Analysis and Enhancements of Index Based Hibernate Search Applications

[1]Mr. Prashant Singh, [2]Mr. P N Barwal
[1] Project Engineer, [2]Joint Director
[1, 2,] Centre for Development of Advanced Computing, Noida, India
[1]prashantsingh@cdac.in, [2]pnbarwal@cdac.in

Abstract— this paper deals with the research in implementing an Index based Hibernate Search Utility for web applications. Hibernate is an open source persistence framework that is based on ORM (Object/Relational Mapping). Often web applications require the need of a search utility that can search exact words or phrase within text, order results by relevance and find similar results based on approximations. Hibernate search coupled with a few techniques like lazy load, NoSQL and Clustering can help answering these requirements in a considerably faster and scalable way. In this paper, by the means of an application we demonstrate how the Hibernate Search works and then how its performance can be increased to meet the requirements of a business application. Finally we compare the results quantitatively with a traditional Hibernate Search based application.

Keywords—Hibernate, Full-text Search, Indexing, Web Application, NoSQL, Clustering.

1.0 INTRODUCTION TO HIBERNATE

Web applications nowadays have to work with huge amount of data to meet the requirement of growing web based development. These requirement lead to numerous data transactions between the application and the database layers. All the requests to the web application are served through these transactions. This often results in hampering the performance of the web application in terms of interacting with the database layer. To cater these developing needs developers started using and additional persistence layer in between the application and the database layer. This layer stores the data that is frequently accessed in a way that can be reached by the application frequently and quickly. This persistence layer, since it needs to interact with the application and the database layer, should be easy to code at the application level. Java developers use Hibernate for this purpose.

Hibernate is an object-relational mapping library for Java developers. Object where refers to the application interface where objects are created in the form of POJOs (Plain Old Java Objects) or Classes. These objects are mapped with the relations or tables of the database using Hibernate. It is easy to work with hibernate bec ause it accesses the database using high level object handling functions over traditional persistence-related database accesses [1] It's a freeware. Hibernate not only maps java classes to database relations, but it also maps java data types to that of the database. Moreover it can easily connect to most of the databases and the user can query these databases using HQL (Hibernate Query Language) which isConverted to the database dialect at runtime by the Hibernate.

Thus, Hibernate ORM (Object/Relational Mapping) provides us with features to enhance the performance of an application. Features like Lazy initialization that initializes the object with a value only when its value is actually required thus saving time and memory both. By using various fetching strategies Hibernate can be customized to work according to the needs of the application. It is designed so as to deliver a highly scalable architecture as it can easily work on a cluster of servers.

Apart from the Hibernate ORM, Hibernate offers various other modules of which the ones we'll be referring to in this paper include Hibernate Search, Hibernate Validator and Hibernate OGM.

In this paper, we study the details of Hibernate and Hibernate Search and by the means of an application, we define the functionality of Hibernate Search and how can it be coupled with other technologies to deliver a high performance web based application.

This paper is outlined as follows. *Section I* provides an introduction to Hibernate and also covers its basic components and modules of Hibernate that we'll be dealing with in this paper. *Section II* describes the architecture of hibernate based search application. This section also covers how we implement Hibernate search traditionally in the Hibernate application. *Section III* talks about the new technologies that can be coupled with the Hibernate Search to enhance its performance. *Section IV* provides a quantitative comparison of the performance of Hibernate Search with that of its enhanced version proposed in this paper. This section also elucidates the uses of such an enhanced application in context of various web applications that rely on 'Searches'. Finally, in *Section V* we conclude, describing the novelty of this study and putting forth its limitations and the scope for future work on Hibernate and its search based applications.

2.0 HIBERNATE ARCHITECTURE

Hibernate architecture is layered to separate its internal programming interfaces from each other. It uses the database and the configuration files to provide the persistence layer interacting between the application and the database layer as depicted in Figure 1.

