doi:103969/j. issn. 0490-6756. 2014. 04. 012

DCQD:一种物联网高性能数据采集平台的设计与实现

黄 健[1],冯 暄[2],翁 凯[3],卢 军[1],郭本俊[1]
(1. 成都信息工程学院软件工程学院,成都 610225;2. 四川省计算机研究院,成都 610041;
3. 四川省旅游信息中心,成都 610021)

摘 要:在物联网环境中,大量的传感器产生了海量的数据. 这些数据通常都需要写入到数据库中来实现数据的分析与应用. 当这些物联网海量传感器数据插入到数据库中的时候,会在存储系统中产生严重的小数据同步写性能瓶颈. 针对此问题,本文设计了一种高性能数据库磁盘缓冲队列 DCQD(Disk Cache Queue for Database). DCQD 在保证物联网采集数据同步写入磁盘,确保不丢失数据的基础上,可以显著优化海量数据插入到数据库中的性能. 实验表明,DCQD 在物联网应用环境中,可以显著地提高数据采集系统的性能.

关键词:物联网;数据采集;海量;高性能;队列
中图分类号:TP393 **文献标识码**:A **文章编号**:0490-6756(2014)04-0707-06

DCQD: Design and imple mentation of a high-performance buffering queue for database disks within the context of massive data acquisition of Internet of Things

HUANG Jian[1], *FENG Xuan*[2], *WENG Kai*[3], *LU Jun*[1], *GUO Ben-Jun*[1]
(1. Software Engineering College, Chengdu University of Information Technology, Chengdu 610225, China;
2. Sichuan Institute of Computer Science, Chengdu 610041, China;
3. Sichuan Provincial Tourism Information Center, Chengdu 610021, China)

Abstract: In the environment of Internet of Things, a large number of sensors generate huge amounts of data. Usually the generated data needs to be stored in the database to support data analyses and various applications. With respect to insert these huge amounts of sensor data into the database, it will lead to serious performance bottleneck of synchronous writing fragment data in the corresponding storage system. To solve this problem, this paper designs a high-performance buffering queue for database disks named as DCQD (Disk Cache Queue for Database). To ensure the collected data that can be synchronously wrote into disk without any data lose, DCQD can significantly optimize the performance on inserting mass data into the database. Experiments show that DCQD can significantly improve the performance of data acquisition system within the environment of Internet of Things.

Key words: Internet of Things; Data acquisition; Mass; High performance; Queue

收稿日期:2013-09-25
基金项目:国家自然科学基金(61203172)
作者简介:黄健(1976—),男,四川成都人,讲师,硕士,研究方向为移动计算. E-mail: huangjian@cuit. edu. cn

1 引 言

随着物联网应用的快速发展，成千上万的传感器通过网络连接起来. 这些传感器每时每刻都在产生数据，这些数据汇总起来形成了海量的物联网传感器数据. 通常，这些数据都需要通过物联网数据采集系统存入到数据库中，供后续的数据分析、设备控制与信息处理使用. 在上述物联网数据采集应用中，一个突出的性能瓶颈是海量的传感器数据同步写入到数据库中，会导致数据库所在的存储系统产生严重的小数据同步写性能，严重地降低整个物联网数据采集系统性能[1]. 这是由于，在现代计算机系统中，磁盘仍然是最主要的存储系统. 由于磁盘的机械动作特性，必须通过寻道、旋转才能读写数据. 因此，海量小数据同步写将导致存储系统中巨量的磁盘寻道和旋转操作，极大地降低存储系统的性能[2].

提高小数据同步写性能的典型方法是将写入数据首先缓存起来，然后将多个小数据整合成大数据后，采用优化的方式再写入到磁盘中[3]. 文献[4]提出了一种基于NVRAM的高可用的磁盘阵列缓冲区[5]，可以实现数据先缓存，然后再优化写入磁盘. 文献[4]提出的方法适用于突发式磁盘写情况下的性能优化. 因为，由价格昂贵的NVRAM构成的缓冲区，其容量不可能太大. 而缓冲区一旦满了，系统将丧失性能优化能力. 在物联网数据采集系统中，传感器数据的生成与写入数据库是持续稳定的，而非突发式. 因此，文献[4]的方法不适用于物联网数据采集系统应用环境.

