International Journal of Electronics, Electrical and Computational System
IJEECS
ISSN 2348-117X
Volume 6, Issue 8
August 2017

Academic Science

An Application for Monitoring Solr

Yamin Alam
Gauhati University Institute of Science and Technology, Guwahati Assam, India
Nabamita Deb
Gauhati University Institute of Science and Technology, Guwahati Assam, India

ABSTRACT

Apache Solr is an open source enterprise search platform built upon a Java library called Lucene. It runs in a Java servlet container such as Tomcat serving as a standalone full-text search server. Because of the active community of Solr, the contributions it made, and the solid foundations with Lucene at its core, Apache's Solr is considered to be a great software package. With some dazzling features Solr also provides distributed search along with index replication, controlling the search and navigation features of huge numbers of the world's biggest websites and thus it is high in demand. Keeping in mind Solr's popularity and the importance of data, an application for monitoring Solr which accumulates some performance metrics so as to ensure that it is available and see if it is working properly or not is the need of ours. This paper involves monitoring of the Solr Server by fetching some performance data using the JMX APIs which are later stored in a database. Later the stored data is then fetched and are shown at the front end in a real-time graph making the monitoring process easy as the application developed will be well equipped with necessary parameters for monitoring the statistics of the server.

Keywords: Solr, Lucene, JMX API, Indexing.

1. INTRODUCTION

The popularity of the Apache Solr Server compelled us to pick it as a base for our paper. It is an open source search platform built upon a Java library called Lucene. Solr is a popular search platform for Web locales because it can index and search multiple sites and return recommendations and suggestions for related content based on the search queries order. Solr can be used to index and search reports as well as email attachments, which makes it popular for enterprise search. It is designed basically for scalability and fault resistance.

Solr works with HTTP and XML and provides APIs for JSON, Python, and Ruby. Some of the important features of Solr include indexing in real time, full-text search, automated index replication, hit highlighting, faceted search, dynamic clustering, automated failover and recovery, extensible through plugins, integration with database, Not only SQL (NoSQL) features, handling of rich documents(parsing and ordering Word, PDF etc) and so forth [1].

As it is the second-most popular enterprise search engine [2], it is important to monitor certain aspects of Solr, so problems can be resolved before they become show stoppers. So developing a simple open source application which can monitor Solr to ensure that it is available and gather performance data for tuning will be useful for all the administrators who are using it. Here in this paper, for monitoring purpose of Solr in the application layer, some performance metrics are gathered which includes Index Size, CPU Process Time, Cache Hit Ratio, CPU Load Percentage, Number of Documents and Heap Size.

The coding is done in such a way that the JMX APIs fetch the statistics from the Solr Server and are later stored in a database along with a timestamp. From that database the values are to be fetched and displayed

at the front end at the next stage. So that for a particular date and time one can easily view the statistics how Solr was responding at that particular point of time and thus monitoring it. This will ease the monitoring purpose as the application developed will be well equipped with necessary parameters for monitoring the statistics of the server. So that when the server or the system fails, one can easily check the statistics of an earlier point of time till the day when server fails and thus producing an efficient program for the corresponding system.

This paper can thus help the researchers as well as the developers who are willing to do research on monitoring the Solr server and make a better application in future. It can also serve as an important material for understanding the behavior of the Solr server i.e. how the Index Size varies with the other parameters of the server, what is the CPU Process Time at the time of failure(if the server fails), how the heap usage is related to the CPU load percentage etc.

2. BACKGROUND STUDY

2.1 AN INTRODUCTION TO LUCENE AND SOLR: Solr is an open source search platform built on top of Lucene. Certainly **Solr = Lucene + Added features**. It runs in a Java servlet container such as Tomcat. Solr was created by Yonik Seeley at CNET and contributed to Apache by CNET. Solr is a complete application and Lucene is the heart of Solr, being in its core. Both Lucene and Solr are search technologies available for free as open source from the Apache Software Foundation. Lucene is the underlying search library, and Solr is a platform built on top of Lucene that makes it easy to build Lucene-based applications. Both Lucene and Solr are full-featured and are scalable, have excellent performance and almost having relevant ranking. These technologies are used today by thousands of organizations and power search applications [1] [3]. As Solr is built on Lucene so it follows the same layout. As parts of the search service value add over Lucene, Solr provides JMX MBeans for obtaining the health status.