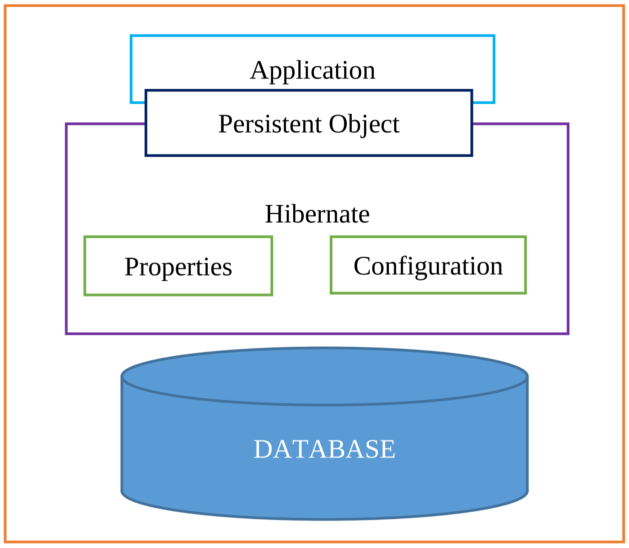


Figure 1: Hibernate Architecture

The main components of its implementation include connection management, transaction management, implementing the ORM by making classes called as 'Entities' to represent the database relations. Then writing the HQL (Hibernate Query Language) to perform data definition and data manipulation operations on the database by simply persisting the java objects in the database.

2.1 HIBERNATE SEARCH

Hibernate Search is a full-text search support for objects stored by Hibernate ORM. It can be used to find words in the text and also order the search results based on the relevance. It uses index based search which is a faster medium of accessing data from the database. It can also find words by approximation which is called fuzzy searching. For example "nw" would search for "nw", "new", "now" and many more. This type of search depends upon the limit of approximations mentioned. It also offers the search using clusters, hence enhancing the performance of the application. It can cluster the indexes using master slave architecture. It can also be used to find results around a certain location, as the 'Entities' can be geolocalized easily with hibernate. It can be used to categorize results based on the price range or bands. Apart from these features what puts it on top is its ease of use as it manages indexes, clusters, synchronizes the tasks seamlessly while the developer only focusses on the application development logic.

Figure 2: Traditional Hibernate Search

Full text search based engines like Apache Lucene also offers free and extensive index based search facilities, however it is not very convenient when used in coupling with and ORM. Because while making an index based search it is very important to keep the indexes up to date and to avoid mismatching in their mapping with the objects. Hibernate search, on the contrary, addresses these issues by indexing the 'Entities' with the help of a few annotations, synchronizes the data automatically and fetches regular objects. Hibernate search however uses Apache Lucene below it.

2.2 INDEXING

Indexing refers to the complete process of capturing data in the form of plain text, building documents, analyzing the data and finally creating indexes. A "Document" here refers to the most fundamental element which contains the raw data in this process. It can be called as a collection of fields. The common literals are filtered initially to increase the content of searchable data. Finally, these documents stored and attached physically using an Index Writer inbuilt in the Lucene.

To set up hibernate search we need to add a few configuration files to pre-setup hibernate application. Then the next step is to add indexes to the entity objects, this can be done using the following annotations:

```
@Entity
@Table (name = "table_name")
@Indexed (index = "name")
@Field (name = "name", analyze= Analyze.YES, store = Store.YES)
```

@Entity and @Table are simply hibernate annotations that are used to refer the java class as a database relation. Whereas @Indexed is used to create the index of that object and @Field annotations are added for the index of an attribute of the relation. This annotation also tells hibernate whether to store the index in memory or not. Thus, these annotations are specific for the searching utility.

We then create an entity manager to interact with the database layer as follows:

```
@Autowired
@PersistenceContext
private EntityManager entityManager;
```

Then we create a FullTextEntityManager to convert the EntityManager to use full text search as

```
FullTextEntityManager fullTextEntityManager =
org.hibernate.search.jpa.Search.getFullTextEntityManager
(entityManager);
```

Then we create a QueryContextBuilder to create the full text queries:

```
QueryContextBuilder queryContextBuilder =
fullTextEntityManager.getSearchFactory ().buildQueryBuilder ();
```

This QueryContextBuilder is then used to create an Entity Context.

```
EntityContext entityContext = QueryContextBuilder.forEntity
(entityClass);
```

Finally, we write query as follows:

```
org.apache.lucene.search.Query titleQuery1 =
entityContext.get ().qb.keyword ().fuzzy ().onFields ("field1","
field2"," field3"," field4").matching (searchInput).createQuery ();
```

Here, keyword () is used to search a particular word referred to as the “searchInput”. Fuzzy () allows search based on approximation that the words similar to searchInput but not exactly equal to the search input will be a part of the results fetched by the query. OnFields () is used to search on the indexes created on field1, field2, field3 and field4 using the @Field annotation mentioned above.