文献[6]提出了一种高性能的磁盘队列存储管理机制FastQueue. FastQueue充分利用了大容量的磁盘缓冲队列来提高消息转发服务器的性能，具有很大的实用价值. 但是，FastQueue重点是提高消息转发服务器的性能，没有涉及消息同步写入数据库模式的分析. 此外，FastQueue并没有提供消息同步写入磁盘的能力，这使得一旦系统发生意外崩溃，FastQueue将丢失采集数据. 因此，FastQueue不适宜应用在物联网数据采集系统中.

本文设计了一种面向物联网海量数据采集的高性能数据库磁盘缓冲队列DCQD(Disk Cache Queue for Database)，充分利用了磁盘缓冲队列的高速特征，在数据库存储系统负荷较高的时候，启动磁盘缓冲队列机制，在确保数据同步写入磁盘的基础上，优化数据插入到数据库中的性能，显著地提高了物联网数据采集系统的性能.

本文的内容组织如下：第二节介绍了DCQD的总体框架；第三节介绍了DCQD的设计；第四节介绍了DCQD的具体应用方式；第五节给出性能对比实验结果；第六节对全文进行了总结.

2 DCQD总体框架

通常物联网数据采集系统的结构如图1所示，数据采集进程通过网络读取传感器的数据，然后写入到数据库中. 物联网中的传感器产生的数据通常不大，但是数量巨大. 所以，数据采集进程会将采集到的大量传感器产生的小数据写入到数据库中，从而在数据库的存储系统中出现大量的小数据同步写操作，导致存储系统的严重性能瓶颈.

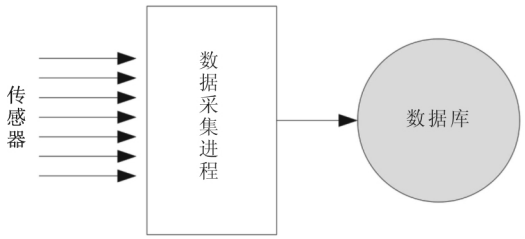

图1 物联网数据采集模型

Fig. 1 Data acquisition model of Internet of things

在物联网数据采集系统应用环境中，传感器数据一般需要同步写入磁盘，而不能在采集后暂时将数据存放在内存中，等待以后再写入到磁盘中. 原因是，如果采集数据仅仅是保存在内存中，一旦物联网数据采集系统发生意外崩溃，在内存中尚未写入磁盘的传感器数据就会丢失. 因此，在物联网数据采集系统中，数据采集进程在接收到一个传感器数据的时候，必须使用数据库的INSERT语句将数据同步写入到数据库中. 数据库接收到写入数据后，会将数据使用数据库的事务处理方式同步写入到磁盘上，从而保证数据不会丢失. 这些大量的小数据同步写操作，会导致数据库存储系统中的磁盘频繁地寻道，从而使存储系统的性能变得非常低[7].

为了解决这个问题，本文在DCQD中引入了高性能磁盘缓冲队列机制，既保证了数据被及时地同步写入磁盘，同时又优化了数据库大量INSERT数据的性能，从而提高了物联网数据采集

系统的性能.

DCQD 的总体框架如图 2 所示.

在 DCQD 中，为了提高小数据同步写的性能，采用了高性能的磁盘缓冲队列. 如图 2 所示，数据采集进程采集到传感器数据后，首先探测数据库存储系统的运行负荷，在数据库运行负荷较小的时候，数据采集进程直接将采集数据写入数据库中；当数据库负荷较大的时候，数据采集进程将采集数据写入到磁盘缓冲队列中，然后再由队列清洗进程同时采用优化方式将这些数据写入到数据库中.

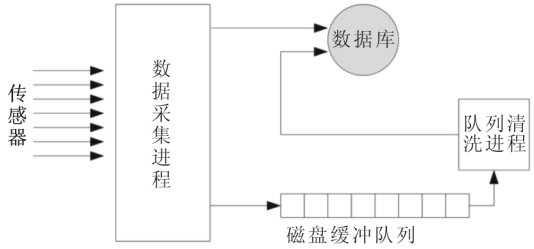

图 2　DCQD 的总体框架

Fig. 2　The overall frame of DCQD

通过 DCQD 的这种运行方式，在数据库存储系统负荷低的时候，DCQD 将采集数据直接写入数据库，避免了时延；当数据库存储系统负荷高的时候，DCQD 使用磁盘缓冲队列来实现采集数据的优化，并写入数据库，降低了数据库存储系统的负荷，从而提高整个物联网数据采集系统的性能.