2.2 THE JAVA MANAGEMENT EXTENSIONS (JMX) API: JMX is a standard API that supplies tools for managing and monitoring of resources such as applications, devices, service oriented systems and networks and the JVM. Those resources are represented by objects called MBeans (Managed Bean). The classes in the API can be loaded dynamically and instantiated. The applications for managing and monitoring can be designed and developed using the JDK [4] [5]. The JMX being a trademark of the Oracle Corporation has been adopted early on by the J2EE community, and is a part of J2SE since version 5.0.

Some of the uses of the JMX technology include changing the configuration of the application, gathering the statistics about the behavior of the application and making them available, alerting about state changes and erroneous conditions [5] [6]. The JMX can be implemented by using the packages **javax.management and sub-packages** (the public JMX API packages) or **com.sun.jmx and sub-packages** (the private Sun implementation packages) in which the APIs can be called only by the JDK software library classes.

JMX Architecture: JMX uses a 3-level architecture which includes the Probe level, the Agent level and the Remote Management level:

1. The Test level or the Instrumentation level or the Probe level contains the tests called MBeans which instruments the resources.
2. The MBeanServer or the Agent level is the heart of JMX which acts as an intermediary between the MBean and the applications.

3. In the Remote Management level the remote applications get access to the MBeanServer through adaptors and connectors [7].

Managed Beans (MBeans): An MBean is a type of JavaBean, created with dependency injection used in the JMX technology. The MBean represents a resource running in the JVM, such as an application or a Java EE technical service such as transactional monitor, JDBC driver etc. The MBeans can be used for collecting statistics on attributes like performance, resources usage, notifying or alerting events like state changes or errors (push) and for getting and setting application configurations or properties (push/pull) [7].Java EE 6 provides a bean class which is nothing but a managed bean that is implemented by a Java class [7].

2.3 INDEXING DATA: If the Solr server doesn't contain any data, one can modify a Solr index by POSTing commands to Solr to add (or update) documents, delete documents, and commit pending adds and deletes. These commands can be in a variety of formats [3] [4].

The **exampledocs** directory contains sample files showing of the types of commands Solr accepts, as well as a java utility for posting them from the command line (a **post.sh** shell script is also available, but for this , one have to use the cross-platform Java client. Run **java -jar post.jar -h** to see its various options) [3] [4].

Open a new terminal window, enter the exampledocs directory, and run "**java -jar post.jar**" on some of the XML files in that directory.

user:~/solr/example/exampledocs$ java –jar post.jar solr.xml monitor.xml

SimplePostTool: version 1.4

SimplePostTool: POSTing files to http://localhost:8983/solr/update. .

SimplePostTool: POSTing file solr.xml

SimplePostTool: POSTing file monitor.xml

SimplePostTool: COMMITing Solr index changes. .

This is how indexing is done for two particular documents in Solr.

If indexing is to be done on all of the sample data, the command is given as:

user:~/solr/example/exampledocs$ java –jar post.jar *.xml

3. METHODOLOGY

3.1 SYSTEM ARCHITECTURE: The Solr server is installed and integrated with JMX. After it is configured, coding is done using Java in such a way that whenever the server is alive and the code is executed, information is fetched from the Solr and it is stored in the database using the JDBC driver. Later from that database, the data stored are again fetched and will be displayed in a UI which is developed using JSP, HTML, REST APIs, JSON objects and JavaScript after the successful login by the administrator.

3.2 FUNCTIONAL DETAILS OF DIFFERENT MODULES

Designing proper and efficient SQL statements and storing to a database: MySql is used for interacting with database by passing queries. The result set is stored in the database. The Java Database Connectivity (JDBC) driver is used for connecting to the database which provides methods for querying and updating data in a database.

Back end programming (Server side): Java is used as back end to serve the server side programming.

