文章编号：2096-1472(2020)-11-17-03
DOI:10.19644/j.cnki.issn2096-1472.2020.11.005

面向互联网应用的大规模数据实时查询优化方法研究

沙梦钒[1]，徐兰梅[2]，滕庆勇[1,3]，王小林[1,2]

(1.安徽工业大学信息技术研究院，安徽 马鞍山 243000；
2.安徽工业大学计算机科学与技术学院，安徽 马鞍山 243000；
3.马鞍山学院，安徽 马鞍山 243000)
✉649945261@qq.com; 1294641349@qq.com; 2322619935@qq.com; wxl@ahut.edu.cn

摘 要：JSON通常被广泛应用于服务器与客户端浏览器之间的数据交互。某些场景下，由于客户端请求数据量过大，会导致服务器运算时间及网络传输时间加长，严重影响用户体验。针对此问题，提出了一种数据压缩、数据缓存及分页传输的优化处理策略，查询数据库的数据缓存至服务器内存中，在相同的查询条件下不再查询数据库而是通过读取内存数据库Redis直接调取数据，并通过Gzip将数据压缩之后再通过分页方法，每次仅传输数据量的一部分到客户端浏览器。通过真实金融大规模数据集进行方法验证，结果表明，该方法能提高了45%以上的查询效率，有效地改善用户体验。

关键词：数据压缩；缓存；内存数据库；分页
中图分类号：TP393.01 **文献标识码**：A

Real-time Query Optimization for Large-scale Data of Internet Applications

SHA Mengfan[1], XU Lanmei[2], TENG Qingyong[1,3], WANG Xiaolin[1,2]

(1.*Information Technology Research Institute*, *Anhui University of Technology*, *Ma'anshan* 243000, *China*;
2.*School of Computer Science*, *Anhui University of Technology*, *Ma'anshan* 243000, *China*;
3.*Ma'anshan University*, *Ma'anshan* 243000, *China*)
✉649945261@qq.com; 1294641349@qq.com; 2322619935@qq.com; wxl@ahut.edu.cn

Abstract: JSON (JavaScript Object Notation) is widely used for data exchange between server and client. However, large amount of data requests may result in long time delay on searching and transmission which will seriously affect users' experience. To solve these problems, this paper proposes an optimization strategy of data compression, data cache and paging transmission. Data of the query database is cached in memory. Next time with the same query conditions, data can be directly retrieved by reading Redis, a memory-based database, instead of being queried from within a disk-based database. The data will be compressed by Gzip compression technology and then transferred only a part by paging method to the browser each time. It is verified though the real large-scale financial data sets. The results show that the proposed strategy can effectively improve query efficiency by more than 45%, with improved user experience.

Keywords: data compression; cache; memory database; paging

1 引言(Introduction)

随着社会的不断发展，计算机技术越来越成熟，大量的软件系统不断地被研发出来服务于人们。人们一般通过浏览器或者APP去获取一些自己需要的信息或者对一些信息进行操作，前后端的信息交互就是实现各类需求的一种重要的手段。然而有些时候我们需要实时查询一些大规模数据，这些大型数据的传输会产生费时、占用过多网络资源等不利影响，严重影响用户体验。如何解决这个问题成为一个重要课题。目前已有人提出过几种类型的解决方案：第一类是查询优化，通常为对数据结构的优化[1]或者对索引的优化，索引是为了加速对表中数据行的检索而创建的一种分散的存储结构。索引是针对表而建立的，它是由数据页面以外的索引页

基金项目：安徽高校自然科学研究重大项目(KJ2019ZD09)；安徽省重点研究与开发计划项目(202004a07020028).

面组成的，每个索引页面中的行都会含有逻辑指针，以便加速检索物理数据[2,3]。第二类是采用了分布式储存查询，分布式网络存储采用可扩展的系统结构，利用多台存储服务器分担存储负荷，利用位置服务器定位存储信息，提高了系统的可靠性、可用性和存取效率[4]。第三类是数据压缩，通过对数据量的压缩再进行查询，之后再进行解压，该方案在查询效率上有很大提升[5,6]。

本文提出了一种思路，把查询数据或查询条件存入内存数据库，再次做相同请求的时候不再查询数据库而是直接读取缓存，并通过分页的方法将大规模数据切割成许多部分，每次仅传输一部分给前端[7,8]。并在传输的过程中利用Gzip压缩技术压缩数据的方法[9]。本文在查询数百万级及更大量的数据的情况下，进行了多次实验，通过设置对照实验得出结论。