3　DCQD 的设计

3.1　磁盘缓冲队列的数据存放方式

物联网数据采集系统中，传感器生成的数据一般都比较小，数据报文的长度通常为几十个字节到几百个字节. 一般而言，在一个物联网数据采集系统中，传感器的种类是多种多样的. 因此，物联网数据采集系统中各种各样的传感器所生成的数据报文的长度通常也是不一样的. 为了提高 DCQD 处理数据报文的效率，设计了几种不同大小的磁盘缓冲队列，来存放不同大小的数据报文.

在 DCQD 中，设置了 64 字节、128 字节、256 字节三种大小磁盘缓冲队列. DCQD 采集的数据报文中，小于或等于 64 字节的数据报文存放在 64 字节磁盘缓冲队列中；大于 64 字节，且小于或等于 128 字节的数据报文存放在 128 字节磁盘缓冲队列中；其它大小的报文以此类推. 在 DCQD 中，大于 256 字节的数据报文通常较少，如果在 DCQD 中采集到了大于 256 字节的数据报文，那么将使用多个连续的 256 字节队列存储空间来存放.

在 DCQD 中，每个采集到的数据报文，都会有一个数据报文索引部分 Message_Index 和一个数据报文数据部分 Message_Data. Message_Index 中记录了数据报文的 ID、采集时间、报文存放位置. Message_Data 中记录了数据报文的实际数据.

在 256 字节磁盘缓冲队列中，由于可能存在使用多个数据报文空间存放一个大于 256 字节数据报文的情况，因此在 256 字节磁盘缓冲队列的 Message_Index 中，增加了报文长度字段. 例如，256 字节磁盘缓冲队列中的 Message_Index 格式如图 3 所示.

ID 4字节	采集时间 4字节	报文存放位置 4字节	报文长度 4字节

图 3　3256 字节 Message_Index 存储格式

Fig. 3　The storage format for Message_Index 256 bytes

在 DCQD 中，数据报文数据部分 Message_Data 与数据报文索引部分 Message_Index 在磁盘上存放的时候，有两种方式：(1) 一个数据报文的 Message_Index 与 Message_Data 相邻存储[8]，如图 4 所示；(2) 将磁盘缓冲队列的存储空间分成索引区与数据区两部分，Message_Index 存放在索引区中，Message_Data 存放在数据区中，如图 5 所示.

Message _Index1	Message _Data1	Message _Index2	Message _Data2	……

图 4　消息数据存储方式(1)

Fig. 4　The storage mode (1) for message data

图 4 中方式(1)的优点是每次向磁盘缓冲队列中写入一个数据报文的时候，由于 Message_Index 与 Message_Data 是连续存放的，因此可以一次性将 Message_Index 和 Message_Data 写入磁盘，磁盘也仅仅需要一次寻道操作. 方式(1)的缺点是如

果DCQD崩溃了，那么要恢复所有的数据报文，就必须扫描整个DCQD磁盘缓冲队列存储区域，这需要耗费非常长的时间.

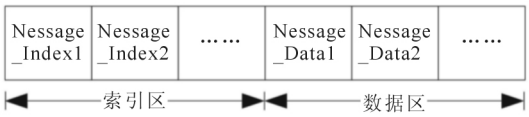

图5 消息数据存储方式(2)

Fig. 5 The storage mode (2) for message data

图5中方式(2)的优点是如果DCQD崩溃，仅仅需要扫描索引区的数据，就可以恢复DCQD中的所有数据报文. 方式(2)的缺点是每次向队列中写入一个数据报文的时候，由于Message_Index与Message_Data没有连续存放，必须写两次磁盘，一次写入Message_Index，一次写入Message_Data. 在绝大多数情况下(除非Message_Index与Message_Data刚好存放在磁盘上的一个磁道，这种情况出现的概率非常低)磁盘需要产生两次寻道操作，效率相对于方式(1)而言，降低了一倍.

