Scaling Apache Solr

Chapter No.1
"Understanding Apache Solr"
In this package, you will find:

The author’s biography

A preview chapter from the book, Chapter no.1 "Understanding Apache Solr"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Hrishikesh Vijay Karambelkar is an enterprise architect with a blend of technical and entrepreneurial experience of more than 13 years. His core expertise involves working with multiple topics that include J2EE, Big Data, Solr, Link Analysis, and Analytics. He enjoys architecting solutions for the next generation of product development for IT organizations. He spends most of his work time now solving challenging problems faced by the software industry.

In the past, he has worked in the domains of graph databases; some of his work has been published in international conferences, such as VLDB, ICDE, and so on. He has recently published a book called Scaling Big Data with Hadoop and Solr, Packt Publishing. He enjoys spending his leisure time traveling, trekking, and photographing birds in the dense forests of India. He can be reached at http://hrishikesh.karambelkar.co.in/.

I am thankful to Packt Publishing for providing me with the opportunity to share my experiences on this topic. I would also like to thank the reviewers of this book, especially Nick Veenhof from Acquira for shaping it up; the Packt Publishing team, Kartik Vedam and Neil Alexander, for their support during bad times; and my dear wife Dhanashree, who stood by me all the time.

For More Information:
Scaling Apache Solr

With the growth of information assets in enterprises, the need to build a rich, scalable search application has becomes critical. Today, Apache Solr is one of the most widely adapted, scalable, feature-rich, and high-performance open source search application servers.

Scaling Apache Solr is intended to enable its users to transform Apache Solr from a basic search server, to an enterprise-ready, high performance, and scalable search application. This book is a comprehensive reference guide that starts with the basic concepts of Solr; it takes users through the journey of how Solr can be used in enterprise deployments, and it finally dives into building a highly efficient, scalable enterprise search application using various techniques and numerous practical chapters.

What This Book Covers

Chapter 1, Understanding Apache Solr, introduces readers to Apache Solr as a search server. It discusses problems addressed by Apache Solr, its architecture, features, and, finally, covers different use cases for Solr.

Chapter 2, Getting Started with Apache Solr, focuses on setting up the first Apache Solr instance. It provides a detailed step-by-step guide for installing and configuring Apache Solr. It also covers connecting to Solr through SolrJ libraries.

Chapter 3, Analyzing Data with Apache Solr, covers some of the common enterprise search problems of bringing information from disparate sources and different information flow patterns to Apache Solr with minimal losses. It also covers advanced topics for Apache Solr such as deduplication, and searching through images.

Chapter 4, Designing Enterprise Search, introduces its readers to the various aspects of designing a search for enterprises. It covers topics pertaining to different data processing patterns of enterprises, integration patterns, and a case study of designing a knowledge repository for the software IT industry.

Chapter 5, Integrating Apache Solr, focuses on the integration aspects of Apache Solr with different applications that are commonly used in any enterprise. It also covers different ways in which Apache Solr can be introduced by the enterprise to its users.

Chapter 6, Distributed Search Using Apache Solr, starts with the need for distributed search for an enterprise, and it provides a deep dive to building distributed search using Apache SolrCloud. Finally, the chapter covers common problems and a case study of distributed search using Apache Solr for the software industry.

For More Information:

Chapter 7, Scaling Solr through Sharding, Fault Tolerance, and Integration, discusses various aspects of enabling enterprise Solr search server to scaling by means of sharding and search result clustering. It covers integration aspects of Solr with other tools such as MongoDB, Carrot2, and Storm to achieve the scalable search solution for an enterprise.

Chapter 8, Scaling Solr through High Performance, provides insights of how Apache Solr can be transformed into a high-performance search engine for an enterprise. It starts with monitoring for performance, and how the Solr application server can be tuned for performing better with maximum utilization of the available sources.

Chapter 9, Solr and Cloud Computing, focuses on enabling Solr on cloud-based infrastructure for scalability. It covers different Solr strategies for cloud, their applications, and deep dive into using Solr in different types of cloud environment.

Chapter 10, Scaling Solr Capabilities with Big Data, covers aspects of working with a very high volume of data (Big Data), and how Solr can be used to work with Big Data. It discusses how Apache Hadoop and its ecosystem can be used to build and run efficient Big Data enterprise search solutions.

Appendix, Sample Configuration for Apache Solr, provides a reference configuration for different Solr configuration files with detailed explanations.

