Analysis of load in single SOLR server and SOLR cloud with Zookeeper

schmittjoaopedro computer science, Data Mining, Java, Statistics, Windows 7 07+00:00 Dezembro 07+00:00 2018 6 Minutes

The objective of this study is to give directions about a methodology used to evaluate the best usage of SOLR in a project. To achieve this objective will be presented a study case in a private company. This study had evaluated how SOLR behaved under different architectures, a single SOLR server instance and a SOLR cloud architecture with zookeeper. The analysis evaluated the performance in query response time and indexes memory consumption. As this analysis was conducted in a private company, is not possible to reproduce the tests because the test data is not public.

(https://github.com/schmittjoaopedro/joaoschmitt.wordpress.com/tree/master/solr/architecture-load-evaluation#indexing-data-with-byte-fields)Indexing data with byte fields.

SOLR as a search engine allows to extrapolate their usage and apply binary fields to index files with their metadata to speed-up the searches. Following is presented an illustrative schema of a document that we want to index:

```
{
    "id": projectId + " - " + revision,
    "projectId": Int,
    "revision": Int,
    "notes": String,
    "status": Int,
    "categoyId": Int,
    "dateUpdated": Date,
    "dateCreated": Date,
    "active": Bool,
    "numeric_characteristics": [],
    "string_characteristics": [],
    "binary_characteristics": []
}
```

Algorithm 1: Object schema

Utilizing the schema of *Algorithm 1*, we created JSON files with around 200 characteristics of text and numeric types and 5 binary characteristics. To make the lists dynamic the vector fields were configured as dynamic fields. In SOLR the binary files are converted to Base64, therefore is important to evaluate how the binary fields influence in the index. To do this task we have created the following packets of JSON files:

- Packed 1 = 4 JSON documents resulting in a packet with 3.6 MB. In this packet binary files presents around 90% of the size.
- Packet 2 = Packet 1 + 53 JSON documents resulting in a packet with 5.1 MB. In this packet binary files represents around 70% of the size.

To evaluate how a single SOLR server instance behaves indexing the packets, next is presented the statistics obtained about the resource usage.

(https://github.com/schmittjoaopedro/joaoschmitt.wordpress.com/tree/master/solr/architecture-load-evaluation#memory-usage)Memory usage

- Packet 1 with 160.000 documents = 440 MB
- Packet 2 with 100.000 documents = 590 MB
- Packet 2 with 200.000 documents = 400 MB
- Packet 2 with 500.000 documents = 400 MB

Related to memory size was not perceived significative differences given by the number and size of documents. This memory was analyzed with the server in idle mode (not processing queries).

(https://github.com/schmittjoaopedro/joaoschmitt.wordpress.com/tree/master/solr/architecture-load-evaluation#disk-usage)Disk usage

- Packet 1 with 160.000 documents = 94.7 GB
- Packet 2 with 100.000 documents = 8.06 GB
- Packet 2 with 200.000 documents = 13.2 GB
- Packet 2 with 500.000 documents = 28.6 GB

Related to disk usage, as expected, a huge amount of space is required to index the JSON objects. In special for Packet 1, where the binary files represent 90% of the JSON object.

(https://github.com/schmittjoaopedro/joaoschmitt.wordpress.com/tree/master/solr/architecture-load-evaluation#query-time)Query time

Next is present the query times considering different combinations of criterias:

- Scenario 1 – filtering by project id
  - Packet 1 with 160.000 documents = 44 ms
  - Packet 2 with 100.000 documents = 26 ms
  - Packet 2 with 200.000 documents = 5 ms
  - Packet 2 with 500.000 documents = 6 ms
- Scenario 2 – filtering by 3 string characteristic with a numeric characteristics
  - Packet 1 with 160.000 documents = 233 ms
  - Packet 2 with 100.000 documents = 24 ms
  - Packet 2 with 200.000 documents = 17 ms
  - Packet 2 with 500.000 documents = 31 ms
- Scenario 3 – filtering by 4 string characteristics
  - Packet 1 with 160.000 documents = 26 ms
  - Packet 2 with 100.000 documents = 5 ms
  - Packet 2 with 200.000 documents = 7 ms
  - Packet 2 with 500.000 documents = 10 ms
- Scenario 4 – filtering by 2 numeric field and 6 string fields
  - Packet 1 with 160.000 documents = 5 ms
  - Packet 2 with 100.000 documents = 21 ms
  - Packet 2 with 200.000 documents = 10 ms
  - Packet 2 with 500.000 documents = 227 ms

Related to the query time tests, was not perceived a significative difference between the results. In some cases, for the Packet 1 the time had achieved around 200 ms, but for the user, it still is a fast response time to be considered critical.

