ElasticSearch的基本原理与用法

一、简介

ElasticSearch和Solr都是基于Lucene的搜索引擎，不过ElasticSearch天生支持分布式，而Solr是4.0版本后的SolrCloud才是分布式版本，Solr的分布式支持需要ZooKeeper的支持。

这里有一个详细的ElasticSearch和Solr的对比：http://solr-vs-elasticsearch.com/

二、基本用法

集群（Cluster）： ES是一个分布式的搜索引擎，一般由多台物理机组成。这些物理机，通过配置一个相同的cluster name，互相发现，把自己组织成一个集群。

节点（Node）：同一个集群中的一个Elasticsearch主机。

Node类型：

1）data node: 存储index数据。Data nodes hold data and perform data related operations such as CRUD, search, and aggregations.

2）client node: 不存储index，处理转发客户端请求到Data Node。

3）master node: 不存储index，集群管理，如管理路由信息（routing infomation），判断node是否available，当有node出现或消失时重定位分片（shards），当有node failure时协调恢复。（所有的master node会选举出一个master leader node）

详情参考：https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-node.html

主分片（Primary shard）： 索引（下文介绍）的一个物理子集。同一个索引在物理上可以切多个分片，分布到不同的节点上。分片的实现是Lucene 中的索引。

注意：ES中一个索引的分片个数是建立索引时就要指定的，建立后不可再改变。所以开始建一个索引时，就要预计数据规模，将分片的个数分配在一个合理的范围。

副本分片（Replica shard）： 每个主分片可以有一个或者多个副本，个数是用户自己配置的。ES会尽量将同一索引的不同分片分布到不同的节点上，提高容错性。对一个索引，只要不是所有shards所在的机器都挂了，就还能用。

索引（Index）：逻辑概念，一个可检索的文档对象的集合。类似与DB中的database概念。同一个集群中可建立多个索引。比如，生产环境常见的一种方法，对每个月产生的数据建索引，以保证单个索引的量级可控。

类型（Type）： 索引的下一级概念，大概相当于数据库中的table。同一个索引里可以包含多个 Type。

文档（Document）：即搜索引擎中的文档概念，也是ES中一个可以被检索的基本单位，相当于数据库中的row，一条记录。

字段（Field）： 相当于数据库中的column。ES中，每个文档，其实是以json形式存储的。而一个文档可以被视为多个字段的集合。比如一篇文章，可能包括了主题、摘要、正文、作者、时间等信息，每个信息都是一个字段，最后被整合成一个json串，落地到磁盘。

映射（Mapping）： 相当于数据库中的schema，用来约束字段的类型，不过 Elasticsearch 的 mapping 可以不显示地指定、自动根据文档数据创建。

Elasticsearch集群可以包含多个索引（indices），每一个索引可以包含多个类型（types），每一个类型包含多个文档（documents），然后每个文档包含多个字段（Fields），这种面向文档型的储存，也算是NoSQL的一种吧。

ES比传统关系型数据库，对一些概念上的理解：

```
Relational DB -> Databases -> Tables -> Rows -> Columns
Elasticsearch -> Indices   -> Types  -> Documents -> Fields
```

从创建一个Client到添加、删除、查询等基本用法：

1、创建Client

```
public ElasticSearchService(String ipAddress, int port) {
        client = new TransportClient()
                .addTransportAddress(new InetSocketTransportAddress(ipAddress,
                        port));
    }
```

这里是一个TransportClient。

ES下两种客户端对比：

TransportClient：轻量级的Client，使用Netty线程池，Socket连接到ES集群。本身不加入到集群，只作为请求的处理。

Node Client：客户端节点本身也是ES节点，加入到集群，和其他ElasticSearch节点一样。频繁的开启和关闭这类Node Clients会在集群中产生“噪音”。

