第57卷 第2期 天 文 学 报 Vol.57 No.2
2016年3月 ACTA ASTRONOMICA SINICA Mar., 2016

doi: 10.15940/j.cnki.0001-5245.2016.02.011

ElasticSearch分布式搜索引擎在天文大数据检索中的应用研究*

陈亚杰[1] 王 锋[1,2†] 邓 辉[1] 刘应波[1]

(1 昆明理工大学云南省计算机技术应用重点实验室 昆明 650500)
(2 中国科学院云南天文台 昆明 650011)

摘要 天文观测数据是天文研究的基础, 但传统的集中式数据检索方法已难以满足日益增长的海量天文数据的高性能检索和查询需求. 提出了一种基于ElasticSearch分布式搜索引擎, 通过River机制对现有的海量FITS(Flexible Image Transport System)数据进行索引构建, 从而实现海量FITS数据高效检索的方法, 并讨论了其中的近实时检索和查询的关键技术. 实测结果表明, 在百万到千万级的天文数据量下, 该方法可获得极高的检索性能, 并能够很方便地集成到现有的天文数据归档系统中, 完全可以满足当前国内各类望远镜系统天文数据的归档要求.

关键词 方法: 数值, 方法: 解析, 天文数据库: 其他诸多方面

中图分类号: P112 **文献标识码**: A

1 引言

FITS(Flexible Image Transport System)[1]是国际天文学联合会(IAU)于1982年确定的用于各天文台之间数据传输、交换的统一标准格式, 是天文学界常用的数据格式. 目前, 大量的天文数据都以FITS格式进行存储. 随着天文观测技术和数据获取及处理技术的逐渐成熟, 各天文观测站获取的天文FITS数据已经非常庞大[2]; 此外, 由于不同观测站根据其自身的考虑会采用不同的存储规范, FITS数据文件基本没有完全统一的存储标准. 因此, 在当前海量数据增长背景下, 对海量的FITS数据进行统一、高效以及近实时的检索成为天文学领域亟待解决的一个难题.

采用分布式[3−4]的存储方式是应对海量数据的有效措施. Crestani等人讨论了基于云的海量数据存储与检索问题[5]; Berriman等人讨论了基于云的数据存储在天文领域中的应用[6]. 由此可见, 基于云的分布式数据管理方式具有有效应对海量FITS数据存储与检索问题的可行性. 之前, 已有很多研究人员对FITS数据的存储和检索问题做了一些很有价值的研究. 樊东卫等人设计了FITS文件管理器, 实现了对中小型规模FITS文件的

2015-09-21收到原稿, 2015-10-06收到修改稿

*国家自然科学基金项目(U1231205, 11303011, 11263004, 11463003, 11163004)及云南省应用基础基金重点项目(2013FA013, 2013FA032)资助

†wf@cnlab.net

管理[7]; 王歆设计并实现了基于非关系型数据库的图像数据库系统[8−9]; 刘应波等人分析了使用NoSQL对可变FITS文件头元数据进行存储和检索的可行性[10]; 崔辰州等人采用存储FITS头的方式, 实现了通过FITS头对FITS文件的存储与检索[11]. FITS数据对于天文学的研究来说非常重要, 然而, FITS数据的海量性以及其非结构化的特性, 为其高效存储和检索带来了难题. 本文结合前人的研究, 继续探究海量天文FITS数据的存储和检索问题, 在使用分布式搜索引擎ElasticSearch关键技术的基础上, 分析并研究了对海量FITS数据进行高效检索的可行性解决方案.

2 FITS

2.1 FITS文件

FITS文件由文件头和数据矩阵两部分组成. 文件头能够唯一标识一个FITS文件, 其中包含了FITS文件的标识符、FITS数据的来源、图像的维数、每个维的大小、图像的大小、图像的最大值和最小值、波长、经度、纬度、图像观测者和望远镜、观测目标、拍摄日期、拍摄时间等一系列描述FITS文件的信息. 除固定长的FITS文件头信息外, 其他部分均为文件的具体数据. 然而, 不同的天文观测站都有其实施规范, 存储的天文FITS数据也都有其独特的风格, 并且不同FITS文件描述的天文信息也不尽相同, 并不是所有的FITS文件头中都含有相同的信息. 文中采用存储FITS文件头的方式实现对FITS数据的存储, 为了对FITS数据进行统一管理, 需要将FITS文件头处理成为一种统一的规范.