The final analysis about the usage of Base64 is that, for a huge amount of documents, it is very disk consuming to be indexed by Solr. In this case, if many files must be indexed would be a better strategy store this files in an external system that is more efficient to this purpose.

(https://github.com/schmittjoaopedro/joaoschmitt.wordpress.com/tree/master/solr/architecture-load-evaluation#comparing-single-solr-with-solr-cloud)Comparing Single SOLR with SOLR cloud

In the SOLR Cloud, the Zookeeper is used to keep the servers synchronized. The objective of Zookeeper is to be used to: notify when new documents need to be indexed, centralize configuration files, elected the server leader, etc. In most part Zookeeper works asynchronous. To install Zookeeper we refer to this tutorial: Installing zookeeper (https://medium.com/@shaaslam/installing-apache-zookeeper-on-windows-45eda303e835).

To start the Zookeeper at least three servers must be up. This configuration guarantee that sufficient servers will be available to create redundancy. In this article, a single server was used for validation purposes, but in real scenarios is important to consider the indications of the fabricant. To configure zookeeper for our tests, in the Zookeeper installation directory we had renamed the file zoo_sample.cfg to zoo.cfg, and after that, we started the Zookeeper with:

```
zookeeper_dir > cd bin zookeeper_dir > zkServer.cmd
```

After that, we downloaded the SOLR servers and unzipped them in a specific directory. Then we copied the solr.xml file to zookeeper using the following command:

```
solr_dir > bin\solr zk cp file:E:\solr\solr-7.3.1\server\solr\solr.xml zk:/solr.xml -z localhost:2181
```

This command will share in zookeeper the solr.xml configuration file. By this way, the new nodes that enter in the cluster will obtain the configuration file from zookeeper. After that, was copied the collection files to zookeeper. This files will be used as template by the new nodes that enter in the cluster.

```
solr_dir > bin\solr zk upconfig -n your-solr-project -d C:\projetos\solr7\your-solr-projects -z
localhost:2181
```

To execute this tests two personal computers were used. In each computer, two SOLR instances were created to make a cluster with four nodes. More details about the SOLR initialization and configuration are available in the following tutorial: SOLR COULD + ZOOKEEPER + WINDOWS (https://joaoschmitt.wordpress.com/2018/07/21/solr-cloud-zookeeper-windows/).

To create the test database, JSON files with around 28 kb of size were used, these files don't have any kind of binary data. To load the files to the SOLR instances the following code, using the spring-data API, was used:

```
int start = 0;
int samplesNumber = 250000;
int chunkSize = 100;
samplesNumber += start;
ObjectMapper mapper = new ObjectMapper();
File dir = new File(Paths.get(rootPath.toString(), "dump").toString());
File[] files = FileUtils.listFiles(dir, new String[]{"json"}, false).toArray(
Map<SampleProject, Long> ovIds = new HashMap<>();
Map<SampleProject, Long> ohIds = new HashMap<>();
Map<SampleProject, String> docIds = new HashMap<>();
List<SampleProject> projects = new ArrayList<>(files.length);

for (int i = 0; i < files.length; i++) {
    projects.add(mapper.readValue(files[i], SampleProject.class));
    projects.get(i).getBinaryValues().clear();
    ovIds.put(projects.get(i), projects.get(i).getProjectId());
    ohIds.put(projects.get(i), projects.get(i).getProjectCategoryId());
    docIds.put(projects.get(i), projects.get(i).getId());
}

List<SampleProject> toSave = new ArrayList<>();
for (int i = start; i < samplesNumber; i++) {
    for (SampleProject project : projects) {
        project.setId(docIds.get(project) + "-" + i);
        project.setProjectId(i + ovIds.get(project));
        project.setProjectCategoryId(i + ohIds.get(project));
        project.setLastVersion(false);
        toSave.add((SampleProject) project.clone());
    }
    if (i % chunkSize == 0) {
        long time = System.currentTimeMillis();
        sampleProjectRepository.saveAll(toSave);
        System.out.println("Samples loaded " + ((i + 1) * files.length) + "
        toSave.clear();
   }
}
```

To generate the database was used the bulk persistence, this kind of strategy reduces the commit time and the cluster synchronization time. In the firsts, tests were used bulks with 4 documents, but the performance was very poor, a better performance was achieved with bulks of 400 documents. This bulk size reduced in 1/6 the data loading time. The average time of the commits was about 6.7 seconds.