2 相关知识(Relevant knowledge)

Redis(Remote Dictionary Server)，即远程字典服务，是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。Redis是一个key-value存储系统，支持各种不同方式的排序。为了保证效率，数据都是缓存在内存中。Redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。同步对读取操作的可扩展性和数据冗余很有帮助。

在Web开发的过程中，通常会遇到分页技术。分页技术的核心思想就是当人们需要提取大量数据的时候，不可能一次提取所有的数据，通过提取其中的一部分给用户，在用户需要的情况下再次提取剩下的部分。做到了大块切割成小块，每次只利用小块部分。常见的分页方法为客户端分页、数据库分页和服务器分页三种方法。

(1)客户端分页。将全部或多页结果数据一次返回给客户端，客户端通过展现组件进行分页控制，优点是减少了客户端和服务器的交互次数，客户端进行数据缓存，提高了系统交互性，缺点则是增加了第一次交互的负荷。

(2)数据库分页。进行数据查询时，数据库返回一页数据给客户端。优点是每次从返回的数据库返回较少数据，当次交互负荷较轻。缺点则是每次切页都需要访问数据库，增加了数据库访问并发性。

(3)服务器分页。是介于上述两种方法之间的方法。优点是在1、2之间达到了平衡，既减少了数据库并发又使服务器和客户端当次交互的负荷减少，缺点则是需要考虑数据缓存，数据同步等问题，增加了系统复杂性。

Gzip是GNUzip的缩写，最早用于UNIX系统的文件压缩。HTTP协议上的Gzip编码是一种用来改进web应用程序性能的技术，web服务器和客户端(浏览器)必须共同支持Gzip。目前主流的浏览器，Chrome、firefox、IE等都支持该协议。常见的服务器如Apache、Nginx、IIS同样支持Gzip。Gzip压缩比率在3到10倍左右，可以大大节省服务器的网络带宽。Gzip 对于要压缩的文件，首先使用LZ77算法的一个变种进行压缩，对得到的结果再使用Huffman编码的方法(实际上gzip根据情况，选择使用静态Huffman编码或者动态Huffman编码)进行压缩。其工作原理如下：

首先由浏览器请求url，并在request header中设置属性accept-encoding:gzip。表明浏览器支持Gzip。在服务器收到浏览器发送的请求之后，判断浏览器是否支持Gzip，如果支持Gzip，则向浏览器传送压缩过的内容，不支持则向浏览器发送未经压缩的内容。一般情况下，浏览器和服务器都支持Gzip，response headers返回包含content-encoding:Gzip。最后浏览器接收到服务器的响应之后判断内容是否被压缩，如果被压缩则解压缩显示页面内容。如图1所示。

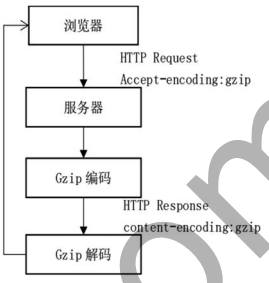


图1 Gzip用于服务器与客户端之间通信的工作原理

Fig.1 Gzip works for communication between server and client

3 实时查询优化方法(Real-time query optimization strategy)

本文针对大规模数据实时查询的情况进行了方法优化，当我们需要查询大规模的数据时，将数据通过key-value的形式储存在Redis里。在后续的查询请求里，我们可以直接查询Redis数据库调取出相应的数据。该操作大大减少了连接数据库的时间，减少了resultSet封装成对象的过程，避免了重复查询数据库的操作。在本文实验中并没有直接将百万级数据放在Redis数据库中，因为数据量庞大，不适合这样操作处理。本次采取的方法是将构造特殊的查询参数放入Redis中，在查询数据的时候从数据库中取出，并结合相应算法快速得出前端所需要的部分并返回给前端。