2、创建/删除Index和Type信息

```
    // 创建索引
    public void createIndex() {
        client.admin().indices().create(new CreateIndexRequest(IndexName))
                .actionGet();
    }

    // 清除所有索引
    public void deleteIndex() {
        IndicesExistsResponse indicesExistsResponse = client.admin().indices()
                .exists(new IndicesExistsRequest(new String[] { IndexName }))
                .actionGet();
        if (indicesExistsResponse.isExists()) {
            client.admin().indices().delete(new DeleteIndexRequest(IndexName))
                    .actionGet();
        }
    }

    // 删除Index下的某个Type
    public void deleteType(){
client.prepareDelete().setIndex(IndexName).setType(TypeName).execute().actionGet();
    }

    // 定义索引的映射类型
    public void defineIndexTypeMapping() {
        try {
            XContentBuilder mapBuilder = XContentFactory.jsonBuilder();
            mapBuilder.startObject()
            .startObject(TypeName)
            .startObject("_all").field("enabled", false).endObject()
                .startObject("properties")
                    .startObject(IDFieldName).field("type", "long").endObject()
                    .startObject(SeqNumFieldName).field("type", "long").endObject()
                    .startObject(IMSIFieldName).field("type", "string").field("index",
"not_analyzed").endObject()
                    .startObject(IMEIFieldName).field("type", "string").field("index",
"not_analyzed").endObject()
                    .startObject(DeviceIDFieldName).field("type", "string").field("index",
"not_analyzed").endObject()
                    .startObject(OwnAreaFieldName).field("type", "string").field("index",
"not_analyzed").endObject()
                    .startObject(TeleOperFieldName).field("type", "string").field("index",
"not_analyzed").endObject()
                    .startObject(TimeFieldName).field("type", "date").field("store", "yes").endObject()
                .endObject()
            .endObject()
            .endObject();

            PutMappingRequest putMappingRequest = Requests
                    .putMappingRequest(IndexName).type(TypeName)
                    .source(mapBuilder);
            client.admin().indices().putMapping(putMappingRequest).actionGet();
        } catch (IOException e) {
            log.error(e.toString());
        }
```

这里自定义了某个Type的索引映射（Mapping）：

1）默认ES会自动处理数据类型的映射：针对整型映射为long，浮点数为double，字符串映射为string，时间为date，true或false为boolean。

2）字段的默认配置是indexed，但不是stored的，也就是 field("index", "yes").field("store", "no")。

公告

昵称： 阿凡卢
园龄： 8年10个月
粉丝： 1129
关注： 17
+加关注

积分与排名

积分 - 443553

排名 - 1054

随笔分类 (185)

Algorithm(29)
Big Data(17)
C#(11)
C/C++(19)
Data Structure(16)
Database(4)
Distributed System(14)
GIS(13)
Java(22)
Programming(30)
Python(3)
Research(7)

友情链接

酷壳
美团技术团队博客
阮一峰的网络日志
ThoughtWorks洞见

最新评论

1. Re:基于flink和drools的实时日志处理

博主可以呀

--消失的白桦林

2. Re:基于flink和drools的实时日志处理

@鱼非我欲 代码已经开源放到GitHub上了，就是个基于flink的项目...

--阿凡卢

3. Re:基于flink和drools的实时日志处理

您好，有详细的说明吗？实现的功能介绍，快速部署的步骤

--鱼非我欲

4. Re:基于Netty与RabbitMQ的消息服务

有问题请教下，请问您qq或者微信多少

--流沙1986

5. Re:flink基本原理

虽然看不太懂

--君君的喵爸

推荐排行榜

1. C#多线程编程(64)
2. RabbitMQ的几种典型使用场景(35)
3. 百度谷歌离线地图解决方案（离线地图下载）(31)
4. NPOI读写Excel(28)
5. 一个轻量级分布式RPC框架--NettyRpc(21)
6. ThreadLocal原理分析与使用场景(17)
7. ZooKeeper基本原理(17)
8. C#操作SQLite数据库(17)

3）这里Disable了“_all”字段，_all字段会把所有的字段用空格连接，然后用“analyzed”的方式index这个字段，这个字段可以被search，但是不能被retrieve。

4）针对string，ES默认会做“analyzed”处理，即先做分词、去掉stop words等处理再index。如果你需要把一个字符串做为整体被索引到，需要把这个字段这样设置：field("index", "not_analyzed")。

5）默认_source字段是enabled，_source字段存储了原始Json字符串（original JSON document body that was passed at index time）。