2.2 FITS文件头

FITS文件头由80字符长的行记录组成，其中行记录是用ASCII字符串写成的. 它由N个记录构成, 每个记录由关键字、值和注释3个部分组成, 值和注释部分可为空. 每个FITS文件的文件头中必须含有SIMPLE、BITPIX、NAXIS和END这4个关键字信息, 并且这4个关键字的顺序是一定的, 它们之间可以存在其他关键字. 第1个关键字必须为SIMPLE, 标识此文件是否为FITS标准格式; 最后一个关键字必须为END, 作为文件结束标志; 关键字BITPIX表示图像数据的格式, 其值为8、16、32、−32或−64; NAXIS表示图像数据矩阵的维数, NAXIS不为0时, 其后紧跟着的关键字表示每个维的大小[12].

由于不同FITS数据的文件头中的N个记录并不完全相同, 这就给所有FITS文件的统一规范存储造成了不便. 然而每个FITS文件的文件头能够唯一标识一个FITS文件. 因此, 可以将所有不同FITS文件头的所有记录抽取出来形成文件头集合, 集合中的每个元素作为一个记录项, 这样, 每条记录便可作为一个FITS文件的元数据来标识此文件. 则一个FITS文件的元数据集合可表示为:

$$
\begin{aligned}
Header = \{&SIMPLE, BITPIX, NAXIS, NAXIS1, NAXIS2,\\
&EXTEND, GONGKEY1, ORIGIN, OBS-SITE,\\
&TELESCOP, OBS-URL, DATE, DATE-OBS,\\
&TIME-OBS, WAVELNTH, WCANAME, CTYPE1,\\
&CTYPE2, CRVAL1, CRVAL2, CRPIX1, CRPIX2,\\
&EPH\_RA, EPH\_DEC, IMTYPE, IMGMN01,\\
&IMGRMS01, IMGSKW01, IMGMIN01, IMGMAX01,\\
&IMGVAR01, \cdots\} .
\end{aligned} \tag{1}
$$

由此, 提取出每个FITS文件的元数据, 将每个FITS文件的元数据作为一行记录标识一个FITS文件, 便可实现在关系型数据库中按文件头统一存储FITS元数据, 为数据检索和访问提供条件.

3　分布式搜索引擎(ElasticSearch)

分布式搜索引擎(ElasticSearch)[13]是一个开源的、基于Lucene[1]的分布式搜索引擎. ElasticSearch可以提供稳定、实时、可靠的检索服务, 具有高可用、易扩展以及近实时的特点. 它采用RESTful的架构风格, 提供了简单易用的查询和共享接口; 使用GET获得请求对象的当前状态, 使用POST改变请求对象的状态, 使用PUT创建一个新对象, 使用DELETE删除请求对象.

ElasticSearch拥有的River机制使得用户可以并行地将大量数据快速同步到集群. 另外, 使用ElasticSearch能够方便地为数据建立索引, 可将一个索引分割成多个索引分片(索引分片数可由用户指定, 默认为5), 然后将多个分片均衡地分布在集群的所有可用节点上, 形成分布式结构, 减轻了单个节点的负担. 在ElasticSearch集群中, 还可以为每个索引分片设置副本(副本数仍然可以由用户自行指定, 默认为1), 当某索引分片失效时, 可以及时使用副本恢复数据. ElasticSearch还拥有自动发现节点机制和快速数据恢复机制, 当有新节点加入集群时, ElasticSearch可及时发现并自动重新进行负载均衡, 为新节点分配数据; 当某节点失效时, 它同样会自动重新为可用节点分配数据.