Front end programming (Client side): JSP, HTML, REST APIs, JSON objects and JavaScript is used for making the Graphical User Interface (GUI). JSON object is also used during this purpose.

Dependencies: log4j, solr-solrj, mysql-connector-java, simplejmx etc are used.

3.3 Data Dictionary

Table 1.Data Dictionary for 'solr_metrics'

Field	Type
id	bigint unsigned
time	Date time
numDocs	bigint unsigned
heapUsage	double unsigned
sizeInMB	double unsigned
cpuLoad	double unsigned
docCacheHitRatio	double unsigned
processCpuTime	double unsigned NULL=Yes

4. IMPLEMENTATION

Step 1: At first the JMX has to be configured in the Solr server. The JMX configuration is provided in the solrconfig.xml [3] [4].

Step 2: A new MBean server has to be created and configured after the configuration of JMX by using the command **<jmxserviceUrl="service:jmx:rmi:///jndi/rmi://localhost:9999/solrjmx"/>**.

If the JMXConnectorServer can't be started (probably because the service URL is bad), an exception is thrown [3] [4].

Step 3: Start the Jetty application server, the Solr Server on port 8983 by using

java -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=9999 -Dcom.sun.management.jmxremote.ssl=falseDcom.sun.management.jmxremote.authenticate=false -jar start.jar [3] [4].

Step 4: Start jconsole (provided with the Sun JDK in the bin directory) by the following steps [3][4]:

i) Run "ant example" to build the example war file.

ii) Start jconsole (provided with the Sun JDK in the bin directory).

iii) Connect to the "start.jar" shown in the list of local processes.

iv) Switch to the "MBeans" tab. "solr" will be listed there.

Step 5: Creating database for storing the values of Solr metrics.

Step 6: Implementing the JMX APIs and fetching data from the server: The code is written in Java. Suitable APIs for all the specified monitoring metrics of Solr are implemented using Java. The jar file thus created is executed using shell script and are stored in the database with a particular timestamp of each. The JDBC driver is used for the database connectivity purpose to the server. The coding is done in such a way that it will fetch the statistics of the Solr server after every 5 seconds and store it in the

database when it is executed. The Solr server has to be alive during the execution process. Fig. 1 shows the flowchart for fetching the statistics from the server.

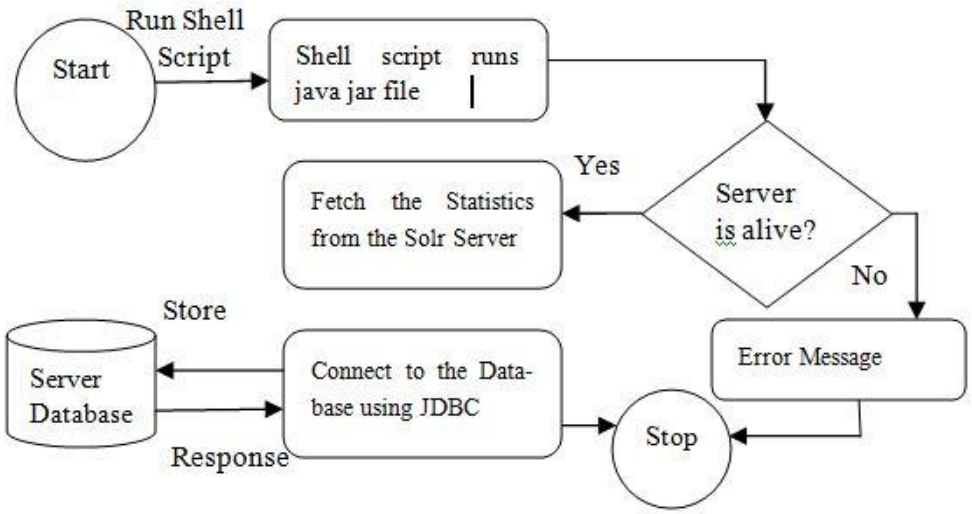

Fig. 1.Flowchart for fetching data from the server

Step 7: Creating the Client Side Graphical User Interface (GUI) for monitoring: The GUI is created using JSP, HTML, JavaScript, REST APIs and JSON objects. The administrator needs to authenticate his/her identity by entering username and password. The password entered is authenticated. If the response is positive it is redirected to the UI page where the administrator needs to give the date and time for which day and hour he wants to monitor the Solr server, else an error message is displayed. If the administrator provides the date and hour as such when the server is not alive, then also an error message is displayed. A checkbox for 'auto refresh' is also provided which will give the current statistics of the Solr after every 5 seconds.