For More Information:
Understanding Apache Solr

The world of information technology revolves around transforming data into information that we can understand. This data is generated every now and then, from various sources, in various forms. To analyze such data, engineers must observe data characteristics, such as the velocity with which the data is generated, volume of data, veracity of data, and data variety. These four dimensions are widely used to recognize whether the data falls into the category of Big Data. In an enterprise, the data may come from its operations network which would involve plant assets, or it may even come from an employee who is updating his information on the employee portal. The sources for such data can be unlimited, and so is the format. To address the need for storage and retrieval of data of a non-relational form, mechanisms such as NOSQL (Not only SQL) are widely used, and they are gaining popularity.

The mechanism of NOSQL does not provide any standardized way of accessing the data unlike SQL in the case of relational databases. This is because of the unstructured data form that exists within NOSQL storage. Some NOSQL implementations provide SQL-like querying, whereas some provide key-value based storage and data access. It does not really address the problem of data retrieval. Apache Solr uses the key-value-based storage as a means to search through text in a more scalable and efficient way. Apache Solr enables enterprise applications to store and access this data in an effective and efficient manner.

In this chapter, we will be trying to understand Apache Solr and we will go through the following topics:

- Challenges in enterprise search
- Understanding Apache Solr
- Features of Apache Solr
- Apache Solr architecture
- Apache Solr case studies

For More Information:  
Understanding Apache Solr

Challenges in enterprise search

The presence of a good enterprise search solution in any organization is an important aspect of information availability. Absence of such a mechanism can possibly result in poor decision making, duplicated efforts, and lost productivity due to the inability to find the right information at any time. Any search engine typically comprises the following components:

1. Crawlers or data collectors focus mainly on gathering the information on which a search should run.
2. Once the data is collected, it needs to be parsed and indexed. So parsing and indexing is another important component of any enterprise search.
3. The search component is responsible for runtime search on a user-chosen dataset.
4. Additionally, many search engine vendors provide a plethora of components around search engines, such as administration and monitoring, log management, and customizations.

Today public web search engines have become mature. More than 90 percent of online activities begin with search engines (http://searchengineland.com/top-internet-activities-search-email-once-again-88964) and more than 100 billion global searches are being made every month (http://searchenginewatch.com/article/2051895/Global-Search-Market-Tops-Over-100-Billion-Searches-a-Month). While the focus of web-based search is more on finding out content on the Web, enterprise searches focus on helping employees find out the relevant information stored in their corporate network in any form. Corporate information lacks useful metadata that an enterprise search can use to relate, unlike web searches, which have access to HTML pages that carry a lot of useful metadata for best results. Overall, building an enterprise search engine becomes a big challenge.

Many enterprise web portals provide searches over their own data; however, they do not really solve the problem of unified data access because most of the enterprise data that is outside the purview of these portals largely remains invisible to these search solutions. This data is mainly part of various sources such as external data sources, other departmental data, individual desktops, secured data, proprietary format data, and media files. Let’s look at the challenges faced in the industry for enterprise search as shown in the following figure:

For More Information:
Let’s go through each challenge in the following list and try to understand what they mean:

- **Diverse repositories**: The repositories for processing the information vary from a simple web server to a complex content management system. The enterprise search engine must be capable of dealing with diverse repositories.

- **Security**: Security in the enterprise has been one of the primary concerns along with fine-grained access control while dealing with enterprise search. Corporates expect data privacy from enterprise search solutions. This means two users running the same search on enterprise search may get two different sets of results based on the document-level access.

- **Variety of information**: The information in any enterprise is diverse and has different dimensions, such as different types (including PDF, doc, proprietary formats, and so on) of document or different locale (such as English, French, and Hindi). An enterprise search would be required to index this information and provide a search on top of it. This is one of the challenging areas of enterprise searches.

For More Information:

• **Scalability**: The information in any enterprise is always growing and enterprise search has to support its growth without impacting its search speed. This means the enterprise search has to be scalable to address the growth of an enterprise.

• **Relevance**: Relevance is all about how closely the search results match the user expectations. Public web searches can identify relevance from various mechanisms such as links across web pages, whereas enterprise search solutions differ completely in the relevance of entities. The relevance in case of enterprise search involves understanding of current business functions and their contributions in the relevance ranking calculations. For example, a research paper publication would carry more prestige in an academic institution search engine than an on-the-job recruitment search engine.

• **Federation**: Any large organization would have a plethora of applications. Some of them carry technical limitations, such as proprietary formats and inability to share the data for indexing. Many times, enterprise applications such as content management systems provide inbuilt search capabilities on their own data. Enterprise search has to consume these services and it should provide a unified search mechanism for all applications in an enterprise. A federated search plays an important role while searching through various resources.