文献[9]指出，现代磁盘的寻道模型是非线性的，大量很小的寻道距离也可能累计产生较大的寻道延迟. 文献[10]指出，在设计磁盘存储系统的时候，将存储系统的索引区与数据区尽可能分布在磁盘上的相邻磁道，将显著提高存储系统的性能. 文献[11]与文献[12]指出，如果写入磁盘的数据分为两个部分，例如是数据索引与数据，那么最好将两个部分整合在一起，一次性连续写入磁盘，这样可以提高存储系统的性能.

为了提高DCQD的存储性能，本系统在DCQD中采用了综合方式(1)与方式(2)的数据报文混合存储方式，具体的做法是：DCQD接收到一个数据报文后，将数据报文的Message_Index和Message_Data两个部分在内存中组合成一个大数据块，然后一次性写入到磁盘中，接下来再将Message_Index放入内存中的Message_Index数组中. 每当完成了20个数据报文的磁盘同步写入操作后，将内存中Message_Index数组中存放的20个Message_Index组合成一个数据块一次性写入到磁盘的索引区中. 这样，在索引区和数据区中都有Message_Index数据. 只是，在尚未完成20个数据报文的磁盘同步写入操作的时间内，索引区中的Message_Index会比数据区中的Message_Index少20个. 如果在这段时间内，系统发生崩溃，恢复全部的数据报文，也仅仅需要扫描磁盘缓冲队列中最后的20个数据报文，避免了扫描整个磁盘缓冲队列数据区. 因此，DCQD由于使用了数据报文的混合存储方式，使得DCQD的数据报文崩溃后恢复效率很高，恢复时间很短.

3.2 磁盘缓冲队列的数据清洗

在DCQD磁盘缓冲队列中保存的数据报文，最终还需要插入到数据库中，这个过程称之为DCQD磁盘缓冲队列数据清洗. DCQD数据清洗与数据写入是同时进行的. 数据清洗过程如下：(1)从索引区中一次性读取20个Message_Index数据；(2)逐一根据Message_Index数据，读取20个Message_Data数据到内存中；(3)将20个Message_Data数据分成5组，每组中的4个Message_Data构成一个数据库的INSERT操作，5个INSERT操作构成一个数据库事务，并使用数据库事务操作将数据插入到数据库中；(4)数据库写入事务完成后，更新索引区中的20个Message_Index数据，将其ID设置为0，表示数据报文已经插入到数据库中，该数据报文所使用的磁盘缓冲队列存储空间被释放，可以被其他报文重复使用.

文献[13]指出，向数据库插入数据的时候，在一个INSERT操作中，使用多VALUES方式在一条INSERT操作中插入多个数据和在一个事务中执行多条INSERT操作，都可以显著地提高数据库插入数据的效率. 由于DCQD在磁盘缓冲队列中缓存了采集的数据报文，因此有条件将多条数据报文采用上述优化方法来插入到数据库，可以显著地提高系统的整体性能. 另外一个方面，如果一个物联网数据采集系统在内存中缓存采集的数据报文，也可以实现上述数据库插入优化过程. 但是，通过在内存中缓存数据报文来实现数据库插入优化，可能导致在系统崩溃的时候，丢失尚未写入到数据库的数据报文. 而DCQD的数据报文缓存是通过在磁盘缓冲队列中将数据报文同步写入磁盘来实现的. 因此即使系统突然发生崩溃，也不会导致数据报文的丢失. 所以，DCQD具有既能确保不丢失数据报文，又能提高存储性能的特性.

4 DCQD应用方式

无论是基于Linux系统或者Windows系统构建的物联网数据采集系统，DCQD都可以得到应

用．最简单的 DCQD 应用方式是在 Linux 或 Windows 系统中，使用一个没有存储文件的独立分区作为 DCQD 的磁盘缓冲队列的磁盘分区．这种方式可以保证 DCQD 磁盘缓冲队列的性能．

DCQD 的最佳应用方式是数据采集服务器拥有自己的 RAID 磁盘阵列来存放系统，同时使用外置磁盘阵列作为数据库磁盘存储空间，如图 6 所示．在物联网数据采集系统中，通常的数据采集服务器，都可以支持实现这样的最佳 DCQD 应用方式．在数据采集服务器中，有内置的 RAID0＋1 磁盘阵列，一般配置两个磁盘，组成 RAID0＋1．同时，数据采集服务器外接 RAID5 磁盘阵列，作为数据库系统的磁盘存储设备．

