实例，并将请求 Call 实例放入其 Call 容器中，若通道写可操作，实例调用自己的 writeRequest()向服务端发送请求；服务端返回的数据在通道读可操作时，根据结果的 callId 返回给 Call 实例响应的请求，并删除已完成请求 Call 实例。原有的 OutputStream 和 InputStream 对象也由 ByteBuffer 类型的 SendBuffer 和 ReceiveBuffer 代替，传统的 IO 是面向流的处理，而新 IO 可以将一部分数据映射到内存中，面向块处理。从新的 RPC 通信流程可以看出，新的客户端避免了原有客户端中的连接阻塞和远程调用出现的一些阻塞问题，提高了客户端的通信性能。

4.2　HBaseRPC 客户端优化模型的实现

为了实现 HBase RPC 客户端通信性能优化模型，重新设计了 HBase RPC 客户端的通信类，类图如图 6 所示。

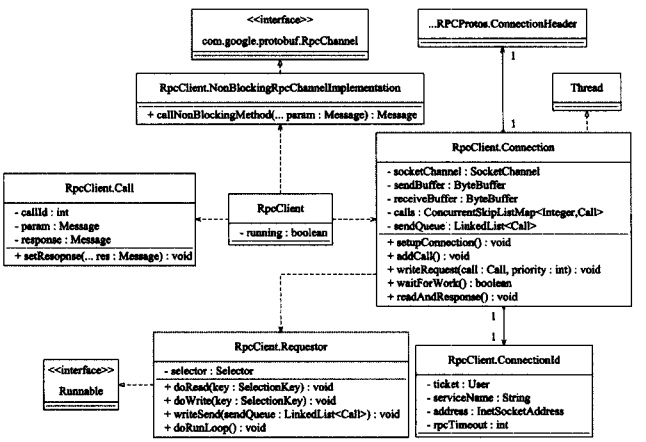

图 6　HBase RPC 客户端优化简要类图

图 6 是客户端实现的简要类结构。HBase RPC 客户端的核心代码位于 RpcClient 类中。最大的变化在于 RpcClient 的 Connection 类、Connection 负责与服务端的连接、Call 实例的管理，以及完成 Call 实例的读写请求；ConnectionHeader 中包含了用户信息和服务名称以及编码信息等，ConnectionId 是 Connection 的唯一标识，Call 用来生成调用对象的实例；请求者类 Requestor 主要负责以非阻塞方式发送和接收调用对象和响应。

Connection 中 SocketChannel 用来连接服务端以及传输数据，sendBuffer 与 receiveBuffer 变量用来缓存写数据和读数据；calls 用来保存当前正在处理的 Call 实例；setupConnection()方法以非阻塞的方式建立与服务端的连接；addCall()方法将正在处理的 Call 实例保存在 calls 中，在接收完一个请求的调用后会删除已经完成的 Call 实例；writeRequest()方法将写数据放入 sendBuffer 中；waitForWork()方法等待读数据，客户端处于工作状态并且有远程调用处理时以及 calls 非空时，返回 true 并继续处理，其他情况返回 false 进而关闭连

接；readResponse()方法读取响应的结果并进行 calls 中的 call 删除工作等后续处理；sendQueue 是与 Requestor 交互的发送队列，由 Requestor 完成此队列上 call 的发送。

NonBlockingRpcChannelImplementation 实现了 com.google.protobuf.RpcChannel 接口，通过实现 callMethod(MethodDescriptor md, RpcController controller, Message request, Message responsePrototype, RpcCallback〈Message〉done)完成非阻塞的客户端调用，callMethod()方法最终调用 connection 实例的 writeRequest()方法将数据写入 sendBuffer，一旦通道可写，由 Requestor 将数据发送给服务端。