4　基于ElasticSearch的海量FITS数据检索

天文观测站每天都在不停地观测, 随着时间的推进和大型天文望远镜技术的日益成熟, 天文数据逐渐累积, 天文观测站正面临PB级海量天文数据的存储和检索问题[8]. 关系型数据库已逐渐无法容纳这些快速增长的天文数据, 随着数据量的暴增, 其存储和检索效率也越来越低. 虽然, 通过元数据可实现对非结构化FITS数据的存储, 但是, 海量的数据却难以使用单机来解决. 将分布在不同地域用于存储数据的机器组成集群, 形成分布式结构, 可解决海量数据的统一管理问题; 当数据量持续增加时, 只需要在集群中增加

[1] http://lucene.apache.org/

节点, 而无需改变已有数据和节点.

如2.2节所述, 提取FITS元数据, 将每个FITS文件的元数据作为一行记录写入关系型数据库SQLServer, 可实现对FITS文件元数据的结构化管理. 然而, 当FITS数据的数据量较大时, 相应地, 元数据记录也会随之增长. 随着元数据记录的增长, 其检索效率将会受到影响, 而使用分布式的方式进行检索和访问将能够大大提高其效率. ElasticSearch能够利用自身机制将不同数据库中的数据并行导入到ElasticSearch分布式集群, 因此, 利用ElasticSearch分布式检索集群可以很方便地解决FITS数据的海量性带来的难题. 所以, 使用关系型数据库存储FITS元数据, 不断将数据库中的数据同步到ElasticSearch分布式集群, 利用ElasticSearch对海量数据进行统一的分布式管理具有一定的可行性.

4.1 FITS数据处理流程

基于ElasticSearch的FITS数据处理流程如图1所示. 首先, 从FITS文件中提取出FITS文件头的行记录信息, 将每个记录作为关系型数据库表中的一个字段, 则表中一行记录便可唯一标识一个FITS文件; 然后, 将FITS文件头记录批量导入关系型数据库. 使用ElasticSearch提供的JDBC的River插件[2]可方便快速地将关系型数据库中的FITS元数据导入ElasticSearch集群; 然后, 使用ElasticSearch分布式检索集群便可对海量FITS数据进行快速、稳定的检索.

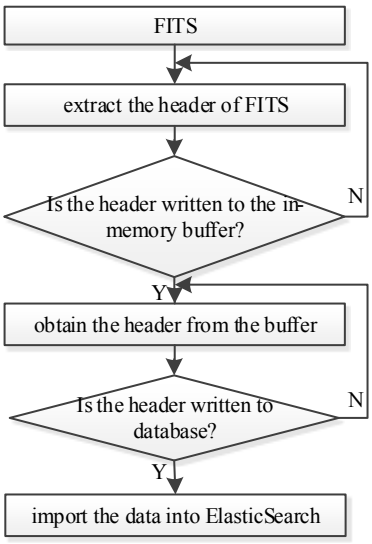

图 1 FITS数据处理流程

Fig. 1 The processing flow of FITS data

4.2 FITS元数据提取

由2.2节所述, 不同组织机构在存储FITS文件时, 根据自身需要, 关注的元数据信息可能并不相同, 所以写入FITS头的内容也存在不同. 本文在实验过程中提取FITS头的所有Key/Value字段, 每个Key作为关系型数据库的一列, 对应的Value构成了关系型数

[2]https://github.com/jprante/elasticsearch-jdbc4

据库中的一行数据.

使用JAVA类库jfits-0.94.jar访问FITS文件[3], 通过类库中FitsFile类的getHDUnit方法返回一个FitsHDUnit类的对象, 通过该FitsHDUnit类对象的getHearder方法返回一个FitsHeader类的对象, 然后通过该FitsHeader类对象的getKeywords方法获取FITS文件头的行记录关键字列表. 由此, 可批量提取每个FITS文件的文件标识记录. 将提取出的每个FITS文件头的关键字列表作为独立一行, 提取所有FITS文件头统一写入关系型数据库SQLServer.

4.3　FITS元数据的索引构建

提取出FITS的标识字段信息写入内存缓存区或临时文件后, 从缓存中获取数据, 将存储的所有FITS标识记录写入数据库的相应数据表中, 数据写入流程如图2所示.