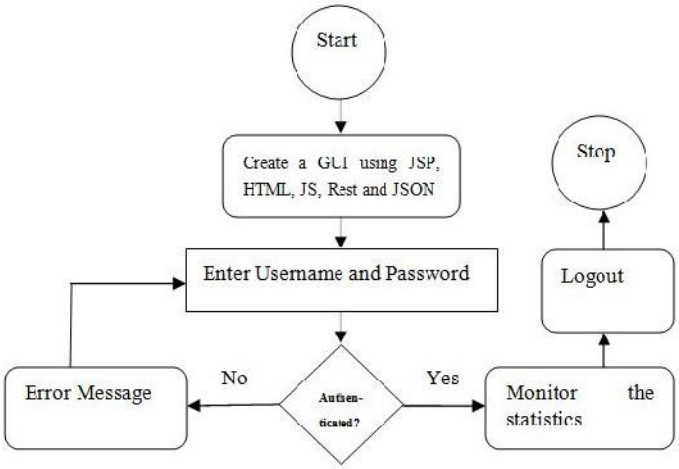

Fig. 2.Flowchart for the GUI

International Journal of Electronics, Electrical and Computational System
IJEECS
ISSN 2348-117X
Volume 6, Issue 8
August 2017

5 RESULTS

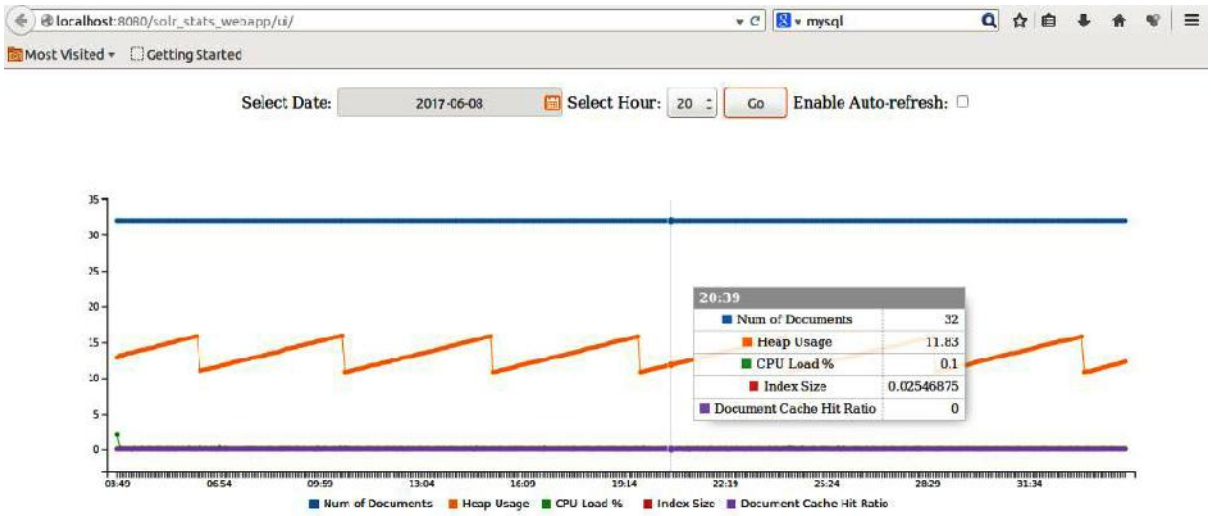

Fig. 3.The client side GUI with varying Heap Usage

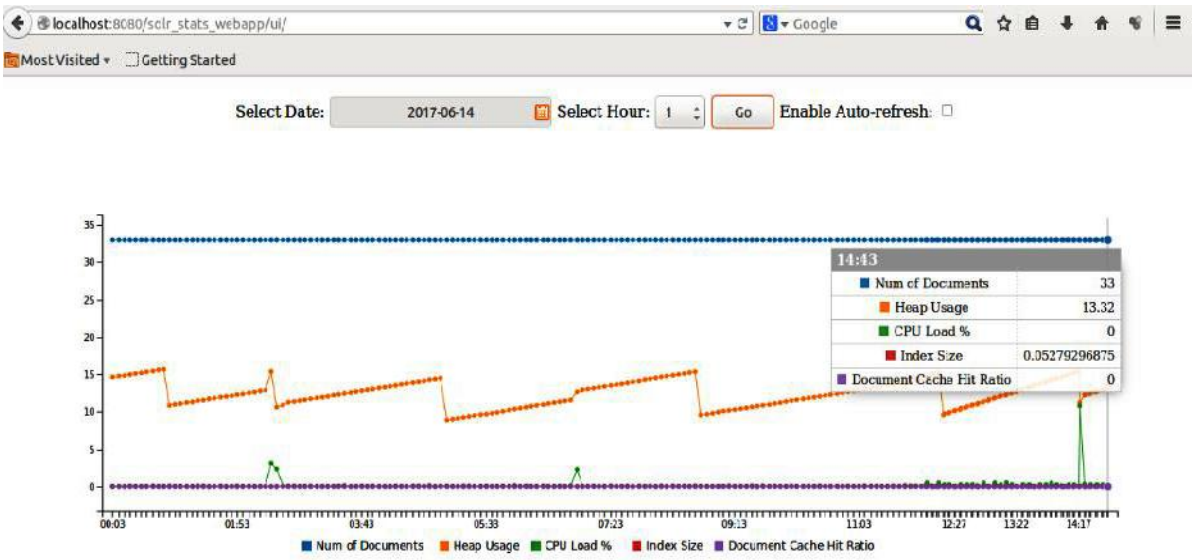

Fig. 4.The client side GUI when more files are added to Solr with proper indexing

The graph will vary accordingly when more files are added in the Solr server's **exampledocs** folder with proper indexing.

6. CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

The goal of developing this monitoring application is to provide the users with the information about the statistics of the Solr server in the application layer which includes the Index Size, Heap Usage, CPU Process Time, Number of documents, CPU Load Percentage and Document Cache Hit Ratio. It can be helpful for the researchers who are doing research on the behavior of the Solr server as in the UI, the graph shows how index size varies with the number of documents, how the heap usage is related to the CPU load percentage and so on.

6.2 FUTURE SCOPE

