Enterprise search with development for network management system

MingXue Wang* Robin Grindrod* Jimmy O'Meara* Mikel Zuzuarregui** Eloy Martinez** Enda Fallon**

Network Management Lab*
Ericsson
Athlone, Ireland

Software Research Institute**
Athlone Institute of Technology
Athlone, Ireland

mingxue.wang;robin.grindrod;jimmy.o.meara@ericsson.com emartinez;martinez@research.ait.ie efallon@ait.ie

***Abstract*—Browsing and searching network information for observation, analysis and troubleshooting is an inherent part of using the features and functions of any Network Management System. Enterprise search has capabilities for handling various data types and sources and big data scalability, and is becoming an emerging technology for such network management functions development. In this paper, we give an overview on work done in our research and prototype team regarding an advanced search project. We provide a brief report on search fundamental knowledge and study of Solr search platform stack. It answers common questions from management and development teams regarding adopting search technology for production development, and gives a Solr reference card for developers. We also introduce two advanced search features, user experience based search recommendation and anomaly detection enhanced search ranking from our research work. Two features are developed to make network searches more efficient as it can help user quickly locate the most valuable search results, but the concept can be adopted for search applications in other domains.**

Keywords—Enterprise search; Solr; Search recommendation; Search ranking; Anomaly detection; Big data analytics; Network management

I. INTRODUCTION

One of the trends in BI (business intelligence) and OLAP (online analytic process) is that people are starting to recognize the value of unstructured information in the business, such as documents, web pages, e-mail messages, and social networking connections and dialogues. It is very hard for database technologies such as SQL, which have generally been used for traditional BI applications, to handle this unstructured data. Adopting enterprise search technologies to give them the ability to handle both structured data from corporate databases and unstructured data sources is becoming a new style of BI application development [1] [2]. This trend also applies to the domain of network management systems. Various sources of unstructured text information, such as alarm problem text, trouble tickets, network log messages, etc. are overlooked in current systems.

Network data comes from various network elements and systems and is stored in different data stores. For example, OSS (Operations support system) command logs are stored in a relational database, user session events are stored in a Hadoop file system, and alarms are stored in an object database. Even for the same type of data store, there may be multiple setups for different types of data. For example, command logs are stored in Sybase RDBMS, but the hardware log is stored in another SQL server RDBMS. Data from very different sources are handled separately in general. Adopting enterprise search to handle data from various data sources gives users a centralized view and correlates network data from different sources. This is important for many cases of network analysis and troubleshooting. For example, a lot of alarms have followed a command issued by a user or high user session impacts happened after a configuration event. This type of analysis is not available from any single data source as the data is stored separately.

As a result, enterprise search is becoming one of the main technology study areas for network management system development in our research team. Since, our work involves developing prototypes and researching new features, the contribution of this paper is twofold:

- Firstly, we give essential knowledge on search technologies and summarized information on different components or frameworks associated with each identified layer of the Solr [3] search platform stack. The enterprise search technology and Solr platform are expected to be exploited in this work. It also could be used as a reference card for quickly finding the tools suitable for solving the problems of building Solr based search applications.
- Secondly, we describe two enhanced search features, resulting from our research into giving a more efficient search experience for network management systems. In the first feature, the system recommends a search query, etc., to users based on search experience learned from other users. In the second feature, the search ranking takes into account the anomaly score of data rather than only relevance or time ranking. Nerveless, our concepts and approaches can be easily adopted to enhance other search application domains.

The rest of the paper is structured as follows, we provide a fundamental understanding of enterprise search for system development in section 2; we identify and describe different software layers of the Solr search platform stack in section 3; two enhanced search features developed for network

management systems will be described in section 4 and finally, we give conclusions.

II. SEARCH FUNDAMENTALS

Search might be a kind of new technology for many software applications. Production teams and management generally require a fundamental understanding of the technology before it can be introduced into production systems. Based our own experience and questions we encountered, we will try to answer the following questions, to give an introduction to search, rather than explaining the complex indexing mechanism or other technical details.

- Why do we need search?
  - *Search and SQL*
- How to give good search results?
  - *Ranking and navigation*
- Can it scale out?
  - *Big data search*
- Is it production ready?
  - *Search software*

These questions will be answered in the following subsections.

A. Search and SQL

With both SQL Relational (or NoSQL) database systems and search platforms such as Solr, data can be stored, indexed, and retrieved. Many databases also have advanced full-text index and search capabilities, for example PostgreSQL and MySQL. Since, with most existing applications, such as network management systems, data is stored in a traditional database, it is a common question to be asked, why should we use or migrate to a search platform for our use cases.

There are overlapping functions between database systems and search platforms. However, the two types of systems have different focuses making them better for different scenarios. Search engines focus on storing and querying indexes of data rather than the data itself. It is possible for data itself to either be stored within the search engine or stored outside the search engine. Hence, search engines do not offer database features for storing data, such as complex tables/schemas for data modelling, ACID (Atomicity, Consistency, Isolation, and Durability) properties for database transactions, etc. but do provide advanced search features such as sophisticated ranking models, highlighting, etc.