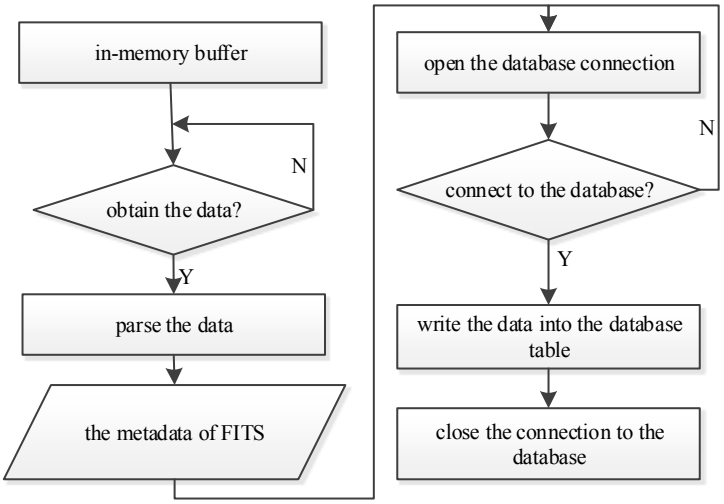

图 2　FITS记录写入数据库流程图

Fig. 2　The flow chart of the FITS records written into the database

当关系型数据库中存储的数据量超过一定程度的时候, 其检索效率会大大降低, 而将这些数据导入ElasticSearch分布式检索集群, 采用分布式结构进行存储和检索, 将会大大提高数据的检索效率. 使用ElasticSearch中JDBC的River插件可将数据快速导入ElasticSearch集群, 并建立索引, 以400万条记录的索引构建为例, 构建方式如图3.

使用图3所示方式为数据构建索引后, 可以根据需要构造查询条件, 可方便地进行结构化查询, 这极大地降低了索引开发管理的难度, 其示意图如图4、图5所示.

此外, 由于ElasticSearch本身提供了分布式集群方式的检索管理机制, 因此, 除了对单一元数据建立索引之外, 使用ElasticSearch还能够方便地对元数据库集群建立索引.

[3]http://www.eso.org/p̃grosbol/fits_java/jfits.html

```
curl -XPUT ‘222.197.221.225: 9200/_river/fitsheader_river/_meta’ -d ‘{
    “type”: “jdbc”,
    “jdbc”: {
        “index”: “fitsheader_index”,
        “type”: “fitsheader_index”,
        “driver”: “com.microsoft.sqlserver.jdbc.SQLServerDriver”,
        “url”: “jdbc:sqlserver://192.168.9.9;databaseName=FitsDB”,
        “user”: “cnlab”,
        “password”: “**********”,
        “sql”: “select top 4000000 * from FitsTable”
    }
}’
```

图 3 在SQLServer数据库上通过River构建索引

Fig. 3 Building the index in the SQLServer database by using the River

图 4 索引构建示意图

Fig. 4 The schematic diagram of the index building

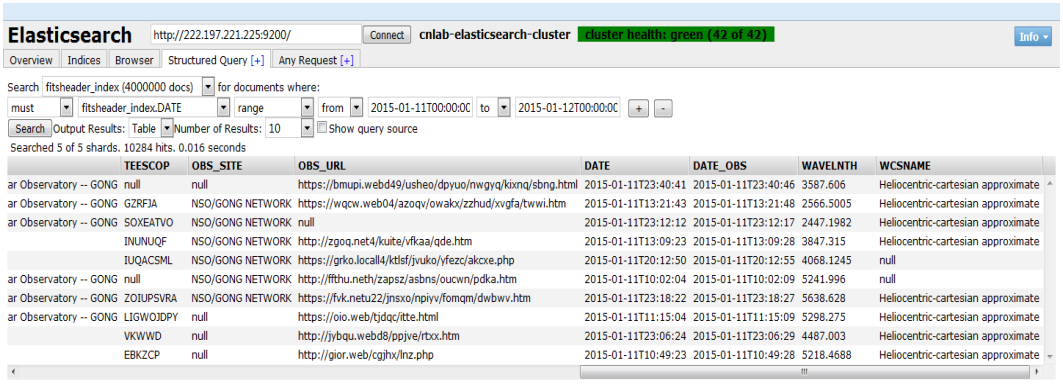

图 5 结构化查询示意图

Fig. 5 The schematic diagram of the structured query

5　实验及讨论

5.1　实验数据

本实验使用1 600万条模拟数据来完成. 4.2节提取出的每个FITS元数据作为数据表的一行记录. 实验中, 将1 600万条数据分成400万条、800万条、1 200万条和1 600万条4组, 在不同条件下进行不同测试.

