Nut

lucene+hadoop 分布式并行计算搜索框架

Alpha 9 文档

作者：曾年仔
mail:zengnianzai@163.com
blog: http://www.blogjava.net/nianzai/
code: http://code.google.com/p/nutla/

1、概述

不管程序性能有多高，机器处理能力有多强，都会有其极限。能够快速方便的横向与纵向扩展是Nut设计最重要的原则，以此原则形成以分布式并行计算为核心的架构设计。以分布式并行计算为核心的架构设计是Nut区别于Solr、Katta的地方。

Nut是一个Lucene+Hadoop分布式并行计算搜索框架，能对千G以上索引提供7*24小时搜索服务。在服务器资源足够的情况下能达到每秒处理100万次的搜索请求。

Nut开发环境：
jdk1.6.0.23+lucene3.0.3+eclipse3.6.1+hadoop0.20.2+zookeeper3.3.2+hbase0.20.6+memcached+mongodb+linux

2、特新

a、热插拔
b、可扩展
c、高负载
d、易使用,与现有项目无缝集成
e、支持排序
f、7*24服务
g、失败转移

3、搜索流程

Nut由Index、Search、Client、Cache和DB五部分构成。(Cache实现了对memcached的支持,DB实现了对hbase,mongodb的支持)
Client处理用户请求和对搜索结果排序。Search对请求进行搜索，Search上只放索引，数据存储在DB中，Nut将索引和存储分离。Cache缓存的是搜索条件和结果文档id。DB存储着数据，Client根据搜索排序结果,取出当前页中的文档id从DB上读取数据。

用户发起搜索请求给由Nut Client构成的集群，由某个Nut Client根据搜索条件查询Cache服务器是否有该缓存，如果有缓存根据缓存的文档id直接从DB读取数据，如果没有缓存将随机选择一组搜索服务器组(Search Group i),将查询条件同时发给该组搜索服务器组里的n台搜索服务器，搜索服务器将搜索结果返回给Nut Client由其排序，取出当前页文档id，将搜索条件和当前文档id缓存，同时从DB读取数据。