本次实验的数据采用金融资产原始数据，对于整张视图查出来的数据，要根据nvc_code和dtm_share_date这两个字段进行排序返回前端进行显示。按照这两个字段进行排序。每次后台只向前台返回n条数据。如何正确地选出这n条数据很关键。思路的关键就是从数据库count出每个资产的总条数，保存在numberList[i],i指该资产在所选的资产里排序第几。对于某些页显示的数据存在多个资产的数据，因此，把某页的数据的组成，用下面这句话来描述：第page页的数据由A资产的第start到第end，一共end-start+1条数据和B资产的第begin到stop，一共top-begin+1条数据和C资产组成。为了确定这些start、end、begin、stop等值，定义一个数组remaList。把该资产数据填充完之后，位于第几页还剩多少条数据需要后面资产填充的，这个多少条数据的数值存在remaList[i],i指该资产在所选的资产里排序第几。为了与前端传来的page字段挂钩，需要在定义一个数组pageList。该数组存放每个资产填充完之后的起始页，若该资产的总条数小于需要填充的条数，该资产存放的值等于前一个资产存放的pageList中的值。下式描述了每一页的分页条数n与资产数据X_i的填充关系。

$$n=\sum_{i=1}^{i=m}\lambda X_i+\alpha_i,\ \lambda\in\{0,1\}$$

n为设置的分页数，即每一页有多少条数据，m为资产数，α为第i个资产所需要填充的数据量，λ为常量，取值为0或者1，当X_i资产的总条数大于分页数n时，λ取0，否则λ取1。

表1是以分页数为50时，7种金融资产数据为例的情况示意图。

表1 分页数为50时数组所需要填充的数据情况

Tab.1 The array needs to be populated with when the number of pages is 50

相关变量	变量与相关资产对应关系									
资产名	A	B	C	D	E	F	G			
每个资产总条数	25	15	60	90	120	45	135			
page(页码数)	1	2	3	4	5	6	7	8	9	10
每一页数据所需要的填充情况	a.25 b.15 c.10	c.50	d.50	d.40 e.10	e.50	e.50	e.10 f.40	f.5 g.45	g.50	g.40
numberList(每种资产所有的总条数)	25	15	60	90	120	45	135			
rameList(每种资产填充后剩余填充数据)	25	10	0	10	40	45	40			
pageList(每种资产填充后的起始页数)	1	1	2	3	5	8	9			

将上述List数组存入redis中后，每当浏览器传回page页请求相应数据时，可以通过读取redis中的数组快速计算出前端所需要的那部分数据并返回给浏览器。

再取得数据后，开始通过服务器开启Gzip压缩，以NGINX为例，首先设置gzip on参数开gzip模式，接下来通过gzip_min_length参数配置压缩起点，例如文件至少大于1k才开启压缩，然后通过gzip_comp_level设置gzip的压缩级别，该级别分为1—9级，级别越大压缩的程度越大，同时对CPU的负担也会逐渐增大。再通过gzip_type参数设置一下压缩的文件类型，本文中为了解决请求大规模数据的优化策略，前后端交互是采取json的数据格式，因此这里配置的文本类型一般设置为application/json，这样就可以针对此类型的请求操作进行压缩。接下来通过gzip_vary参数配置是否在http-header开启Accept-Encoding，开启该参数配置是为了让浏览器通知服务器自己是支持Gzip压缩模式的，当浏览器接收到压缩数据后会自动解压操作。再通过gzip_buffers参数设置缓冲区大小，以4k为单位，若大于4k时，例如7k则分配为2*4k的缓冲区。最后设置一下gzip_http_version参数，该参数是协议版本，例如版本为1.1则设置1.1即可。

在压缩完数据后，服务器向前端开始传输数据，此时通过页数参数，选择其中的部分数据返还给前端浏览器。例如页码参数为1时，我们可以传第1至100条的数据，假设此时查询的数据共计100万条，本次仅仅只传输了万分之一，当用户需要下一批数据时，我们在通过页码参数2再次请求服务器，服务器从redis缓存的100万条数据中直接取出第101至200条数据。在每一次的请求操作过程中，用户都只拿到大规模数据的一小部分。每一次请求的数据量可以根据具体需求做出改变。

4 实验结果及分析(Experimental results and analysis)

本次实验环境为一台PC机，8G内存，i5-5200U，2.20GHz处理器，100M移动宽带，浏览器为Chrome。服务器部分是采用分布式系统架构，每台服务器的硬件环境为CPU：4核Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz，内存16GB，硬盘200GB，操作系统采用Linux发行版之一的Centos 7.8版本。

本文通过设置对照实验，开启Gzip压缩和未开启Gzip压缩时的效果对比、分页传输效果对比试验。本文中的实例因数据查询条件过于复杂，在未分页的情况下，百万级数据会显示超时，且对浏览器负担较重。所以在Gzip压缩对比实验中在保证相同的分页数情况下获取实验数据以此观察压缩效果(表2)，在分页效果对比实验中，在相同数据量级下不同级别的分页数量来获取实验数据以此观察出分页效果(表3)。以百万级的数据为基准，逐渐增大数据量，通过10次采样取平均值的方法开始试验。试验对比结果如图2所示。