在图 6 所示的最佳应用模式中，DCQD 的磁盘缓冲队列设置在数据采集服务器上的 RAID0＋1 磁盘阵列上，数据库存储设置在数据采集服务器外接的 RAID5 磁盘阵列上．这样 DCQD 的磁盘缓冲队列与数据库的磁盘存储是两个独立的硬件存储通道，不会导致磁盘缓冲队列与数据库存储同时工作时的相互影响，从而具有更高的性能．目前广泛使用的服务器上内置的 RAID0＋1 阵列一般可以配置 500 G 到 1 T 字节的高速磁盘，这样保证了 DCQD 中的磁盘缓冲队列通常具有 500 G 到 1 T 字节的缓冲空间，可以提供对大型物联网数据采集系统的支持．

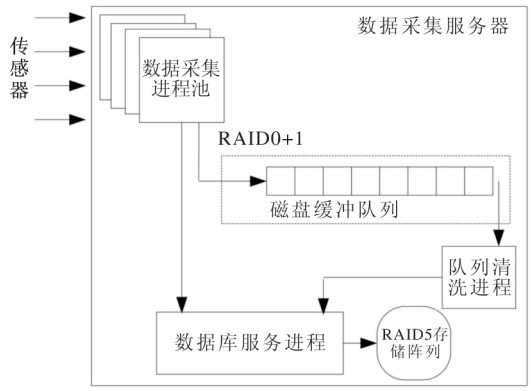

图 6　DCQD 实际应用方式

Fig. 6　The practical application mode for DCQD

5　性能对比实验

为了验证 DCQD 的性能，本文使用了一台 IBM xServer 3850 作为数据采集服务器来进行性能对比实验．IBM xServer 3850 的应用方式如图 6 所示，服务器上运行的操作系统为 Windows Server 2008 企业版，运行的数据库软件为 MySQL 数据库．图 7 是测试 10 万条采集数据的写入时间，对比的两种应用模式分别是应用 DCQD 和采集程序将采集的数据字节写入到 MySQL 数据库中．DCQD 中的磁盘缓冲区的大小设置为 1 G 字节．在图 7 中，横坐标是记录的大小．分别是 64 字节与 4096 字节，最后一个混合大小是 70％的数据报文为 64 字节，30％的数据报文为 4096 字节．

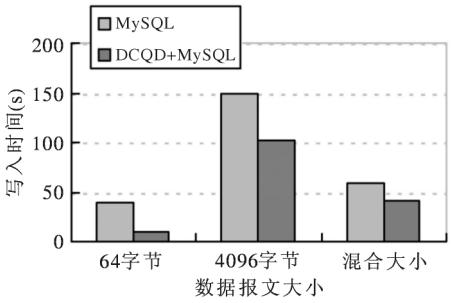

图 7　写入时间对比实验

Fig. 7　Writing time comparison experiments

从图 7 可以看到，无论在数据报文是 64 字节，还是 4096 字节或者是混合情况，DCQD 都比直接写入数据库的情况下具有更好的性能．

图 8 是测试 DCQD 模式与直接写入 MySQL 模式的最大数据采集带宽的性能对比，测试的环境与图 7 类似，测试 60 s 时间内插入到数据库中的记录条数．横坐标是数据采集报文的大小，例如横坐标 64 表示数据报文 80％是 64 字节大小，其它的报文在 64 字节至 4096 字节之间随机选择；纵坐标是插入的记录条数，单位是万条．从图 8 可以看到，在不同报文大小情况下，DCQD 的最大写入吞吐率都远远高于直接写入 MySQL 数据库的情况．

图 9 是在图 8 实验基础上，对报文平均写入延迟的对比．可以看到，在平均写入延迟上，DCQD 模式与直接写入 MySQL 模式，基本上是一致的．分析其原因在于，直接写入 MySQL 模式导致了 MySQL 数据库处于重负荷情况，数据报文在执行 INSERT 操作写入到数据库的时候，延迟很大．虽然 DCQD 模式引入了在磁盘上缓冲数据报文的过程，但是因其优化了写入数据库的性能，使数据库同步写入这个环节整体效率更高，所以

总体上数据报文的平均写入延迟与直接写入MySQL模式对比是相当的. 图8与图9的实验表明了DCQD模式在系统重负荷情况下，显著提高了系统的最大吞吐量，同时还保证了数据报文的平均写入延迟不会变长.