详情参考：

https://www.elastic.co/guide/en/elasticsearch/guide/current/mapping-intro.html

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-store.html

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-all-field.html

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-source-field.html

3、索引数据

```
// 批量索引数据
public void indexHotSpotDataList(List<Hotspotdata> dataList) {
    if (dataList != null) {
        int size = dataList.size();
        if (size > 0) {
            BulkRequestBuilder bulkRequest = client.prepareBulk();
            for (int i = 0; i < size; ++i) {
                Hotspotdata data = dataList.get(i);
                String jsonSource = getIndexDataFromHotspotData(data);
                if (jsonSource != null) {
                    bulkRequest.add(client
                            .prepareIndex(IndexName, TypeName,
                                    data.getId().toString())
                            .setRefresh(true).setSource(jsonSource));
                }
            }

            BulkResponse bulkResponse = bulkRequest.execute().actionGet();
            if (bulkResponse.hasFailures()) {
                Iterator<BulkItemResponse> iter = bulkResponse.iterator();
                while (iter.hasNext()) {
                    BulkItemResponse itemResponse = iter.next();
                    if (itemResponse.isFailed()) {
                        log.error(itemResponse.getFailureMessage());
                    }
                }
            }
        }
    }
}

// 索引数据
public boolean indexHotspotData(Hotspotdata data) {
    String jsonSource = getIndexDataFromHotspotData(data);
    if (jsonSource != null) {
        IndexRequestBuilder requestBuilder = client.prepareIndex(IndexName,
                TypeName).setRefresh(true);
        requestBuilder.setSource(jsonSource)
                .execute().actionGet();
        return true;
    }

    return false;
}

// 得到索引字符串
public String getIndexDataFromHotspotData(Hotspotdata data) {
    String jsonString = null;
    if (data != null) {
        try {
            XContentBuilder jsonBuilder = XContentFactory.jsonBuilder();
            jsonBuilder.startObject().field(IDFieldName, data.getId())
                    .field(SeqNumFieldName, data.getSeqNum())
                    .field(IMSIFieldName, data.getImsi())
                    .field(IMEIFieldName, data.getImei())
                    .field(DeviceIDFieldName, data.getDeviceID())
                    .field(OwnAreaFieldName, data.getOwnArea())
                    .field(TeleOperFieldName, data.getTeleOper())
                    .field(TimeFieldName, data.getCollectTime())
                    .endObject();
            jsonString = jsonBuilder.string();
        } catch (IOException e) {
            log.equals(e);
        }
    }

    return jsonString;
}
```

ES支持批量和单个数据索引。

4、查询获取数据

```
// 获取少量数据100个
private List<Integer> getSearchData(QueryBuilder queryBuilder) {
    List<Integer> ids = new ArrayList<>();
    SearchResponse searchResponse = client.prepareSearch(IndexName)
            .setTypes(TypeName).setQuery(queryBuilder).setSize(100)
            .execute().actionGet();
    SearchHits searchHits = searchResponse.getHits();
    for (SearchHit searchHit : searchHits) {
        Integer id = (Integer) searchHit.getSource().get("id");
        ids.add(id);
    }
    return ids;
}

// 获取大量数据
private List<Integer> getSearchDataByScrolls(QueryBuilder queryBuilder) {
    List<Integer> ids = new ArrayList<>();
    // 一次获取100000数据
    SearchResponse scrollResp = client.prepareSearch(IndexName)
            .setSearchType(SearchType.SCAN).setScroll(new TimeValue(60000))
            .setQuery(queryBuilder).setSize(100000).execute().actionGet();
    while (true) {
        for (SearchHit searchHit : scrollResp.getHits().getHits()) {
            Integer id = (Integer) searchHit.getSource().get(IDFieldName);
            ids.add(id);
        }
        scrollResp = client.prepareSearchScroll(scrollResp.getScrollId())
                .setScroll(new TimeValue(600000)).execute().actionGet();
        if (scrollResp.getHits().getHits().length == 0) {
            break;
        }
    }

    return ids;
}
```

这里的QueryBuilder是一个查询条件，ES支持分页查询获取数据，也可以一次性获取大量数据，需要使用Scroll Search。

5、聚合（Aggregation Facet）查询

```
// 得到某段时间内设备列表上每个设备的数据分布情况<设备ID，数量>
public Map<String, String> getDeviceDistributedInfo(String startTime,
        String endTime, List<String> deviceList) {

    Map<String, String> resultsMap = new HashMap<>();

    QueryBuilder deviceQueryBuilder = getDeviceQueryBuilder(deviceList);
    QueryBuilder rangeBuilder = getDateRangeQueryBuilder(startTime, endTime);
    QueryBuilder queryBuilder = QueryBuilders.boolQuery()
```

9. 基于GMap.Net的地图解决方案(15)

10. C++中的new、operator new与placement new(15)

```
                .must(deviceQueryBuilder).must(rangeBuilder);

        TermsBuilder termsBuilder = AggregationBuilders.terms("DeviceIDAgg").size(Integer.MAX_VALUE)
                .field(DeviceIDFieldName);
        SearchResponse searchResponse = client.prepareSearch(IndexName)
                .setQuery(queryBuilder).addAggregation(termsBuilder)
                .execute().actionGet();
        Terms terms = searchResponse.getAggregations().get("DeviceIDAgg");
        if (terms != null) {
            for (Terms.Bucket entry : terms.getBuckets()) {
                resultsMap.put(entry.getKey(),
                        String.valueOf(entry.getDocCount()));
            }
        }
        return resultsMap;
    }
```