4、索引流程

Hadoop Mapper/Reducer 建立索引。再将索引从HDFS分发到各个索引服务器。
对索引的更新分为两种：删除和添加（更新分解为删除和添加）。

a、删除

在HDFS上删除索引，将生成的*.del文件分发到所有的索引服务器上去或者对HDFS索引目录删除索引再分发到对应的索引服务器上去。

b、添加

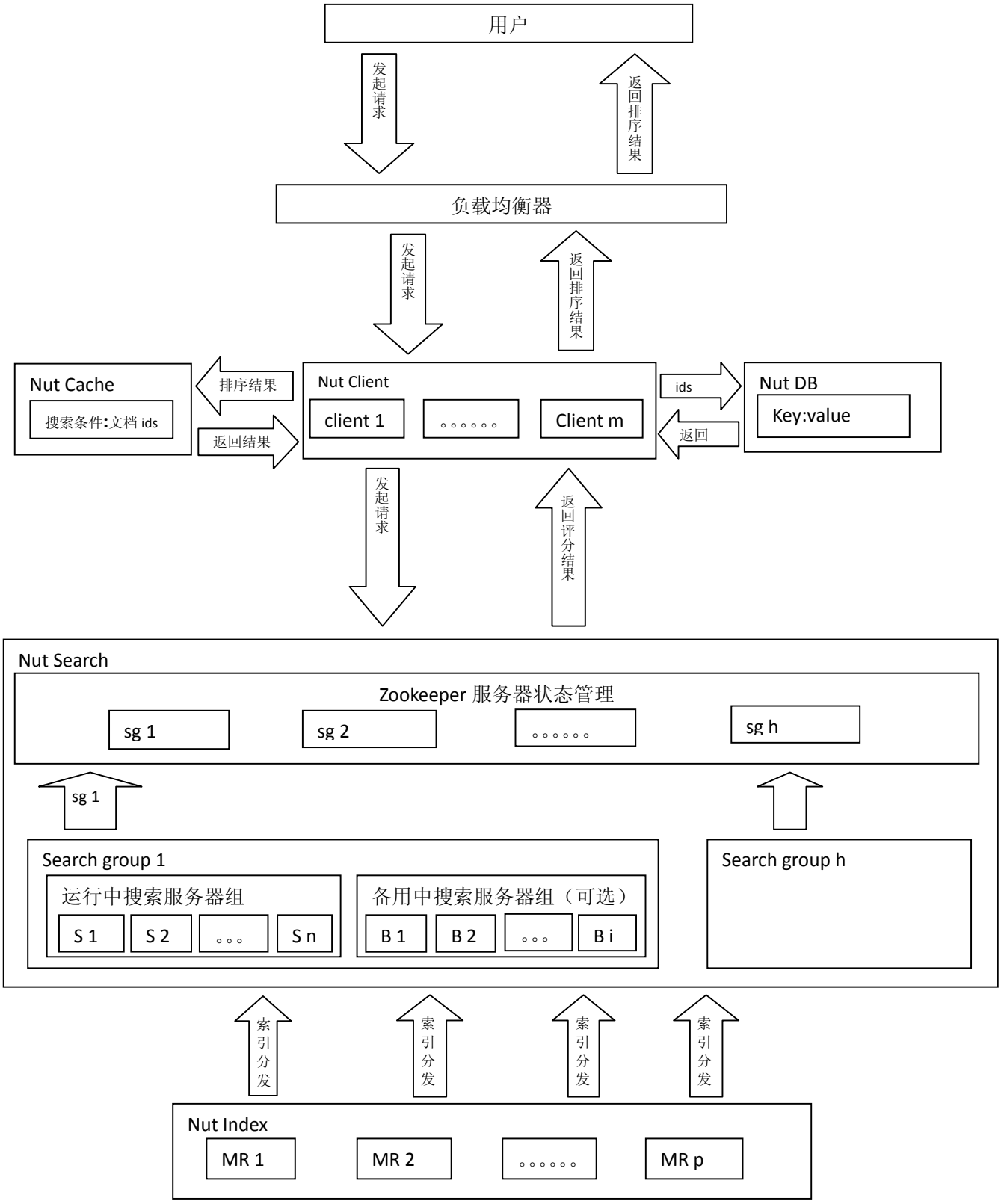
新添加的数据用另一台服务器来生成。

删除和添加步骤可按不同定时策略来实现。

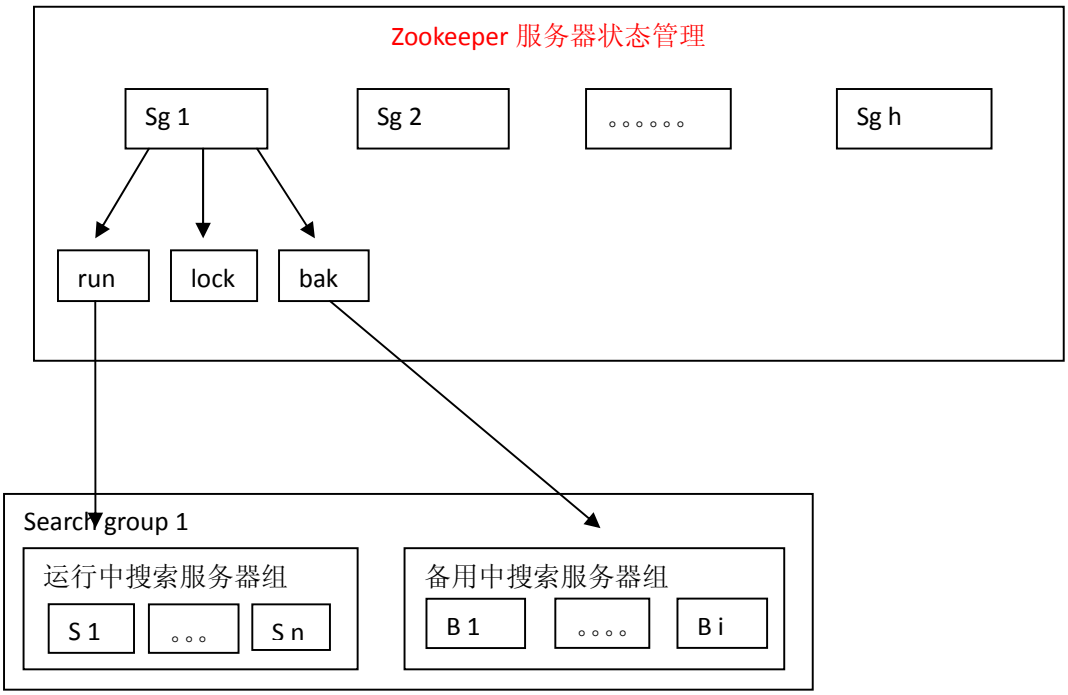
5、Nut分布式并行计算特点

Nut分布式并行计算虽然也是基于M/R模型，但是与Hadoop M/R模型是不同的。在Hadoop M/R模型中 Mapper和Reducer是一个完整的流程，Reducer依赖于Mapper。数据源通过Mapper分发本身就会消耗大量的I/O，并且是消耗I/O最大的部分。所以Hadoop M/R 并发是有限的。Nut M/R模型是将Mapper和Reducer分离，各自独立存在。在Nut中 索引以及索引管理 构成 M,搜索以及搜索服务器组 构成 R。
以一个分类统计来说明Nut分布式并行计算的流程。假设有10个分类，对任意关键词搜索要求统计出该关键词在这10个分类中的总数。同时假设有10组搜索服务器。索引以及索引管理进行索引数据的Mapper，这块是后台独自运行管理的。Nut Client将这10个分类统计分发到10组搜索服务器上，每组搜索服务器对其中一个分类进行Reducer，并且每组搜索服务器可进行多级Reducer。最后将最终结果返回给Nut Client。

6、设计图


用户
发起请求
返回排序结果
负载均衡器
发起请求
返回排序结果
Nut Cache
搜索条件:文档 ids
排序结果
返回结果
Nut Client
client 1
。。。。。。
Client m
ids
返回
Nut DB
Key:value
发起请求
返回评分结果
Nut Search
Zookeeper 服务器状态管理
sg 1
sg 2