表2 在相同分页数量下开启压缩前后效果对比

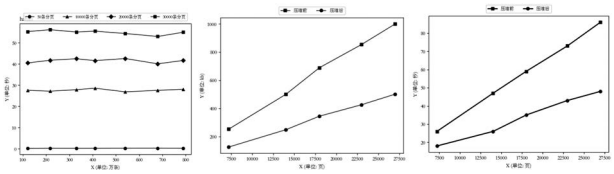
Tab.2 Contrast before and after starting compression with the same number of pages

实验变量	实验数据				
分页数	7200	14000	18000	23000	27000
压缩前	26s	47s	59s	73s	86s
压缩后	18s	26s	35s	43s	48s
压缩前	254kb	501kb	690kb	854kb	1000kb
压缩后	127kb	250kb	346kb	427kb	502kb

表3 相同数据量级下不同分页数量的传输效果对比

Tab.3 Comparison of transfer effects of different number of pages under the same data

实验变量	实验数据						
数据量	$120*10^4$	$217*10^4$	$330*10^4$	$410*10^4$	$540*10^4$	$680*10^4$	$790*10^4$
50	0.21s	0.27s	0.25s	0.26s	0.3s	0.31s	0.27s
10000	27.6s	27.2s	27.9s	28.6s	26.9s	27.6s	28.1s
20000	40.5s	41.8s	42.5s	41.6s	42.6s	40.1s	41.7s
30000	55.3s	56.2s	55.1s	55.5s	54.4s	53s	55s



(a)不同分页数量的效果对比　(b)压缩前后大小对比　(c)压缩前后传输时间对比

图2 对照实验数据情况

Fig.2 Comparison of experimental data

从图2和表2可得知，本次实验开启的Gzip压缩率约为50%，在分页数量分别设置为一页传输7200条数据，一页传输14000条数据，一页传输18000条数据，一页传输23000条数据，一页传输27000条数据共五组情况下，在压缩前文件大小分别约为254kB、501kB、690kB、854kB、1000kB，传输数据所需要的时间分别约为26秒、47秒、59秒、73秒、86秒。当开启Gzip压缩后文件大小分别约为127kB、250kB、346kB、427kB、502kB，传输数据所需要的时间分别约为18秒、26秒、35秒、43秒、48秒。两组数据的对比可以明显看出，随着每一页分页量的增加，从查询到传输给前端的时间也逐渐增加，呈线性增长。开启Gzip压缩后，数据的大小减少了约50%，所花费的时间也是比未压缩时少了很多，通过此次类比可以得出：在百万级的数据量级别下，Gzip压缩有着非常有效地优化效果。

从图2和表3可知，在都设置了Gzip压缩的条件下，分别设置了分页数为50、10000、20000、30000的四组对照实

验。由图中可得知，在分页数为50条一页的情况下，平均仅仅需要0.2秒便可获取到数据，并且随着量的增加，最后到700百万的时候时间也保持着稳定，在0.27秒左右便可传输完毕。在分页数为10000时，七组数据量的实验下，时间平均在27秒左右。在分页数为20000时，七组数据量的时间则平均在41秒左右。在分页数为30000时，七组数据量的时间则平均在55秒左右。四组的对照实验可以看出，时间在数据量的增大下，几乎保持着水平趋势，相对稳定。且随着分页数越来越小，所需要的时间也越来越少。由此可以推断出，在大规模数据传输下，合理的分页能有效地提高传输效率，缩短传输时间，很好的改善用户的体验。

试验结果表明，未采用内存数据库、Gzip压缩及分页传输的数据查询在数据规模较小时，处理时间在接受范围之内。当数据规模足够庞大时，处理时间过长，客户端浏览器负担很重。而基于内存数据库、Gzip压缩和分页传输的大规模数据查询时，较好地解决了困难，在不同等级的数据量下均有良好的表现。

5 结论(Conclusion)

在传统的B/S模式的互联网应用遇到大规模数据查询时的高计算延时、高传输时间问题下，本文从内存数据库、Gzip压缩及分页传输的角度去改善效率问题。从大量数据实验测试，该方案的确改善了效率问题，减少了查询时间，提高了网络传输效率，证实了该方案的可行性。该方案虽然有较为良好的效果，但是从实际应用的效果分析，还有着进一步改善提高的空间。

参考文献(References)

[1] 王小林,刘敏,徐宏,等.一种移动互联网序列化数据的传输优化方法[J].安徽工业大学学报(自然科学版),2017,34(01):71-75.