A federated search enables users to run their search queries on various applications simultaneously in a delegated manner. Participating applications in a federated search perform the search operation using their own mechanism. The results are then combined and ranked together and presented as a single search result (unified search solution) to the user.
Let's take a look at fictitious enterprise search implementation for a software product development company called ITWorks Corporation. The following screenshot depicts how a possible search interface would look:

A search should support basic keyword searching, as well as advanced searching across various data sources directly or through a federated search. In this case, the search is crawling through the source code, development documentation, and resources capabilities, all at once. Given such diverse content, a search should provide a unified browsing experience where the result shows up together, hiding the underlying sources. To enable rich browsing, it may provide refinements based on certain facets as shown in the screenshot. It may provide some interesting features such as sorting, spell checking, pagination, and search result highlighting. These features enhance the user experience while searching for information.

For More Information:
Apache Solr – an overview

The need to resolve the problems of enterprise search has triggered interest in many IT companies to come up with an enterprise search solution. This includes companies such as Oracle, Microsoft, and Google who sell their solutions to customers. Doug Cutting created the open source information retrieval library called Apache Lucene during 1997. It became part of sourceforge.net (one of the sites hosting open source projects and their development). Lucene was capable of providing powerful full text search, and indexing capabilities on Java. Later in 2001, the Lucene project became a member of the Apache software foundation. The open source community contributed significantly to the development of Apache Lucene, which has led to exponential growth until this point in time. Apache Lucene is widely used in many organizations for information retrieval and search.

Since Apache Solr uses Apache Lucene for indexing and searching, Solr and Lucene index are the same. That means Apache Solr can access indexes generated using Lucene; although, we may just need to modify the Solr schema file to accommodate all the attributes of the Lucene index. Additionally, if Apache Solr is using a different Lucene library, we need to change <luceneMatchVersion> in solrconfig.xml. This is particularly useful when the client would like to upgrade his custom Lucene search engine to Solr without losing data.

Features of Apache Solr

Apache Solr comes with a rich set of features that can be utilized by enterprises to make the search experience unique and effective. Let's take an overview of some of these key features. We will understand how they can be configured in the next chapter at deeper level.

Solr for end users

A search is effective when the searched information can be seen in different dimensions. For example, if a visitor is interested in buying a camera and he visits online shopping websites and searches for his model. When a user query is executed on the search, a search would rank and return a huge number of results. It would be nice, if he can filter out the results based on the resolution of the camera, or the make of the camera. These are the dimensions that help the user improve querying. Apache Solr offers a unique user experience that enables users to retrieve information faster.

For More Information:
Powerful full text search

Apache Solr provides a powerful full text search capability. Besides normal search, Solr users can run a search for specific fields, for example, `error_id:severe`. Apache Solr supports wildcards in the queries. A search pattern consisting only of one or more asterisks will match all terms of the field in which it is used, for example, `book_title:*`. A question mark can be used where there might be variations for a single character. For example, a search for `?ar` will match with `car`, `bar`, `jar` and a search for `c?t` will match with `cat`, `cot`, `cut`. Overall, Apache supports the following power expressions to enable the user to find information in all possible ways as follows:

- Wildcards
- Phrase queries
- Regular expressions
- Conditional login (and, or, not)
- Range queries (date/integer)

Search through rich information

Apache Solr search can generate indexes out of different file types including many rich documents such as HTML, Word, Excel, Presentations, PDF, RTF, E-mail, ePub formats, the `.zip` files, and many more. It achieves this by integrating different packages such as Lucene, and Apache Tika. These documents when uploaded to Apache Solr get parsed and an index is generated by Solr for search. Additionally, Solr can be extended to work with specific formats by creating customer handlers/adapters for the same. This feature enables Apache Solr to work best for enterprises dealing with different types of data.

Results ranking, pagination, and sorting

When searching for information, Apache Solr returns results page-by-page starting with top K results. Each result row carries a certain score, and the results are sorted based on the score. The result ranking in Solr can be customized as per the application's requirements. This allows the user's flexibility to search more specifically for relevant content. The size of the page can be configured in Apache Solr configuration. Using pagination, Solr can compute and return the results faster than otherwise. Sorting is a feature that enables Solr users to sort the results on certain terms or attributes, for example, a user might consider sorting of results based on increasing price order on an online shopping portal search.

For More Information:
Facets for better browsing experience

Apache Solr facets do not only help users to refine their results using various attributes, but they allow better browsing experience along with the search interface. Apache Solr provides schema-driven, context-specific facets that help users discover more information quickly. Solr facets can be created based on the attributes of the schema that is designed before setting up the instance. Although Apache Solr works on a schema defined for the user, it allows them to have flexibility in the schema by means of dynamic fields, enabling users to work with content of a dynamic nature.