5.2　实验环境

在windows7(CPU为Pentium(R) Dual-Core CPU E6500 @ 2.93GHz, 内存为4G)下, 使用IntelliJ IDEA 14.0.2编译器编写JAVA程序来提取FITS文件的元数据信息, 并将数据批量导入SQLServer数据库中; 根据已导入的数据使用sql data generator随机生成更多数据. 使用系统配置为Quad-Core AMD Opteron(TM) Processor 2352 CPU @ 2.10 GHz, 内存为4G的服务器(操作系统为CentOS release 6.4)配置ElasticSearch分布式搜索引擎. 测试工具为JMeter.

5.3　实验结果

实验中共设置以下4个查询条件进行测试:

Q1. 查询2015年1月11日拍摄的天文图像;

Q2. 查询纬度为45° (即：latitude=45°)的天文图像;

Q3. 查询纬度和经度均为45° (即：latitude=45° 且longitude=45°)的天文图像;

Q4. 查询2015年拍摄的, 并且经纬度均在45° 到90° 之间(即年份为2015, 45° <latitude <90° 且45° <longitude <90°)的天文图像.

测试分为2组进行:

TEST1: 单机环境下的测试

测试在4个条件下进行查询时, 当数据量逐渐增加时, 其查询响应时间的变化规律.

TEST2: 多机扩展环境下的测试

测试某一固定条件下, 在集群节点数逐渐增加时, 其查询响应时间的变化规律.

响应时间的测试工具为JMeter, 使用10组100个线程, 模拟同时进行100个并发用户的访问, 最后取10组结果的平均值.

5.3.1　单机实验

为了对比, 先设计单机实验, 实验使用ElasticSearch的默认设置(分片数为5, 副本数为1). 单机测试即ElasticSearch集群中只有一个节点的情况: 仅有一个节点时, 所有索引分片全部分配在这个唯一节点上, 副本分片未分配, 此时集群状态为亚健康状态.

在单机实验中, 构建ElasticSearch标准的查询条件进行测试, 以在400万条数据情况下Q4的查询为例, 其查询条件构建方式如图6所示.

按照图6所示方式构建查询条件进行查询, 分别测试了查询条件Q1-Q4, 在数据量为400万条、800万条、1 200万条和1 600万条4种情况下得到结果条数和查询响应时间. 所有查询的结果数量如表1所示, 查询响应时间如图7所示.

```
curl -XPOST '222.197.221.225: 9200/_river/fitsheader_river/_meta' -d '{
"query": {
    "bool": {
      "must": [{"range ": {
              "fitsheader_index.DATE": {
                  "from": "2015"}}
        },
        {"range": {
              "fitsheader_index. latitude": {
                  "from": "45° ",
                   "to": "90°}}
        },
        {"range": {
               "fitsheader_index. longitude": {
                   "from": "45°",
                   "to": "90°}}
        }]
    }
  }
}'
```

图 6 Q4在400万条数据下的查询条件构建

Fig. 6 The query condition building of Q4 when the number of the data is 4 million