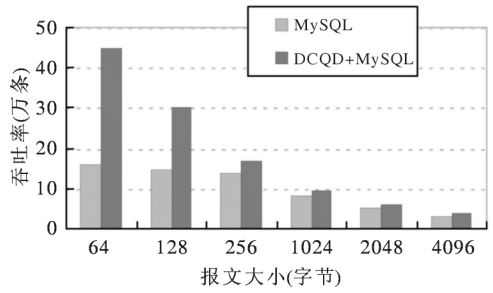

图8 不同大小数据报文的最大写入带宽

Fig. 8 The maximum write bandwidth for data packets of different sizes

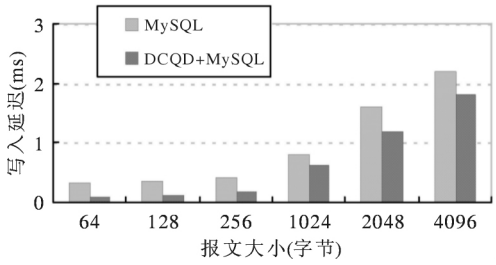

图9 不同大小数据报文在最大带宽情况下的写入延迟

Fig. 9 Writing delay for data packets of different sizes in the case of maximum bandwidth

6 结束语

本文基于物联网数据采集的典型应用情况，针对海量采集数据写入数据库而导致的系统小数据同步写性能低下的问题，设计了面向物联网高速海量数据采集的高性能数据库磁盘缓冲队列DCQD. 在DCQD设计中，使用磁盘缓冲区来实现基于小数据同步写的采集报文缓冲，然后将缓冲的采集数据采用优化的方式高效率地写入到数据库中. DCQD在设计磁盘缓冲区的时候，还采用了新颖的混合报文存储方式，既保证了磁盘缓冲区的性能，又提供了高效率的崩溃恢复机制. 基于上述设计，DCQD很好地利用了磁盘缓冲区高速的特性，对物联网采集数据进行缓冲，然后再使用优化方式写入到数据库中，既保证了采集数据可靠性，又大幅提高了整个物联网数据采集系统的性能，具有显著的应用价值.

参考文献:

[1] 丁治明，高需. 面向物联网海量传感器采样数据管理的数据库集群系统框架[J]. 计算机学报，2012，35(6)：1175.

[2] Peacock J K. The counterpoint fast file system [C]// Proceedings of the USENIX Winter Conference. Dallas，Texas，USA：USENIX Association，1988.

[3] McVoy L W，Kleiman S R. Extent-like performance from a UNIX file system[C]// Proceedings of the USENIX Winter Conference. Dallas，Texas，USA：USENIX Association，1991.

[4] 熊建刚，冯丹. 高可用的磁盘阵列Cache的设计和实现[J]. 计算机工程与科学，2006，28(8)：119.

[5] 郭丹，卢正添，陈文. 一种数据表实时备份系统的高速缓存方法[J]. 四川大学学报:自然科学版，2010，47(5)：998.

[6] 魏青松，卢显良，周旭. FastQueue一种高性能的磁盘队列存储管理机制[J]. 计算机科学，2003，30(10)：81.

[7] Sweeney A，Doucette D，Hu W，*et al*. Scalability in the XFS file system[C]// Proceedings of the USENIX 1996 Annual Technical Conference. San Diego，CA：USENIX Association，1996.

[8] 黄传波，胡晓勤，马晓旭. 一种基于Winnowing分块的重复数据删除备份与恢复系统设计与实现[J]. 四川大学学报：自然科学版，2012，49(3)：535.

[9] Ruemmler C，Wilkes J. An introduction to disk drive modeling[C]. Comput，1994，27(3)：17.

[10] Mckusick M K，Joy W N，Leffler S J，*et al*. A fast file system for UNIX[J]. ACM Trans on Comput Sys，1984，2(3)：181.

[11] Ganger G R，Kaashoek M F. Embedded inodes and explicit grouping：exploiting disk bandwidth for small files[C]//Annual USENIX Technical Conference. Anaheim，CA：USENIX Association，1997.

[12] Rosenblum M，Ousterhout J K. The design and implementation of a log-structured file system[J]. ACM Transactions on Computer Systems，1992，10(1)：26.

[13] Zawodny J D，Balling D J. High performance MySQL [M]. USA：O'Reilly Media，Inc，2004.