Based on the schema attributes, Apache Solr generates facet information at the time of indexing instead of doing it on the stored values. That means, if we introduce new attributes in the schema after indexing of our information, Solr will not be able to identify them. This may be solved by re-indexing the information again.

Each of these facet elements contain the filter value, which carries a count of results that match among the searched results. For the newly introduced schema attributes, users need to recreate the indexes that are created before. There are different types of facets supported by Solr. The following screenshot depicts the different types of facets that are discussed:

The facets allow you to get aggregated view on your text data. These aggregations can be based on different compositions such as count (number of appearances), time based, and so on. The following table describes the facets and their description supported by Apache Solr:

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field-Value</td>
<td>You can have your schema fields as facet components here. It shows the count of top fields. For example, if a document has tags, a field-value facet on the tag Solr field will show the top N tags, which are found in the matched result as shown in the image.</td>
</tr>
</tbody>
</table>

For More Information:
Facet Description

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Range faceting is mostly used on date/numeric fields, and it supports range queries. You can specify start and end dates, gap in the range and so on. There is a facet called date facet for managing dates, but it has been deprecated since Solr 3.2, and now the date is being handled in range faceting itself. For example, if indexed, a Solr document has a creation date and time; a range facet will provide filtering based on the time range.</td>
</tr>
<tr>
<td>Pivot</td>
<td>A pivot gives Solr users the ability to perform simple math on the data. With this facet, they can summarize results, and then get them sorted, take an average, and so on. This gives you hierarchical results (also sometimes called hierarchical faceting).</td>
</tr>
<tr>
<td>Multi-select</td>
<td>Using this facet, the results can be refined with multiple selects on attribute values. These facets can be used by the users to apply multiple criteria on the search results.</td>
</tr>
</tbody>
</table>

**Advanced search capabilities**

Apache Solr provides various advanced search capabilities. Solr comes with a more like this feature, which lets you find documents similar to one or more seed documents. The similarity is calculated from one or more fields of your choice. When the user selects similar results, it will take the current search result and try to find a similar result in the complete index of Solr.

When the user passes a query, the search results can show the snippet among the searched keywords highlighted. **Highlighting** in Solr allows fragments of documents that match the user's query to be included with the query response. Highlighting takes place only for the fields that are searched by the user. Solr provides a collection of highlighting utilities, which allow a great deal of control over the field's fragments, the size of fragments, and how they are formatted.

When a search query is passed to Solr, it is matched with the number of results. The order in which the results are displayed on the UI is based on the relevance of each result with the searched keyword(s), by default. **Relevance** is all about proximity of the result set with the searched keyword that is returned by Solr when a query is performed. This proximity can be measured in various ways. The relevance of a response depends upon the context in which the query was performed. A single search application may be used in different contexts by users with different needs and expectations. Apache Solr provides the relevant score calculations based on various factors such as the number of occurrences of searched keyword in the document or the co-ordination factor, which relates to the maximum number of terms matched among the searched keywords. Solr not only gives flexibility to users to choose the scoring, but also allows users to customize the relevant ranking as per the enterprise search expectations.

For More Information:

Apache Solr allows **spell checker** based on the index proximity. There are multiple options available under the label, in one case Solr provides suggestions for the misplaced word when searched, in the other case Solr returns a suggestion to the user with the **Did you mean** prompt. The following screenshot shows an example of how these features would look on Apache Solr’s client side:

Additionally, Apache Solr has a suggest feature that suggests the query terms or phrases based on incomplete user inputs. With the help of suggestions, users can choose from the list of suggestions as they start typing a few characters. These completions come from the Solr index generated at the time of data indexing from the first top-k matches ranked based on relevance, popularity, or the order of alphabets. Consider the following screenshot:

In many enterprises, location-based information along with text data brings value in terms of visual representation. Apache Solr supports geospatial search. A Solr search provides advanced geospatial capabilities in the search by which users can sort the results based on geographical distances (longitude and latitude), or rank the results based on proximity. This capability comes from the Lucene spatial module.

---

For More Information:

Enterprises are not limited to any languages and often contain a landscape of non-English applications used daily by the employees. Sometimes, the documentation has local languages. In such cases, an enterprise search is required to have the capability to work on various languages instead of limiting itself on one. Apache Solr has built-in language detection and provides language specific text analysis solutions for many languages. Many times, the implementers need to customize the Solr instance to work for us as per their requirements for multi-lingual support.

Administration
Like any other enterprise search operations, Apache Solr facilitates system administrators with various capabilities. This section discusses different features supported at the administration level for Apache Solr.