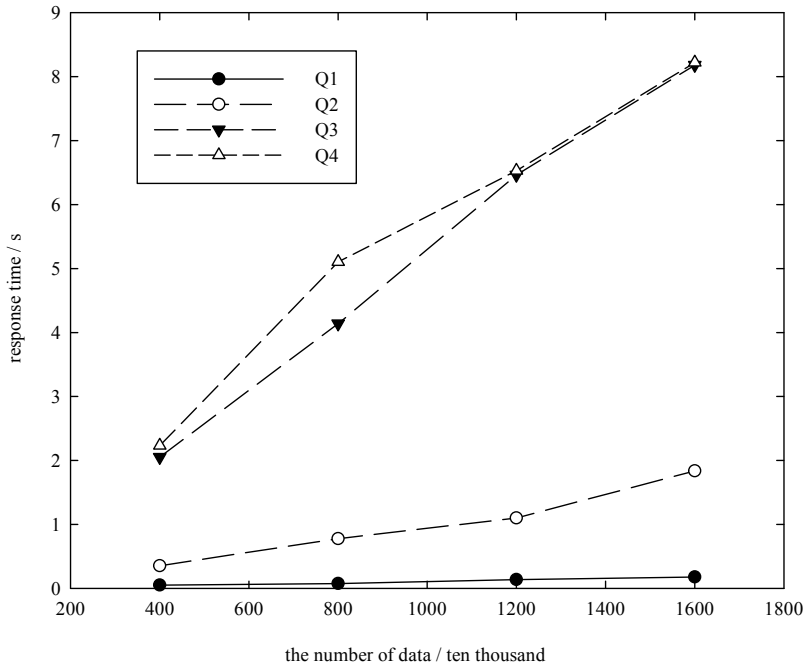

图 7 单机下查询响应时间

Fig. 7 The response time of query in a single computer

表 1　查询结果数(单位：万条)

Table 1　The number of the query results (unit: ten thousand)

query condition \ the number of data	400	800	1200	1600
Q1	1.0284	2.0523	3.1111	4.1443
Q2	99.9603	200.0655	300.0845	400.0697
Q3	99.9353	199.8670	300.0659	400.1160
Q4	30.4745	60.9619	91.4584	121.9936

如表1和图7所示, 在同一条件下, 随着数据量的增加, 显然, 查得的结果数量相应增加, 而且查询响应时间随之增加; 此外, 不同的查询条件在数据量相同时, 其响应时间也有差距. 图7表明, 在同一查询条件下, 查询响应时间与数据量成正相关; 数据量相同时, 查询响应时间与查询条件的复杂程度也成正相关.

5.3.2　多机扩展实验

在多机扩展实验中, 集群节点数逐渐增加, 当节点数大于等于2时, 所有分片(包括副本分片)都被均衡地分配到所有可用节点上, 此时集群状态为健康状态. 为研究集群中节点数对查询响应时间的影响, 在条件Q1下, 每增加一个节点, 就分别查询一次在不同数据量情况下Q1的查询响应时间.

在400万条数据的情况下, 其查询条件构建方式如图8所示.

```
curl -XPOST '222.197.221.225: 9200/_river/fitsheader_river/_meta' -d '{
"query": {
    "bool": {
        "must": [{"range": {
            "fitsheader_index.DATE": {
                "from": "2015-01-11T00:00:00",
                "to": "2015-01-12T00:00:00"}}
        }]
    }
  }
}'
```

图 8　Q1在400万条数据下的查询条件构建

Fig. 8　The query condition building of Q1 when the number of data is 4 million

使用相同的方式构建查询条件进行测试, 在不同数据量和不同节点数情况下, 其查询响应时间结果如图9所示.

图9表明, 在不同数据量下, 当集群节点数增加时, 查询响应时间都会随之降低; 随着节点数的增加, 响应时间会逐渐趋于平稳; 但是, 对于不同的数据量, 响应时间达到稳定的节点数却不相同, 例如: 在400万条数据量的情况下, 节点数到达3时, 响应时间便开始趋于稳定; 而在1 600万条数据量的情况下, 节点数到达5时, 响应时间才趋于稳定. 在

集群节点数增加到一定值时, 查询响应时间总会达到尽可能的最小值, 因此, 对于海量数据的检索来说, 只需要根据数据量增加集群节点数, 对分布式检索集群进行水平扩展便可达到高效检索的目的.