(https://github.com/schmittjoaopedro/joaoschmitt.wordpress.com/tree/master/solr/architecture-load-evaluation#solr-cloud-results)Solr cloud results

The first analysis carried was about the size of the index, the next table presents the results for two amount of data.

Num docs	PC 1 – shard1	PC 1 – shard3	PC 2 – shard2	PC 2 – shard4
250.000	110.92 MB	110.34 MB	114.39 MB	114.33 MB
1.000.000	381.44 MB	370.4 MB	385.01 MB	383.23 MB

Based on the shards size is possible to see that the JSON objects well distributed around the nodes. With respect to query times we considered the execution time of first request after starting the servers:

- Scenario 1 – Filter by project ID
  - 250.000 documentos = 242 ms
  - 1.000.000 documentos = 400 ms
- Scenario 2 – Filter by 1 numerical characteristic and 3 string characteristics
  - 250.000 documentos = 438 ms
  - 1.000.000 documentos = 705 ms
- Scenario 3 – Filter by 4 string characteristics
  - 250.000 documentos = 69 ms
  - 1.000.000 documentos = 54ms
- Scenario 4 – Filter by 2 numeric characteristics and 5 string characteristics
  - 250.000 documentos = 145 ms
  - 1.000.000 documentos = 823 ms
- Scenario 5 – Filter all, limited to 10 rows
  - 250.000 documentos = 50 ms
  - 1.000.000 documentos = 21 ms

(https://github.com/schmittjoaopedro/joaoschmitt.wordpress.com/tree/master/solr/architecture-load-evaluation#single-solr-results)Single Solr results

To compare the single Solr server architecture against the Solr cloud architecture, the next results are related to a single Solr server. Related to the amount of data we have:

Num. docs	PC 1

Num. docs	PC 1
250.000	396.91 MB
1.000.000	1.44 GB

And related to the query times we have:

- Scenario 1 – Filter by project ID
  - 250.000 documentos = 23 ms
  - 1.000.000 documentos = 5 ms
- Scenario 2 – Filter by 1 numerical characteristic and 3 string characteristics
  - 250.000 documentos = 25 ms
  - 1.000.000 documentos = 22 ms
- Scenario 3 – Filter by 4 string characteristics
  - 250.000 documentos = 8 ms
  - 1.000.000 documentos = 13 ms
- Scenario 4 – Filter by 2 numeric characteristics and 5 string characteristics
  - 250.000 documentos = 24 ms
  - 1.000.000 documentos = 27 ms
- Scenario 5 – Filter all, limited to 10 rows
  - 250.000 documentos = 3 ms
  - 1.000.000 documentos = 10 ms

(https://github.com/schmittjoaopedro/joaoschmitt.wordpress.com/tree/master/solr/architecture-load-evaluation#conclusions)Conclusions

Based on the results obtained we can draw the following conclusions.

Related to the Base64 indexation, it is not recommended if there are many documents to be indexed. As observed in our analysis indexing 160.000 documents with each document having around 900Kb of size caused by binary data, results in an index with around 90 GB of disk space consumption. The main problem here is related to disk size because as observed, the query time and memory usage were not affected by Base64 fields.

Concerning the Solr cloud vs single server architecture, there is an expensive time spent in the synchronization process to apply queries among the nodes of the cluster. Therefore, if a cluster strategy is not well defined or if you have a simple scenario that a single server is sufficient don't aggregate such a complexity in the project. In the end, a single server can delivery query response time faster for scenarios similar to the scenario studied here.

Com as etiquetas :
java,
load,
solr,
solr-cloud

Publicado por schmittjoaopedro

Graduado como bacharel em Sistemas de Informação pelo Centro Universitário Católica de Santa Catarina campus Jaraguá do Sul. Formado no Ensino Médio pelo Senai com Técnico em Redes de Computadores Articulado. Atualmente desenvolvedor JEE/Web em Sistemas de Engenharia na WEG. Pesquisador no período de faculdade em Informática pela Católica de Santa Catarina. Contato 47 - 99615 2305 E-mail: schmittjoaopedro@gmail.com Web page: https://joaoschmitt.wordpress.com/ Linkedin: https://www.linkedin.com/in/joao-pedro-schmitt-60847470/ Curriculum lattes: http://lattes.cnpq.br/9304236291664423 Twitter: @JooPedroSchmitt *Ver todos os artigos de schmittjoaopedro*

One thought on “Analysis of load in single SOLR server and SOLR cloud with Zookeeper”

QQPokerRaja diz:
15 15+00:00 Abril 15+00:00 2019 às 10:38 | Editar
I love looking through an article that can make people think.
Also, thank you for allowing for me to comment!

Responder