Apache Solr has built-in administration user interface for administrators and Solr developers. Apache Solr has evolved its administration screen. Version 4.6 contains many advanced features. The administration screen in Solr looks like the following screenshot:

The Admin UI provides a dashboard that provides information about the instance and the system. The logging section provides Apache logging service (log4j) outputs and various log levels such as warning, severe, and error. The core admin UI details out management information about different cores. The thread dump screen shows all threads with CPU time and thread time. The administrators can also see stack trace for threads.

[17]

For More Information:
A collection represents complete logical index, whereas a Solr core represents an index with a Solr instance that includes configuration and runtime. Typically, the configuration of Solr core is kept in the `/conf` directory. Once the user selects the core, they get access to various core-specific functions such as current configuration view, test UI for testing various handlers of Solr, and schema browser. Consider the following features:

- **JMX monitoring**: The Java Management Extension (JMX) technology provides the tools for managing and monitoring of web-based, distributed system. Since Version 3.1, Apache Solr can expose the statistics of runtime activities as dynamic Managed Beans (MBeans). The beans can be viewed in any JMX client (for example, JConsole). With every release, MBeans gets added, and administrators can see the collective list of these MBeans using administration interface. (Typically, it can be seen by accessing: `http://localhost:8983/solr/admin/mbeans/`).

- **Near real time search**: Unlike Google's lazy index update, based on crawler's chance of visiting certain pages, the enterprise search at times requires fast index updates based on the changes. It means, the user wants to search the near real time databases. Apache Solr supports soft commit.

Whenever users upload documents to the Solr server, they must run a commit operation, to ensure that the uploaded documents are stored in the Solr repository. A soft commit is a Solr 4.0 introduced feature that allows users to commit fast, by passing costly commit procedures and making the data available for near real-time search.

With soft commit, the information is available immediately for searching; however, it requires normal commit to ensure the document is available on a persistent store. Solr administrators can also enable autosoft commit through Apache Solr configuration.

- **Flexible query parsing**: In Apache Solr, query parsers play an important role for parsing the query and allowing the search to apply the outcome on the indexes to identify whether the search keywords match. A parser may enable Solr users to add search keywords customizations such as support for regular expressions or enabling users with complex querying through the search interface. Apache Solr, by default, supports several query parsers, offering the enterprise architects to bring in flexibility in controlling how the queries are getting parsed. We are going to understand them in detail in the upcoming chapters.

For More Information:
• **Caching**: Apache Solr is capable of searching on large datasets. When such searches are performed, the cost of time and performance become important factors for the scalability. Apache Solr does caching at various levels to ensure that the users get optimal performance out of the running instance. The caching can be performed at filter level (mainly used for filtering), field values (mainly used in facets), query results (top-k results are cached in certain order), and document level cache. Each cache implementation follows a different caching strategy, such as least recently used or least frequently used. Administrators can choose one of the available cache mechanisms for their search application.

• **Integration**: Typically, enterprise search user interfaces appear as a part of the end user applications, as they only occupy limited screens. The open source Apache Solr community provides client libraries for integrating Solr with various technologies in the client-server model. Solr supports integration through different languages such as Ruby, PHP, .NET, Java, Scala, Perl, and JavaScript. Besides programming languages, Solr also integrates with applications, such as Drupal, WordPress, and Alfresco CMS.

### Apache Solr architecture

In the previous section, we have gone through various key features supported by Apache Solr. In this section, we will look at the architecture of Apache Solr. Apache Solr is a J2EE-based application that internally uses Apache Lucene libraries to generate the indexes as well as to provide a user friendly search. Let's look at the Solr architecture diagram as follows:

![Apache Solr Architecture Diagram](https://example.com/solr_architecture.png)

For More Information:

The Apache Solr instance can run as a single core or multicore; it is a client-server model. In case of a multicore, however, the search access pattern can differ. We are going to look into this in the next chapter. Earlier, Apache Solr had a single core, which in turn, limited the consumers to run Solr on one application through a single schema and configuration file. Later, the support for creating multiple cores was added. With this support, one can run one Solr instance for multiple schemas and configurations with unified administrations. For high availability and scalability requirements, Apache Solr can run in a distributed mode. We are going to look at it in Chapter 6, Distributed Search Using Apache Solr. There are four logical layers in which the overall Solr functionality can be divided. The storage layer is responsible for management of indexes and configuration metadata. The container is the J2EE container on which the instance will run, and Solr engine is the application package that runs on top of the container, and, finally, the interaction talks about how clients/browser can interact with Apache Solr server. Let's look at each of the components in detail in the upcoming sections.