Aggregation查询可以查询类似统计分析这样的功能：如某个月的数据分布情况，某类数据的最大、最小、总和、平均值等。

详情参考：https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/java-aggs.html

三、集群配置

配置文件elasticsearch.yml

集群名和节点名：

#cluster.name: elasticsearch

#node.name: "Franz Kafka"

是否参与master选举和是否存储数据

#node.master: true

#node.data: true

分片数和副本数

#index.number_of_shards: 5
#index.number_of_replicas: 1

允许其他网络访问:

network.host: 0

master选举最少的节点数，这个一定要设置为整个集群节点个数的一半加1，即N/2+1

#discovery.zen.minimum_master_nodes: 1

discovery ping的超时时间，拥塞网络，网络状态不佳的情况下设置高一点

#discovery.zen.ping.timeout: 3s

注意，分布式系统整个集群节点个数N要为奇数个！！

如何避免ElasticSearch发生脑裂（brain split）：http://blog.trifork.com/2013/10/24/how-to-avoid-the-split-brain-problem-in-elasticsearch/

即使集群节点个数为奇数，minimum_master_nodes为整个集群节点个数一半加1，也难以避免脑裂的发生，详情看讨论：https://github.com/elastic/elasticsearch/issues/2488

四、常用查询

curl -X<REST Verb> <Node>:<Port>/<Index>/<Type>/<ID>

Index info:

curl -XGET 'localhost:9200'
curl -XGET 'localhost:9200/_stats?pretty'
curl -XGET 'localhost:9200/{index}/_stats?pretty'
curl -XGET 'localhost:9200/_cluster/health?level=indices&pretty=true'
curl -XGET 'localhost:9200/{index}?pretty'
curl -XGET 'localhost:9200/_cat/indices?v'
curl -XGET 'localhost:9200/{index}/_mapping/{type}?pretty'

Mapping info:
curl -XGET 'localhost:9200/subscriber/_mapping/subscriber?pretty'

Index search:
curl -XGET 'localhost:9200/subscriber/subscriber/_search?pretty'

Search by ID:
curl -XGET 'localhost:9200/subscriber/subscriber/5000?pretty'

Search by field:
curl -XGET 'localhost:9200/subscriber/subscriber/_search?q=ipAddress:63.141.15.45&&pretty'

Delete index:
curl -XDELETE 'localhost:9200/subscriber?pretty'

Delete document by ID:
curl -XDELETE 'localhost:9200/subscriber/subscriber/5000?pretty'

Delete document by query:
curl -XDELETE 'localhost:9200/subscriber/subscriber/_query?q=ipAddress:63.141.15.45&&pretty'

五、基本原理

1、ES写数据原理

每个doc，通过如下公式决定写到哪个分片上：

shard= hash(routing) % number_of_primary_shards

Routing 是一个可变值，默认是文档的 _id ，也可以自定义一个routing规则。

默认情况下，primary shard在写操作前，需要确定大多数（a *quorum*, or majority）的shard copies是可用的。这样是为了防止在有网络分区（network partition）的情况下把数据写到了错误的分区。

A quorum是由以下公式决定：

int((primary + number_of_replicas) / 2) + 1，number_of_replicas是在index settings中指定的复制个数。

确定一致性的值有：one (只有primary shard)，all (the primary and all replicas)，或者是默认的quorum。

如果没有足够可用的shard copies，elasticsearch会等待直到超时，默认等待一分钟。

- 一个新文档被索引之后，先被写入到内存中，但是为了防止数据的丢失，会追加一份数据到事务日志（trans log）中。 不断有新的文档被写入到内存，同时也都会记录到事务日志中。这时新数据还不能被检索和查询。
- 当达到默认的刷新时间或内存中的数据达到一定量后，会触发一次 Refresh，将内存中的数据以一个新段形式刷新到文件缓存系统中并清空内存。这时虽然新段未被提交到磁盘，但是可以提供文档的检索功能且不能被修改。
- 随着新文档索引不断被写入，当日志数据大小超过 512M 或者时间超过 30 分钟时，会触发一次 Flush。 内存中的数据被写入到一个新段同时被写入到文件缓存系统，文件系统缓存中数据通过 Fsync 刷新到磁盘中，生成提交点，日志文件被删除，创建一个空的新日志。