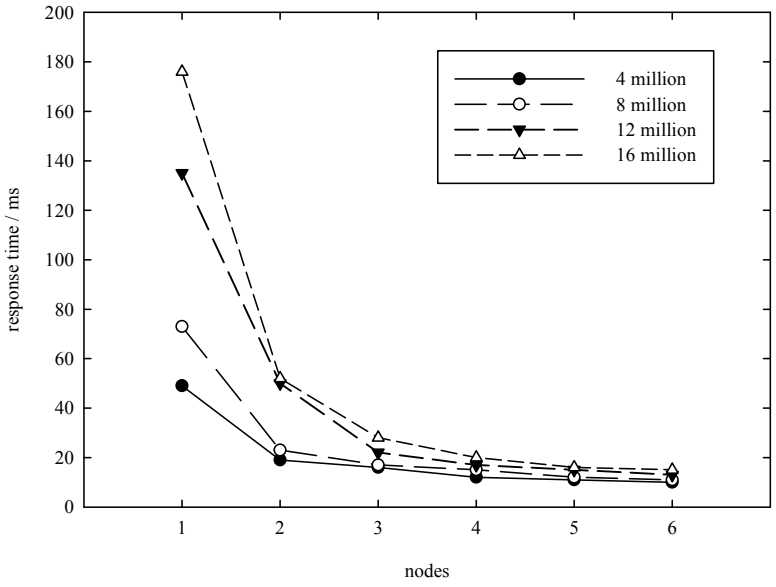

图 9 多机集群下查询响应时间

Fig. 9 The response time of query in the cluster

6 总结

由于天文数据的急速增长和海量观测数据的检索需要, 传统的关系型数据库在存储和检索海量数据时存在着诸多的不足, 本文研究了一种基于ElasticSearch分布式搜索引擎的天文海量FITS数据的高效检索方法. 通过存储FITS文件元数据的方式, 实现了FITS数据的统一存储; 利用ElasticSearch对索引的易扩展、易管理和易维护的特点来解决海量FITS数据的高效检索问题. 实验证明, 这种基于ElasticSearch的海量数据检索方法在检索效率和扩展性方面都有较大优势, 可以满足用户高效检索的需求. 论文研究的方法不仅仅适用于天文FITS数据, 也可以为其他天文数据格式提供参考和借鉴.

参 考 文 献

[1] Pence W D, Chiappetti L, Page C G, et al. A&A, 2010, 524: A42
[2] Lin Q, Lu X M, Jiang X J. NewA, 2013, 21: 33
[3] Andreeva J, Anjum A, Barrass T, et al. ITNS, 2005, 52: 884
[4] Xiao W, Ji C L, Li J D. AMM, 2013, 303: 2235
[5] Crestani F, Markov I. Advances in Information Retrieval, 2013, 7814: 865
[6] Berriman G B, Groom S L. Communications of the ACM, 2011, 54: 52
[7] 樊东卫, 崔辰州, 赵永恒. 天文研究与技术, 2011, 8: 306
[8] 王歆. 天文学报, 2013, 54: 382

[9] Wang X. ChA&A, 2014, 38: 211
[10] 刘应波, 王锋, 季凯帆, 等. 计算机应用研究, 2015, 32: 461
[11] 崔辰州, 李文, 于策, 等. 天文研究与技术, 2008, 5: 116
[12] 季凯帆, 曹文达, 宋谦. 云南天文台台刊, 1996: 60
[13] Rafał K, Marek R. ElasticSearch可扩展的开源弹性搜索解决方案. 时金桥, 柳厅文, 徐菲, 等, 译. 北京: 电子工业出版社, 2015: 25

The Application of ElasticSearch in the Massive Astronomical Data Retrieval

CHEN Ya-jie[1] WANG Feng[1,2] DENG Hui[1] LIU Ying-bo[1]
(1 Yunnan Key Laboratory of Computer Technology Application, Kunming University of Science and Technology, Kunming 650500)
(2 Yunnan Astronomical Observatories, Chinese Academy of Sciences, Kunming 650011)

Abstract Astronomical observational data are the fundamental element for modern astronomical researches. However, with the rapid increase of astronomical data, the traditional centralized retrieval methods are hard to meet the requirements of high-performance data retrieval. In the study, we present a novel method which is based on the ElasticSearch distributed retrieval engine and River mechanism to create data indexes, and provide high performance data retrieval for massive FITS (Flexible Image Transport System) data. We discuss the key technologies of the nearly real-time retrieval and query. The experimental results show that the method is capable of obtaining high retrieval performance especially for the cases in which the number of the FITS data exceeds millions or even tens of millions. Meanwhile, the method can be easily integrated into the current astronomical data archiving systems, and completely meet the archive requirements of all kinds of astronomical telescope systems.

Key words methods: numerical, methods: analytical, astronomical databases: miscellaneous