**Storage**

The storage of Apache Solr is mainly used for storing metadata and the actual index information. It is typically a file store, locally configured in the configuration of Apache Solr. The default Solr installation package comes with a Jetty servlet and HTTP server, the respective configuration can be found in the $solr.home/conf folder of Solr installation. An index contains a sequence of documents. Additionally, external storage devices can be configured in Apache Solr, such as databases or Big Data storage systems. The following are the components:

- A document is a collection of fields
- A field is a named sequence of terms
- A term is a string

The same string in two different fields is considered a different term. The index stores statistics about terms in order to make term-based search more efficient. Lucene's index falls into the family of indexes known as an inverted index. This is because it can list, for a term, the documents that contain it. Apache Solr (underlying Lucene) indexing is a specially designed data structure, stored in the filesystem as a set of index files. The index is designed with a specific format in such a way to maximize query performance.
Inverted index is an index data structure for storing mapping from data to actual words and numbers to its location on the storage disk. The following are the strings:
Str[1] = "This is a game of team"
Str[2] = "I do not like a game of cricket"
Str[3] = "People play games everyday"

We have the following inverted file index:
This {1}
Game {1, 2, 3}
Of {1, 2}

**Solr application**
There are two major functions that Solr supports—indexing and searching. Initially, the data is uploaded to Apache Solr through various means; there are handlers to handle data within specific category (XML, CSV, PDF, database, and so on). Once the data is uploaded, it goes through a stage of cleanup called update processor chain. In this chain, initially, the de-duplication phase can be used to remove duplicates in the data to avoid them from appearing in the index unnecessarily. Each update handler can have its own update processor chain that can do document-level operations prior to indexing, or even redirect indexing to a different server or create multiple documents (or zero) from a single one. The data is then transformed depending upon the type.

Apache Solr can run in a master-slave mode. Index replicator is responsible for distributing indexes across multiple slaves. The master server maintains index update and the slaves are responsible for talking with the master to get them replicated for high availability. Apache Lucene core gets packages as library with the Apache Solr application. It provides core functionality for Solr such as index, query processing, searching data, ranking matched results, and returning them back.

Apache Lucene comes with a variety of query implementations. Query parser is responsible for parsing the queries passed by the end search as a search string. Lucene provides TermQuery, BooleanQuery, PhraseQuery, PrefixQuery, RangeQuery, MultiTermQuery, FilteredQuery, SpanQuery, and so on as query implementations.
Index searcher is a basic component of Solr searched with a default base searcher class. This class is responsible for returning ordered match results of searched keyword ranked as per the computed score. The index reader provides access to indexes stored in the filesystem. It can be used to search for an index. Similar to the index searcher, an index writer allows you to create and maintain indexes in Apache Lucene.

The analyzer is responsible for examining the fields and generating tokens. Tokenizer breaks field data into lexical units or tokens. The filter examines the field of tokens from the tokenizer and either it keeps them and transforms them, or discards them and creates new ones. Tokenizers and filters together form a chain or pipeline of analyzers. There can only be one tokenizer per analyzer. The output of one chain is fed to another. Analyzing the process is used for indexing as well as querying by Solr. They play an important role in speeding up the query as well as index time; they also reduce the amount of data that gets generated out of these operations. You can define your own customer analyzers depending upon your use case. In addition to the analyzer, Apache Solr allows administrators to make the search experience more effective by means of taking out common words such as is, and, and are through the stopwords feature. Solr supports synonyms, thereby not limiting search to purely text match. Through the processing of stemming, all words such as played, playing, play can be transformed into the base form. We are going to look at these features in the coming chapters and the appendix. Similar to stemming, the user can search for multiterms of a single word as well (for example, play, played, playing). When a user fires a search query on Solr, it actually gets passed on to a request handler. By default, Apache Solr provides DisMaxRequestHandler. You can visit http://wiki.apache.org/solr/DisMaxRequestHandler to find more details about this handler. Based on the request, the request handler calls the query parser. You can see an example of the filter in the following figure:

---

For More Information:
The query parser is responsible for parsing the queries, and converting it to Lucene query objects. There are different types of parsers available (Lucene, DisMax, eDisMax, and so on). Each parser offers different functionalities and it can be used based on the requirements. Once a query is parsed, it hands it over to the index searcher. The job of the index reader is to run the queries on the index store and gather the results to the response writer.

The response writer is responsible for responding back to the client; it formats the query response based on the search outcomes from the Lucene engine. The following figure displays the complete process flow when a search is fired from the client:

Apache Solr ships with an example schema that runs using Apache velocity. Apache velocity is a fast open source templates engine, which quickly generates HTML-based frontend. Users can customize these templates as per their requirements, although it is not used for production in many cases.

Index handlers are a type of update handler, handling the task of add, update, and delete function on documents for indexing. Apache Solr supports updates through the index handler through JSON, XML, and text format.