2、ES读数据原理

Elasticsearch中的查询主要分为两类，Get请求：通过ID查询特定Doc；Search请求：通过Query查询匹配Doc。

- 对于Search类请求，查询的时候是一起查询内存和磁盘上的Segment，最后将结果合并后返回。这种查询是近实时（Near Real Time）的，主要是由于内存中的Index数据需要一段时间后才会刷新为Segment。
- 对于Get类请求，查询的时候是先查询内存中的TransLog，如果找到就立即返回，如果没找到再查询磁盘上的TransLog，如果还没有则再去查询磁盘上的Segment。这种查询是实时（Real Time）的。这种查询顺序可以保证查询到的Doc是最新版本的Doc，这个功能也是为了保证NoSQL场景下的实时性要求。

所有的搜索系统一般都是两阶段查询，第一阶段查询到匹配的DocID，第二阶段再查询DocID对应的完整文档，这种在Elasticsearch中称为query_then_fetch，还有一种是一阶段查询的时候就返回完整Doc，在Elasticsearch中称作query_and_fetch，一般第二种适用于只需要查询一个Shard的请求。除了一阶段，两阶段外，还有一种三阶段查询的情况。搜索里面有一种算分逻辑是根据TF（Term Frequency）和DF（Document Frequency）计算基础分，但是Elasticsearch中查询的时候，是在每个Shard中独立查询的，每个Shard中的TF和DF也是独立的，虽然在写入的时候通过_routing保证Doc分布均匀，但是没法保证TF和DF均匀，那么就有会导致局部的TF和DF不准的情况出现，这个时候基于TF、DF的算分就不准。为了解决这个问题，Elasticsearch中引入了DFS查询，比如DFS_query_then_fetch，会先收集所有Shard中的TF和DF值，然后将这些值带入请求中，再次执行query_then_fetch，这样算分的时候TF和DF就是准确的，类似的有DFS_query_and_fetch。这种查询的优势是算分更加精准，但是效率会变差。另一种选择是用BM25代替TF/DF模型。

在新版本Elasticsearch中，用户没法指定DFS_query_and_fetch和query_and_fetch，这两种只能被Elasticsearch系统改写。

3、ES选主（select master）

ES的master选举原理如下：

1. 对所有可以成为master的节点根据nodeId排序，每次选举每个节点都把自己所知道节点排一次序，然后选出第一个（第0位）节点，暂且认为它是master节点。

2. 如果对某个节点的投票数达到一定的值（可以成为master节点数n/2+1）并且该节点自己也选举自己，那这个节点就是master。否则重新选举。
3. 对于brain split问题，需要把候选master节点最小值设置为可以成为master节点数n/2+1（quorum）

六、Elasticsearch插件

1、elasticsearch-head是一个elasticsearch的集群管理工具：./elasticsearch-1.7.1/bin/plugin -install mobz/elasticsearch-head

github地址：https://github.com/mobz/elasticsearch-head

2、elasticsearch-sql：使用SQL语法查询elasticsearch：./bin/plugin -u https://github.com/NLPchina/elasticsearch-sql/releases/download/1.3.5/elasticsearch-sql-1.3.5.zip --install sql

github地址：https://github.com/NLPchina/elasticsearch-sql

3、elasticsearch-bigdesk是elasticsearch的一个集群监控工具，可以通过它来查看ES集群的各种状态。

安装：./bin/plugin -install lukas-vlcek/bigdesk

访问：http://192.103.101.203:9200/_plugin/bigdesk/

github地址：https://github.com/hlstudio/bigdesk

4、elasticsearch-servicewrapper插件是ElasticSearch的服务化插件

https://github.com/elasticsearch/elasticsearch-servicewrapper

DEPRECATED: The service wrapper is deprecated and not maintained. 该项目已不再维护。

例子代码在GitHub上：https://github.com/luxiaoxun/Code4Java

参考：

https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/index.html

https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

https://www.elastic.co/guide/en/elasticsearch/guide/current/distrib-write.html

http://stackoverflow.com/questions/10213009/solr-vs-elasticsearch

分类： Big Data， Distributed System

标签： Elasticsearch， 大数据， 分布式

« 上一篇： Solr与MySQL查询性能对比
» 下一篇： ZooKeeper基本原理

posted @ 2015-10-11 16:35 阿凡卢 阅读(53101) 评论(9) 编辑 收藏 举报