Requestor 主要负责以非阻塞方式发送和接收调用对象和响应，connection 在远程过程调用前将 socketChannel 注册到 Requestor 的 selector 上，selector 对通道轮询监听，通道可写时将请求数据发送给服务端，通道可读时获取服务端的响应并将结果分发给 connection 完成调用。成员变量 selector 是选择器实例，该实例用来进行通道的注册，选择通道上 I/O 的读写操作；doRead()方法与 doWrite()方法分别读取服务端的响应数据和向服务端发送数据；writeSend()方法将一个连接上发送队列的 Call 实例发送到服务端进行处理，并最终调用通道的 write()方法完成数据的发送。doRunLoop()为 requestor 线程中 run()方法中的循环操作方法，只有等客户端退出 Requestor 才停止工作。Requestor 完成工作的主要算法如图 7 所示。

```
1. WHILE running==true
2. IF selector.select()==0
3. continue
4. END IF
5. iterator = selector.selectedKeys().iterator()
6. WHILE iterator.hasNext()
7. key = iterator.next()
8. iterator.remove()
9. IF key.isWritable()
10. doWrite(key)
11.        END IF
12.        IF key.isReadable()
13. doRead(key)
14.        END IF
15. END WHILE
16. END WHILE
```

图 7　非阻塞通信轮询算法

5　HBase RPC 客户端通信性能测试

实验在 HBase 分布式环境下对 RPC 客户端进行验证。HBase 部署在 6 台普通台式机上，硬件平台为 1 台 Intel(R) Core(TM)2 Duo CPU E7500 2.93GHz，4G 内存作为 HMaster；3 台 HRegionServer，1 台 Zookeeper，1 台测试 client 均为 Intel(R)Core(TM)i3-2130 3.40GHz，2GB 内存。软件平台为 CentOS6.5-32 位、jdk-1.7、hbase-0.98.5 和 ganglia-3.1.7。实验采用 Ganglia 对集群性能进行监控，从而得出 HMaster 和 HRegionServer 服务器的一系列指标。ProcessCallTime_mean 为服务器上执行 RPC 操作的平均时间，它反映了服务端执行 RPC 操作的性能；QueueCallTime_mean 为服务端执行 RPC 操作排队时间(即操作到达和操作执行时间的延迟)的平均时间，它反映了服务端的负载。

图 8 和图 9 是客户端对改进前的 HBase 的 HMaster 和 HRegionServer 进行写操作的 RPC 测试数据统计。从图中可以看出，客户端 Client 的主要 put 操作是在 HRegionServer3 上完成的，故服务端 RPC 执行的平均时间大于其他节点，队列中等待的平均时间也要大于其他节点，且上限为默认配置文件中的 10。

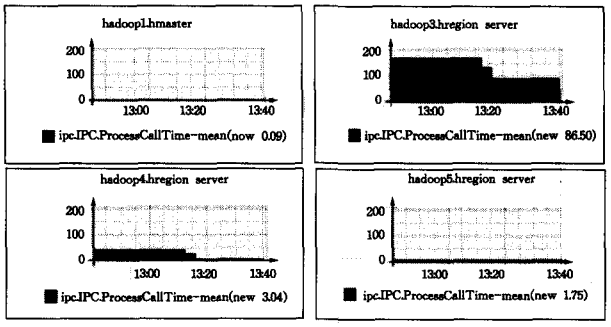

图 8　ipc.IPC.ProcessCallTime_mean

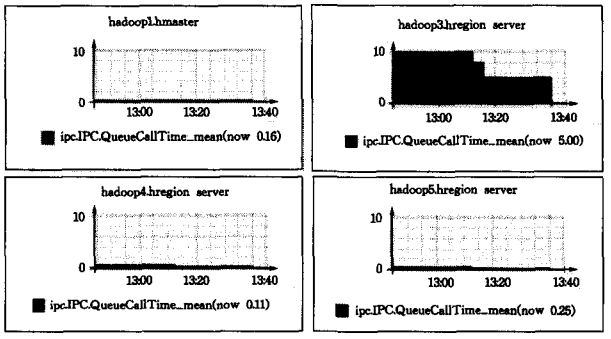

图 9　ipc.IPC.QueueCallTime_mean

在实验中发现 HBase 内部子系统内的 RPC 通信的并不频繁，改进后的 HBase RPC 客户端对 HBase 系统内部的通信并没有很明显的影响，HBase 读写数据请求主要发生在客户端与 HRegionServer 之间。因此对 HBase 客户端的读写操作进行对比测试，实验对大小为 4.6M 的 docx 文件进行了 put 和 get 操作。