。。。。。。
sg h
sg 1
Search group 1
运行中搜索服务器组
S 1
S 2
。。。
S n
备用中搜索服务器组（可选）
B 1
B 2
。。。
B i
Search group h
索引分发
索引分发
索引分发
索引分发
Nut Index
MR 1
MR 2
。。。。。。
MR p

7、Zookeeper服务器状态管理策略



在架构设计上通过使用多组搜索服务器可以支持每秒处理100万个搜索请求。
每组搜索服务器能处理的搜索请求数在1万—1万5千之间。如果使用100组搜索服务器，理论上每秒可处理100万个搜索请求。

假如每组搜索服务器有100份索引放在100台正在运行中搜索服务器(run)上，那么将索引按照如下的方式放在备用中搜索服务器(bak)上：index 1, index 2, index 3, index 4, index 5, index 6, index 7, index 8, index 9, index 10放在B 1 上，index 6, index 7, index 8, index 9, index 10, index 11, index 12, index 13, index 14, index 15放在B 2上。。。。。。index 96, index 97, index 98, index 99, index 100, index 5, index 4, index 3, index 2, index 1放在最后一台备用搜索服务器上。那么每份索引会存在3台机器中（1份正在运行中，2份备份中）。
尽管这样设计每份索引会存在3台机器中，仍然不是绝对安全的。假如运行中的index 1, index 2, index 3同时宕机的话，那么就会有一份索引搜索服务无法正确启用。这样设计，作者认为是在安全性和机器资源两者之间一个比较适合的方案。

备用中的搜索服务器会定时检查运行中搜索服务器的状态。一旦发现与自己索引对应的服务器宕机就会向lock申请分布式锁，得到分布式锁的服务器就将自己加入到运行中搜索服务器组，同时从备用搜索服务器组中删除自己，并停止运行中搜索服务器检查服务。