[2] 夏秀峰,张刘畅,刘向宇.面向大规模图数据的分布式可达性索引与查询策略[J].计算机工程,2018,44(03):65-72.

[3] 陈大伟,张清,刘敏.试论云计算环境下的分布式存储技术[J].科技展望,2016,26(31):16.

[4] 郭庆,朱一凡,谢莹莹,等.面向大规模网络流量数据的实时汇聚查询关键技术研究[J].小型微型计算机系统,2020,41(06):1314-1320.

[5] FuTao Ni, Jian Zhang, Mohammad N. Noori. Deep learning for data anomaly detection and data compression of a long-span suspension bridge[J]. Computer Aided Civil and Infrastructure Engineering, 2020,35(7):685-700.

[6] Rui-Peng Hu, Chun-Xiong Huang. Optimized compression technology for spatial data network transmission[J]. Journal of Discrete Mathematical Sciences and Cryptography, 2018,21(2):557-562.

[7] 奚科芳.常见关系数据库实现分页[J].数字技术与应用,2020,38(01):44-45.

[8] 王志娟,班娅萌,平金珍.基于AJAX技术和JAVAEE的分页查询优化[J].信息通信,2019(01):118-119.

[9] 赵雅倩,李龙,郭跃超,等.基于OpenCL的Gzip数据压缩算法[J].计算机应用,2018,38(S1):112-115;130.

作者简介：

沙梦钒(1995-)，男，硕士生.研究领域：互联网应用技术.

徐兰梅(1994-)，女，硕士生.研究领域：微服务前端技术.

滕庆勇(1994-)，男，硕士，助教.研究领域：敏捷软件开发技术.

王小林(1964-)，男，硕士，教授.研究领域：自然语言处理.本文通讯作者.

(上接第36页)

关锁等功能，屏幕提示信息、语音播报和一键关锁功能使密码锁更加人性化。系统易于安装、稳定性好，安全可靠，可应用于柜子锁和门锁等多种场所。但是系统存在一些不足之处，缺少电池模式，语音模块拥有优越的语音识别和语音合成功能，没有充分发挥语音模块的作用。今后，将增加电池模式，使系统拥有市电供电、充电宝供电、电池供电等三种供电方式，满足用户对供电方式的要求，解决现有的智能门锁在供电方面存在的使用不便的问题。增加语音密码识别功能，但是语音密码开门的同时泄露了密码，安全性差[3]，因此还需要设置语音密码识别的开和关。为了提高密码锁的安全性，可以增加GSM模块，若三次密码输入错误，键盘锁死三分钟，同时通过GSM模块向用户手机号发送提醒短信。还可以增加远程访问功能，使用蓝牙技术建立智能手机和密码锁之间的通信[7]。

参考文献(References)

[1] 杨朋飞,聂亮,陈靖,等.基于STC89C52单片机的指纹密码锁系统设计与实现[J].传感器与微系统,2020,39(05):81-83;86.

[2] 徐庆伟,郭振铎,刘洲峰.基于STM32的电子密码锁设计[J].中原工学院学报,2018,29(06):61-65.

[3] 栾禄祥.基于GPRS和激光虚拟键盘的智能电子门锁系统[J].计算机应用,2016,36(S2):319-321.

[4] 刘永雷,赵曰峰.基于STM32的智能电子密码锁系统设计[J].山东师范大学学报(自然科学版),2018,33(03):322-327.

[5] 范冠鹏,李永军,王亚杰,等.基于超声波传感器和STM32的局部放电监控系统设计[J].仪表技术与传感器,2020(06):55-58.

[6] 张萍,马树军,史可福.基于51单片机的指纹电子密码锁的设计与实现[J].实验室研究与探索,2018,37(08):134-138;161.

[7] Chi-Huang Hung, Yong-Yi Fanjiang, Kun-Chih Chung, et al. A Door Lock System with Augmented Reality Technology[C]. 2017 IEEE 6th Global Conference on Consumer Electronics(GCCE 2017), Nagoya, Japan, 2017.

作者简介：

谭 虹(1971-)，女，本科，信息系统项目管理师.研究领域：嵌入式技术.