This query is then converted to a full text query using the following snippet:

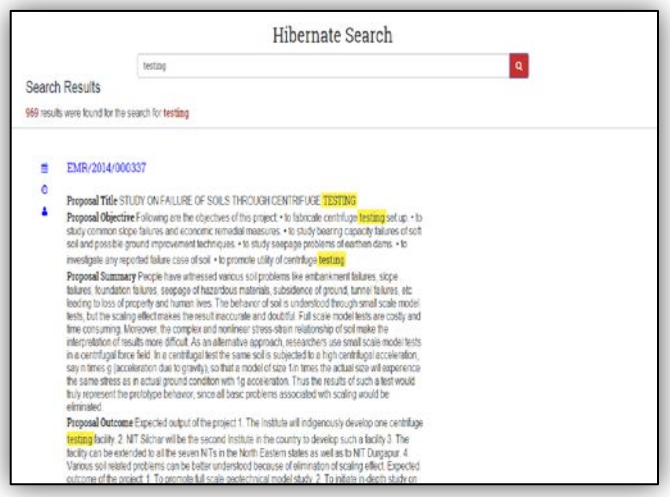
```
Javax.persistence.Query fullTextQuery =
fullTextEntityManager.createFullTextQuery (titleQuery1);
```

```
fullTextQuery.setFirstResult (n);
```

This enables the search to bring results starting from n.

```
fullTextQuery.setMaxResults (max);
```

This enables the search to bring max results from the database search.


Hibernate Search

testing

Search Results

969 results were found for the search for testing

EMR/2014/000337

Proposal Title STUDY ON FAILURE OF SOILS THROUGH CENTRIFUGE TESTING

Proposal Objective Following are the objectives of this project • to fabricate centrifuge testing set up. • to study common slope failures and economic remedial measures. • to study bearing capacity failures of soft soil and possible ground improvement techniques. • to study seepage problems of earthen dams. • to investigate any reported failure case of soil • to promote utility of centrifuge testing

Proposal Summary People have witnessed various soil problems like embankment failures, slope failures, foundation failures, seepage of hazardous materials, subsidence of ground, tunnel failures, etc leading to loss of property and human lives. The behavior of soil is understood through small scale model tests, but the scaling effect makes the result inaccurate and doubtful. Full scale model tests are costly and time consuming. Moreover, the complex and nonlinear stress-strain relationship of soil make the interpretation of results more difficult. As an alternative approach, researchers use small scale model tests in a centrifugal force field. In a centrifugal test the same soil is subjected to a high centrifugal acceleration, say n times g (acceleration due to gravity), so that a model of size 1/n times the actual size will experience the same stress as in actual ground condition with 1g acceleration. Thus the results of such a test would truly represent the prototype behavior, since all basic problems associated with scaling would be eliminated.

Proposal Outcome Expected output of the project 1. The Institute will indigenously develop one centrifuge testing facility. 2. NIT Silchar will be the second Institute in the country to develop such a facility 3. The facility can be extended to all the seven NITs in the North Eastern states as well as to NIT Durgapur. 4. Various soil related problems can be better understood because of elimination of scaling effect. Expected outcome of the project 1. To promote full scale geotechnical model study. 2. To initiate in-depth study on

Figure 3: Tradition Hibernate Search Results

Thus, we have traditionally made a search application that can search through all the text data of the web application. As we can see in the Figure 3, the results are shown for the word “testing” and out of the order of 10,000 results in the database the search brings about approximately 900 results.

3.0 ENHANCED HIBERNATE SEARCH

Hibernate search alone provides a decent search mechanism that uses the features of Hibernate for the object relation mapping coupled with the full text index based search. It itself promotes loading results in batches. However, by using a few more technologies on top of it can help increase the usage of hibernate search on a large scale enterprise application. Technologies. We now give a brief of the technologies that we have coupled the traditional hibernate search to enhance and scale its performance:

3.1 LAZY LOADING

This is provided by hibernate itself. Lazy Loading refers to the method of fetching an entity’s associated options only when they are actually requested by the framework. To enable lazy loading explicitly you must use "fetch = FetchType.LAZY" on an association which you want to lazy load when you are using hibernate annotations.

For example:

```
@OneToMany( mappedBy = "person", fetch = FetchType.LAZY )
private Set<PersonEntity> persons;
```

Hibernate provides a proxy implementation of the associations lazily mapped together and intercepts the calls using these proxy implementations. When the requested information goes missing, it is loaded from the database before the control is given to the parent control.

International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-2, May 2015
ISSN: 2395-3470
www.ijseas.com

3.2 AJAX PAGINATION

AJAX (Asynchronous JavaScript and XML) is a mechanism to update the data on a page without the need for it to reload the complete page. This technique we found useful particularly in the search application because this enables the application to fetch limited data at a time. For example, the search that the traditional application made on a database consisting of 10000 values, had about 900 results. Of those, only 10 results would appear on the first page. And using ajax based pagination, the subsequent results will be called in a much faster way, then loading all the 900 results together. An example of AJAX pagination is shown in Figure 4.