**Data Import Handler (DIH)** provides a mechanism for integrating different data sources with Apache Solr for indexing. The data sources could be relational databases or web-based sources (for example, RSS, ATOM feeds, and e-mails).

---

For More Information:  
Although DIH is a part of Solr development, the default installation does not include it in the Solr application; they need to be included in the application explicitly.

Apache Tika, a project in itself extends capabilities of Apache Solr to run on top of different types of files. When a document is assigned to Tika, it automatically determines the type of file, that is, Word, Excel, PDF and extracts the content. Tika also extracts document metadata such as author, title, and creation date, which if provided in schema, go as text field in Apache Solr. This can later be used as facets for the search interface.

**Integration**

Apache Solr, although a web-based application, can be integrated with different technologies. So, if a company has Drupal-based e-commerce sites, they can integrate the Apache Solr application and provide its rich-faceted search to the user. It can also support advanced searches using the range search.

**Client APIs and SolrJ client**

The Apache Solr client provides different ways of talking with Apache Solr web application. This enables Solr to easily get integrated with any application. Using client APIs, consumers can run a search, and perform different operations on indexes. The *Solr Java (SolrJ)* client is an interface of Apache Solr with Java. The SolrJ client enables any Java application to talk directly with Solr through its extensive library of APIs. Apache SolrJ is a part of the Apache Solr package.

**Other interfaces**

Apache Solr can be integrated with other various technologies using its API library and standards-based interfacing. JavaScript-based clients can straightaway talk with Solr using JSON-based messaging. Similarly, other technologies can simply connect to the Apache Solr running instance through HTTP, and consume its services either through JSON, XML, or text formats. Since Solr can be interacted through standard ways, clients can always build their own pretty user interface to interact with the Solr server.
Chapter 1

Practical use cases for Apache Solr

Publicly there are plenty of public sites who claim the use of Apache Solr as the server. We are listing a few here, along with how Solr is used:

- **Instagram**: Instagram (a Facebook company) is one of the famous sites, and it uses Solr to power its geosearch API
- **WhiteHouse.gov**: The Obama administration's website is inbuilt in Drupal and Solr
- **Netflix**: Solr powers basic movie searching on this extremely busy site
- **Internet archive**: Search this vast repository of music, documents, and video using Solr
- **StubHub.com**: This ticket reseller uses Solr to help visitors search for concerts and sporting events.
- **The Smithsonian Institution**: Search the Smithsonian's collection of over 4 million items

You can find the complete list of Solr usage (although a bit outdated) at http://wiki.apache.org/solr/PublicServers. You may also visit to understand interesting case studies about Contextual Search for Volkswagen and the Automotive Industry. The scope of this study is beyond Apache Solr, and talks about semantic search (RDF-based) to empower your overall enterprise industry.

Let's look at how Apache Solr can be used as an enterprise search in two different industries. We will look at one case study in detail, and we will understand how Solr can play a role in the other case study in brief.

Now that we have understood Apache Solr architecture and the use cases, let's look at how Apache Solr can be used as an enterprise search in two different industries.

**Enterprise search for a job search agency**

In this case, we will go through a case study for the job search agency, and how it can benefit using Apache Solr as an enterprise search platform.

For More Information:  
Problem statement
In many job portal agencies, the enterprise search helps reduce the overall time employees spend in matching the expectations from customers with the resumes. Typically, for each vacancy, customers provide a job description. Many times, job description is a lengthy affair, and given the limited time each employee gets, he has to bridge the gap between these two. A job search agency has to deal with various applications as follows:

- Internal CMS containing past information, resumes of candidates, and so on
- Access to market analysis to align the business with expectation
- Employer vacancies may come through e-mails or online vacancy portal
- Online job agencies are a major source for supplying new resumes
- An external public site of the agency where many applicants upload their resumes

Since a job agency deals with multiple systems due to their interaction patterns, having unified enterprise search on top of these systems is the objective to speed up the overall business.

Approach
Here, we have taken a fictitious job search agency who would like to improve the candidate identification time using enterprise search. Given the system landscape, Apache Solr can play a major role here in helping them speed up the process. The following screenshot depicts interaction between unified enterprise searches powered by Apache Solr with other systems:

For More Information:
The figure demonstrates how enterprise search powered by Apache Solr can interact with different data sources. The job search agency interacts with various internal as well as third-party applications. This serves as input for Apache Solr-based enterprise search operations. It would require Solr to talk with these systems by means of different technology-based interaction patterns such as web services, database access, crawlers, and customized adapters as shown in the right-hand side. Apache Solr provides support for database; for the rest, the agency has to build an event-based or scheduled agent, which can pull information from these sources and feed them in Solr. Many times, this information is raw, and the adapter should be able to extract field information from this data, for example, technology expertise, role, salary, or domain expertise. This can be done through various ways. One way is by applying a simple regular expression-based pattern on each resume, and then extracting the information. Alternatively, one can also let it run through the dictionary of verticals and try matching it. Tag-based mechanism also can be used for tagging resumes directly from information contained in the text.