为能够更快速的得到搜索结果，设计上将搜索服务器分优先等级。通常是将最新的数据放

在一台或几台内存搜索服务器上。通常情况下前几页数据能在这几台搜索服务器里搜索到。如果在这几台搜索服务器上没有数据时再向其他旧数据搜索服务器上搜索。
优先搜索等级的逻辑是这样的：9最大为搜索全部服务器并且9不能作为level标识。当搜索等级level为1，搜索优先级为1的服务器，当level为2时搜索优先级为1和2的服务器，依此类推。

8、 web.xml配置

```
<!-- 启动zookeeper连接检查 -->
<servlet>
    <servlet-name>ZooKeeperInitServlet</servlet-name>
    <servlet-class>nut.nianzai.servlet.ZooKeeperInitServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<!-- 服务器状态显示 -->
<servlet>
    <servlet-name>StatusServlet</servlet-name>
    <servlet-class>nut.nianzai.servlet.StatusServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>StatusServlet</servlet-name>
    <url-pattern>/status.cgi</url-pattern>
</servlet-mapping>
```

9、搜索插件

a、通过实现QueryPlugin接口来实现自定义查询功能

```
public class ExampleQueryPlugin implements QueryPlugin
{
    @Override
    /**
     * 将自己要实现的查询功能在这实现
     */
    public Query query(String keyword) throws Exception
    {
        Analyzer analyzer = new ChineseAnalyzer();
        QueryParser parser1 = new QueryParser(Version.LUCENE_30,"title", analyzer);
        QueryParser parser2 = new QueryParser(Version.LUCENE_30,"descs", analyzer);
        Query query1 = parser1.parse(keyword);
        query1.setBoost(1.2f);
        Query query2 = parser2.parse(keyword);
        query2.setBoost(0.8f);
        BooleanQuery bq = new BooleanQuery();
        bq.add(query1, BooleanClause.Occur.MUST);
```

```
        bq.add(query2, BooleanClause.Occur.SHOULD);
        return bq;
    }

    @Override
    /**
     * 实现自己的排序功能
     */
    public Sort sort()
    {
        Sort sort=new Sort(new SortField("reviews", SortField.INT, true));
        return sort;
    }

    @Override
    /**
     * 实现自己的filter功能
     */
    public Filter filter()
    {
        return null;
    }
}
```

b、配置plugin.properties文件

```
#自定义查询插件名称=自定义查询插件实现类名
basicQuery=nut.nianzai.plugin. ExampleQueryPlugin
```

c、将自定义查询插件和plugin.properties打包生成nut-plugin.jar部署在每台服务的lib目录中

10、索引插件

a、通过实现IndexPlugin接口来实现自定义生成索引功能

```
public class ExampleIndexPlugin implements IndexPlugin
{
    private static Analyzer analyzer = new ChineseAnalyzer();

    @Override
    public void create(File file,Iterable<Text> values)
    {
        try
        {
            IndexWriter indexWriter = new
IndexWriter(NIOFSDirectory.open(file),analyzer,true,IndexWriter.MaxFieldLength.UNLIMITED)
```

```
;
            indexWriter.setRAMBufferSizeMB(256);
            indexWriter.setUseCompoundFile(false);

            for (Text str : values)
            {
                Document doc = new Document();
            String[] ss=str.toString().split("<<,>>");
            doc.add(new Field("id",ss[0], Field.Store.YES,Field.Index.NOT_ANALYZED));
            doc.add(new Field("title", ss[1], Field.Store.NO,Field.Index.ANALYZED));
            doc.add(new Field("descs", ss[2], Field.Store.NO,Field.Index.ANALYZED));
            doc.add(new Field("reviews",ss[3],Field.Store.YES,Field.Index.NOT_ANALYZED));
            indexWriter.addDocument(doc);
            }
            indexWriter.optimize();
            indexWriter.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    @Override
    public void merge(Directory[] dirs,Directory localWorkingDir) throws IOException
    {
        IndexWriter writer = new IndexWriter(localWorkingDir, analyzer,
true,IndexWriter.MaxFieldLength.UNLIMITED);
        writer.setRAMBufferSizeMB(256);
        writer.setUseCompoundFile(false);
        writer.addIndexesNoOptimize(dirs);
        writer.optimize();
        writer.close();
    }

    @Override
    public void delete(Directory dir, String ids)   throws IOException
    {
        IndexWriter writer = new
IndexWriter(dir,analyzer,false,IndexWriter.MaxFieldLength.UNLIMITED);
        writer.deleteDocuments(NutUtil.getTerms(ids));
        writer.close();
    }
```