Figure 4: AJAX PAGINATION

3.3 NOSQL

NoSQL provides a mechanism for representing data in a form different than that of a traditional relational database. It helps in simplicity and scaling of huge amounts of data. That is precisely why we preferred NoSQL as a technology to enhance the search based application. They are often used in big data and real-time applications. Hibernate OGM (Object/Grid Mapper) provides a persistent support for the NoSQL.

3.4 EHCACHE

EHCACHE can also be coupled with hibernate to automatically cache common queries in memory to provide substantially lower latency in the searching process. EHCACHE is provides a second level cache for hibernate. After adding the required configuration files, EHCACHE can be easily included in the hibernate application. An example of the snippet to add EHCACHE using annotations is:

@Cache (usage=CacheConcurrencyStrategy.READ_ONLY, region="department")

4.0 RESULTS

We compared the performances of both these search implementations on a Core i7 machine with a 1.7 GHz Quad core processor, and simulated the clustering on an amazon EC2 platform using Hadoop. We searched 100 words, phrases and sentences from the database. And averaged the total time in showing the results on the web application. For testing purposes the application was running on localhost to avoid the network delays. The results clearly depicts that the performance of the Blue i.e. the simple search is very low even for a very small set of records. Whereas the Red line representing the Traditional Hibernate Search implementation provides a significant improvement in its performance. But, the Grey line stands out as the relative time taken by the Enhanced search for a larger dataset is much less than that of the Traditional hibernate search and the simple search.

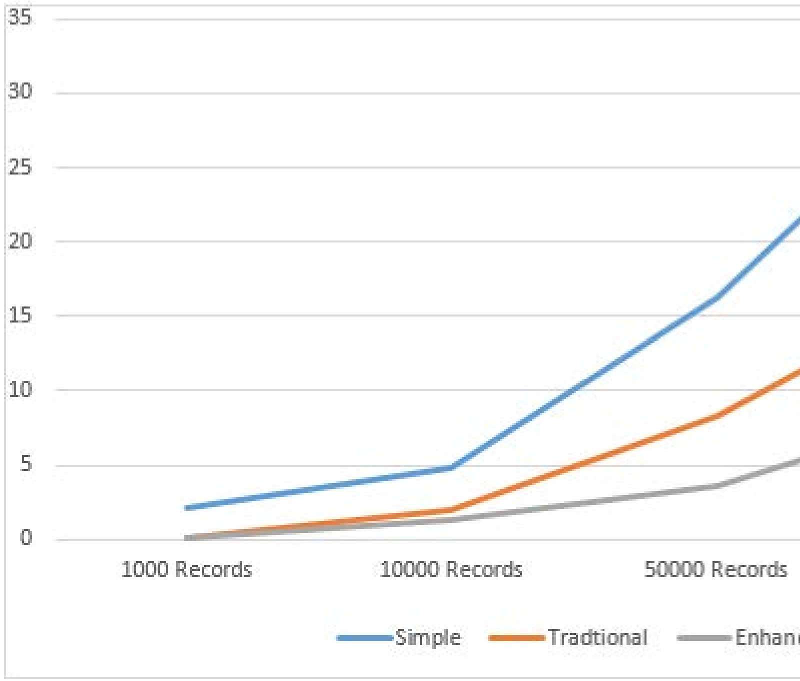


Figure 5: Traditional Vs Enhanced

5.0 CONCLUSION

Thus this paper studies the various aspects of Hibernate its requirement in scaling the needs of current web applications. We can see that the huge amount of data present in the modern databases that needs to be searched upon can be accessed easily by using Hibernate Search. As it provides a developer friendly interface to persist data and to get results much faster than the orthodox. The detailed mechanism involved behind the Hibernate Search is beyond the scope of this paper. And the future work would be to identify the rivals of hibernate and Hibernate Search and to compare the performances of each one of them.

REFERENCES

[1] Vasavi, B. "HIBERNATE TECHNOLOGY FOR AN EFFICIENT BUSINESS APPLICATION EXTENSION." Journal of Global Research in Computer Science 2.6 (2011): 118-125.

[2] en.wikipedia.org/wiki/Ehcache

[3] www.hibernate.org

[4] en.wikipedia.org/wiki/NoSQL

[5] acupof.blogspot.in/2011/02/lucene-and-hibernate-search-small.html