put 操作和 get 操作在阻塞模式和非阻塞模式下的时间对比如图 10 和图 11 所示。实验在相同情况下对 4.6M 的 docx 文档分别进行了写和读操作，在 HBase 客户端请求密集的情况下，非阻塞模式下的操作的总时间消耗要小于阻塞模式。在客户端请求并不密集时，HBaseRPC 阻塞式客户端不会对 HBase 通信性能造成明显的影响，而非阻塞模式需要新的线程来分担 CPU 的工作，因此阻塞模式内存负载略低于非阻塞式。虽然非阻塞模式需要 HBase 客户端损耗一定性能来保证 Requestor 线程对连接通道进行轮询以及数据 I/O 操作，但是为了提高 HBase 在密集请求下的通信性能，这种性能损耗是可以接受的。

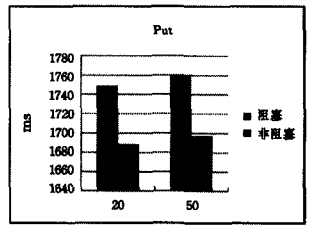

图 10　put 操作在阻塞模式和非阻塞模式下的时间对比

(下转第 110 页)

[10] Carzaniga A, Rosenblum D S, Wolf A L. Design and evaluation of a wide-area event notification service [J]. ACM Transactions on Computer Systems, 2001, 19(3): 332-383

[11] Altinel M, Franklin M J. Efficient Filtering of XML Documents for Selective Dissemination of Information[C]// 26th International Conference on Very Large Data Bases. 2000: 53-64

[12] Diao Y, Fischer P, Franklin M J, et al. Yfilter: Efficient and scalable filtering of XML documents[C]// 18th International Conference on Data Engineering. IEEE, 2002: 341-342

[13] Zhang Cheng-wen, Su Sen, Chen Jun-liang. Genetic Algorithm on Web Services Selection Supporting QoS [J]. Chinese Journal of Computers, 2006, 29(7): 1029-1037(in Chinese)
张成文,苏森,陈俊亮. 基于遗传算法的 QoS 感知的 Web 服务选择[J]. 计算机学报, 2006, 29(7): 1029-1037

[14] Ma You, Wang Shang-guang, Sun Qi-bo, et al. Web Service Quality Metric Algorithm Employing Objective and Subjective Weight[J]. Journal of Software, 2014, 25(11): 2473-2485(in Chinese)
马友,王尚广,孙其博,等. 一种综合考虑主客观权重的 Web 服务 QoS 度量算法[J]. 软件学报, 2014, 25(11): 2473-2485

[15] Kritikos K, Plexousakis D. Requirements for QoS-Based Web Service Description and Discovery[J]. IEEE Transactions Service Computing, 2009, 2(4): 320-337

[16] Al-Masri E, Mahmoud Q H. Discovering the best Web service [C]// 16th International Conference on World Wide Web. Alberta, CANADA, 2007: 1257-1258

(上接第 101 页)

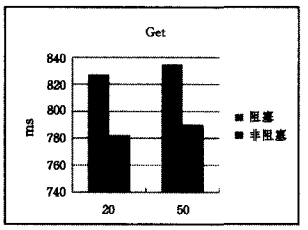

图 11 get 操作在阻塞模式和非阻塞模式下的时间对比

结束语 远程过程调用是 HBase 架构中重要的底层通信框架,随着当前数据服务量的快速增长,客户端的请求越来越频繁,阻塞通信机制已成为影响 HBase 的性能的重要因素。本文分析 HBase RPC 客户端和服务端的通信机制,HBase RPC 客户端的阻塞会导致数据传输性能的下降,为此提出了非阻塞模式的通信模型,并通过 Java NIO 实现了 HBase RPC 非阻塞式客户端。实验分析表明,在大数量及密集数据读写的环境下,HBase RPC 客户端的非阻塞式通信要优于阻塞式通信。