b、配置hadoop.properties文件

```
 indexplugin=nut.nianzai.plugin.ExampleIndexPlugin
```

c、将自定义建索引插件和hadoop.properties文件打包成nut-index.jar放在hadoop环境中运行

11、HBase数据插入插件

a、通过实现HBasePlugin接口来实现自定义插入数据功能

```
public class ExampleHBasePlugin implements HBasePlugin
{
    @Override
    public Put insert(Text value)
    {
        String[] ss=value.toString().split("<<,>>");
        Put put = new Put(Bytes.toBytes(Integer.parseInt(ss[0])));
        put.add(Bytes.toBytes("title"), Bytes.toBytes(""), Bytes.toBytes(ss[1]));
        put.add(Bytes.toBytes("descs"), Bytes.toBytes(""), Bytes.toBytes(ss[2]));
        put.add(Bytes.toBytes("reviews"), Bytes.toBytes(""), Bytes.toBytes(ss[3]));
        return put;
    }
}
```

b、配置hadoop.properties文件

```
 indexplugin=nut.nianzai.plugin.ExampleHBasePlugin
```

c、将自定义建索引插件和hadoop.properties文件打包成nut-hbase.jar放在hadoop环境中运行

12、测试

```
public class NutTest
{
    //分布式搜索例子
    public static void main(String[] arg)
    {
        try
        {
            ZkCheck.start();
            Thread.sleep(6*1000);//等待6秒,用于通信初始化

            //构造查询对象
            Parameter parameter=new Parameter();
            parameter.setQuery("basicQuery");
            parameter.setKeyword("soho");
            parameter.setNo(1);
            parameter.setPs(2);
            parameter.setSortfield("reviews");//如果实现了排序，指明排序字段，需要根据这个来对结果进行排序

            //只统计
            int n=Client.parallelTotal(parameter);
            System.out.println(n);

            //搜索
            NutDB db=new MongoDB(House.class);
            NutCached cached=new MemCached();
            List<Object> ll=Client.parallelSearch(cached,db,parameter);
```

```
            System.out.println(parameter.getRscount());//返回搜索记录总数
            for(int i=0;i<ll.size();i++)
            {
                System.out.println(((House)ll.get(i)).getId());
                System.out.println(((House)ll.get(i)).getTitle());
                System.out.println(((House)ll.get(i)).getDescs());
                System.out.println(((House)ll.get(i)).getReviews());
            }
        }
        catch(Exception e)
        {e.printStackTrace();}
    }
}
```

13、服务器端部署

将nut压缩包解压部署在服务器上，根据服务操用途使用相应的启动服务命

a、运行搜索服务器启用命令nutserver.sh或nutserver.bat

b、备用搜索服务器启用命令checkserver.sh或checkserver.bat

14、用户管理工具

nut.sh(nut.bat)为命令行用户管理工具。

a、新建一个zookeeper根节点

```
sh nut.sh zk create 192.168.195.128:2181 nutzk
```

b、查看根节点nutzk下的子节点

```
sh nut.sh  zk list 192.168.195.128:2181 nutzk
```

c、删除根节点nutzk

```
sh nut.sh  zk delete 192.168.195.128:2181 nutzk
```

d、将目录（文件）从hdfs 复制到本地目录

```
sh nut.sh  copy h2l hdfs://192.168.195.128:9000/user/nianzai/nutindex/0/
/home/nianzai/index/
```

e、将目录（文件）从本地复制到hdfs

```
sh nut.sh copy l2h /home/nianzai/1.txt hdfs://192.168.195.128:9000/user/nianzai/input/
```

f、新建搜索节点

```
sh nut.sh zk searchgroup 192.168.195.128:2181 sg1
```