The statistics of the Solr server can be monitored with the help of the application developed. Though the results obtained from this application are satisfactory, there is a provision for future enhancement of this application which includes:

- The application can be improved by incorporating more security features to it.
- More performance metrics can be added to it according to the needs in order to make it more efficient.
- The GUI can be made more powerful by adding more features.

REFERENCES

[1] Apache Solr Homepage, http://lucene.apache.org/solr/, last accessed 2017/01/27.
[2] DB-Engines Ranking - popularity ranking of search engines, https://db-engines.com/en/ranking/search%20engine, last accessed 30 January 2017/01/30.
[3] Rafal Kuć: Apache Solr 3.1 Cookbook. Packt Publishing, Birmingham (2011).
[4] Apache Solr, Apache Solr Reference Guide Covering Apache Solr 4.6. Licensed to the Apache Software Foundation (ASF) (2013).
[5] JDK Java Management Extensions (JMX), https://docs.oracle.com/javase/8/docs/technotes/guides/jmx/, last accessed 2017/02/12.
[6] Lesson: Overview of the JMX Technology, https://docs.oracle.com/javase/tutorial/jmx/overview/, last accessed 2017/02/12.
[7] Java Management Extensions, http://en.wikipedia.org/wiki/Java_Management_Extensions, last accessed 2017/02/14.