参 考 文 献

[1] NoSQL Database[EB/OL]. 2014. 8. http://nosql-database.org

[2] Apache HBase[EB/OL]. 2014. 8. http://hbase.apache.org

[3] Sun Wei-qin. Java Network Programming [M]. Beijing: Publishing House of Electronics Industry, 2007: 82-86(in Chinese)
孙卫琴. Java 网络编程精解[M]. 北京:电子工业出版社, 2007: 82-86

[4] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[C]//Proceedings of the Seventh Symposium on Operating System Desgin and Implementation. 2006

[5] Huang Jian, Ouyang Xiang-yong, Jose J, et al. High-Performance Design of HBase with RDMA over InfiniBand[C]// IEEE International Parallel and Distributed Processing Symposium. IEEE, 2012: 774-785

[6] Lu Xiao-yi, Islam N S, Wasi-ur-Rahman M, et al. High-Performance Design of Hadoop RPC with RDMA over InfiniBand[C]// International Conference on Parallel Processing. IEEE, 2013: 641-650

[7] Bao Xian-qiang, Liu Ling, Xiao Nong, et al. HConfig: Resource adaptive fast bulk loading in HBase[C]// International Conference on Collaborative Computing: Networking, Applications and Worksharing. IEEE, 2014: 215-224

[8] Tian Sheng-li, Xu Xi-shan, Yang Shu-qiang, et al. Optimization for the Access Interface of MapReduce in HBase[C]// Collections of Ninth Annual Academic Conference of China Institute of Communications. China Institute of Communications, 2012(in Chinese)
田胜利,徐锡山,杨树强,等. 针对 HBase 的 MapReduce 访问接口的优化[C]//第九届中国通信学会学术年会论文集. 中国通信学会, 2012

[9] Luo Yan-xin. Research and Implementation on HBASE Based Column-Oriented Compression Algorithms [D]. Guangzhou: South China University of Technology, 2011(in Chinese)
罗燕新. 基于 HBASE 的列存储压缩算法的研究与实现[D]. 广州:华南理工大学, 2011

[10] Cheng Peng-sen, An Jun-xiu. The key as dictionary compression method of inverted index table under the HBase database[J]. Journal of Software, 2013, 8(5): 1086-1093

[11] Kang Yi. The Design and Implementation of HBase Large Object Storage[D]. Nanjing: Nanjing University, 2013(in Chinese)
康毅. HBase 大对象存储方案的设计与实现[D]. 南京:南京大学, 2013

[12] Harter T, Borthankur D, Dong Si-ying, et al. Analysis of HDFS Under HBase: A Facebook Messages Case Study[C]// Proceedings of the 12th USENIX Conference on File and Storage Technologies. USENIX, 2014: 199-212

[13] A Summary of Application and Optimization of HBase in Taobao[EB/OL]. 2014. 8. http://blog.nosqlfan.com/html/3694.html(in Chinese)
HBase 在淘宝的应用和优化小结[EB/OL]. 2014. 8. http://blog.nosqlfan.com/html/3694.html

[14] Perfection and Innovation of HBase in Jingdong[EB/OL]. 2014. 8. http://www.sootoo.com/content/455783.shtml(in Chinese)
HBase 在京东的完善与创新[EB/OL]. 2014. 8. http://www.sootoo.com/content/455783.shtml

[15] Liu Shao-hui. HBase Used in Xiaomi [R]. China Hadoop Summit. 2013(in Chinese).
刘绍辉. 小米 Hbase 实践[R]. 中国 Hadoop 技术峰会, 2013

[16] Geoge L. HBase 权威指南[M]. 代志远,刘佳,蒋杰,译. 北京:人民邮电出版社, 2013: 302-304

[17] Hbase Hmaster Architecture[EB/OL]. 2014-8. http://blog.zahoor.in/2012/08/hbase-hmaster-architecture/