Based on the requirements, now Apache Solr must provide rich facets for candidate searches as well as job searches, which would have the following facets:

- Technology-based dimension
- Vertical- or domain-based dimension
- Financials for candidates
- Timeline of candidates' resume (upload date)
- Role-based dimension

Additionally, mapping similar words (J2EE—Java Enterprise Edition—Java2 Enterprise Edition) through Solr really helps ease the job of agency's employees for automatically producing the proximity among these words, which have the same meaning through the Apache Solr synonym feature. We are going to look at how it can be done in the upcoming chapters.

**Enterprise search for energy industry**

In this case study, we will learn how enterprise search can be used within the energy industry.
Problem statement

In large cities, the energy distribution network is managed by companies, which are responsible for laying underground cables, and setting up power grids at different places, and transformers. Overall, it's a huge chunk of work that any industry will do for a city. Although there are many bigger problems in this industry where Apache Solr can play a major role, we will try to focus on this specific problem.

Many times, the land charts will show how the assets (for example, pipe, cable, and so on) are placed under the roads and the information about lamps are drawn and kept in a safe. This has been paper-based work for long time, and it's now computerized. The field workers who work in the fields for repairs or maintenance often need access to this information, such as assets, pipe locations, and so on.

The demand for this information is to locate a resource geographically. Additionally, the MIS information is part of the documents lying on CMS, and it's difficult to locate this information and link it with geospatial search. This in turn drives the need for the presence of the enterprise search. Additionally, there is also a requirement for identifying the closest field workers to the problematic area to ensure quick resolution.

Approach

For this problem, we are dealing with information coming in totally different forms. The real challenge is to link this information together, and then apply a search that can provide a unified access to information with rich query. We have the following information:

- **Land Charts**: These are PDFs, paper-based documents, and so on, which are fixed information
- **GIS information**: These are coordinates, which are fixed for assets such as transformers, and cables
- **Field engineers’ information**: This gives the current location and is continuously flowing
- **Problem/Complaints**: This will be continuous, either through some portal, or directly fed through the web interface

The challenges that we might face with this approach include:

- Loading and linking data in various formats
- Identifying assets on map
- Identifying the proximity between field workers and assets
- Providing better browsing experience on all this information
Apache Solr supports geospatial search. It can bring a rich capacity by linking assets. Information with geospatial world creates a confluence to enable the users to access this information. It can bring a rich capacity by linking asset information with the geospatial world and creating a confluence to enable the users to access this information at their finger tips.

However, Solr has its own limitations in terms of geospatial capabilities. For example, it supports only point data (latitude, longitude) directly; all the other data types are supported through JTS.

Java Topology Suite (JTS) is a java-based API toolkit for GIS. JTS provides a foundation for building further spatial applications, such as viewers, spatial query processors, and tools for performing data validation, cleaning, and integration.

For the given problem, GIS and land chart will feed information in the Solr server once. This will include linking all assets with GIS information through the custom adapter. The complaint history as well as the field engineers' data will be continuous, and the old data will be overwritten; this can be a scheduled event or a custom event, based on the new inputs received by the system. To meet the expectations, the following application components will be required (minimum):

- Custom adapter with scheduler/event for field engineers' data and complaint register information providing integration with gateways (for tapping GIS information of field engineers) and portals (for the complaint register)
- Lightweight client to scan the existing system (history, other documentation) and load in Solr
- Client application to provide end user interface for enterprise search with URL integration for maps
- Apache Solr with superset schema definition and configuration with support for spatial data types

For More Information:
The following screenshot provides one of the possible visualizations for this system. This system can be extended to further provide more advanced capabilities such as integration with Optical Character Recognition (OCR) software to search across paper-based information, or even to generate dynamic reports based on filters using Solr. Apache Solr also supports output in XML form, which can be applied with any styling and the same can be used to develop nice reporting systems.

Summary

In this chapter, we have tried to understand problems faced by today's industry with respect to enterprise search. We went through Apache Solr and its features to understand its capabilities, followed by the Apache Solr architecture. At the end of this chapter, we saw a use case about how Apache Solr can be applied in the job search agency as an enterprise search.

In the next chapter, we will install and configure the Solr instance for our usage.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.