Search engines can offer various advantages over SQL database in many use cases:

- Advanced search features

Many advanced search features such as auto-suggestion and proximity search are not available in database systems due to the different focus of such systems. In many cases, search engines are used as a secondary index of a database to enhance the search capability while also reducing full database scans.

- Fast query speed

Search engines are designed and optimized for finding relevant documents from built indexes for a search query. Unlike a database, it does not need to perform a full table scan that would be required for a simple wildcard-based text SQL search. It does not have many of the constraints of database design, such as how quickly an individual document can be updated or retrieved. This means that search engines generally have faster query response times compared to databases.

- Data information centralization

Enterprise data is normally stored in various data stores, such as SQL databases, content repositories and file systems. It is difficult for users to link information between different data sources. A search engine has the ability to handle various data sources and types (structured and unstructured). This makes it possible to provide an information portal with all the enterprise data in a single interface.

- Google like experience

Both SQL and Solr queries have expressive syntaxes for complex data queries. A SQL query is designed for structured data and requires specified data tables to match users' data queries. In a search engine, data is de-normalized documents. A simple query can start with any terms or words just like Google search. It is not necessary for users to know any query syntaxes or data schemas to start data discovery.

B. Ranking and Navigation

Search applications are not only able to retrieve matching information; the most important part is being able to find the 'right' information to answer the users' questions. There could be millions or billions of matching or relevant search results; there are only a very small amount of results that a typical user is willing to browse. It can be like trying to find the right drop in an ocean for users. Two fundamental approaches to tackling this problem in search application development are: ranking and navigation.

- Ranking

Ranking can determine the most important (top-ranked) query results based on ranking algorithms. Search engines generally have one or more ranking algorithms built in, such as the Vector Space Model (VSM) based algorithm in Lucene [4]. These algorithms are fairly complex and consider many factors to rank results, for example, how important a word is to a document over the whole document collection. Different algorithms may be needed for better results for different application cases. For example, to measure importance of web pages, (Google) PageRank is a link-based ranking algorithm that takes into account hyperlink information between web pages.

Ranking is one of the fundamental problems in information retrieval. Optimizing existing or developing new better ranking algorithms requires a huge amount of

scientific and domain knowledge, and could be a difficult task. In most cases, application developers probably just use the built-in ranking algorithms with some offered customization functions. For example, a 'boosting' function can assign more weight to words in the title than the content of books.

- Navigation

Navigation gives a UI interface (e.g. advanced search page) to allow navigation of search results. It, like a file manager or browser, has research results organized in different 'directories' or 'filters'. It is a very common UI feature in search engines, especially in e-commerce websites. For example, clothes can be categorised by different sizes, price, etc. It helps users to quickly reduce the search scope through clicks. It makes it easy for untrained users to find the specific data they are interested in. Two common techniques are available for developing search navigation: faceted navigation and document clustering.

Faceted Navigation: is based on faceted classification which classifies documents using multiple taxonomies (sets of attributes or facets). For example, a collection of books might be classified using multiple attributes such as author, title, date, etc. Hence, allowing users to explore a collection of information by applying multiple filters. Faceting is an available feature in Lucene and Solr for application developers to directly use in a search UI.

Document clustering: is based on cluster analysis of document contents to allow automatically grouping of documents into different topics or subjects. For example, results of a search for "Ireland" might be grouped into different topics such as sport, finance, food, etc. As a result, users can quickly select interesting topic groups and filter out undesired ones. This feature is offered as a Solr search component.

C. Big data search

A critical challenge in a variety of industry sectors, such as telecoms, nowadays is that IT applications need to handle very large and complex data sets, which are difficult for traditional data management or data processing systems to handle. This makes "Big data" one of the hottest topics in IT industry today. The challenges include various data related tasks such as collection, storage, analysis and of course, search and discovery.

Scalability for big data problem does not exist for current common search platforms, Solr or Elasticsearch. Elasticsearch was initially developed for the purpose of creating a scalable search solution. In the case of Solr, it offers SolrCloud for setting up a cluster of Solr servers to scale out and ZooKeeper is used for cluster coordination and configuration, just as in the Hadoop cluster. As the result, Solr search is an available add-on feature of many commercial Hadoop distributions, such as those from Cloudera and MapR. In these cases, Solr is one of distributed applications integrated into the Hadoop platform and managed by Hadoop management component (YARN) to improve the cluster efficiency.

SolrCould distributes both the index process and the queries automatically. It uses ZooKeeper to automatically elect a new cluster leader when a leader goes down. This avoids the single point of failure of a fixed master slave cluster. The main underlying concept of SolrCloud for data distribution is the same as database systems, by having a large dataset split into multiple shards. Shards are the partitioning unit for the index data, so that search load for that the dataset can be split across multiple servers and search results are merged across those shards.

D. Search software

In the market of search engines or platforms, there are plenty of commercial products which offer great features, such as Splunk. Nevertheless, we were more interested in open-source projects for research prototype development, as it has the convenience of being able to study the internal technical details and implement customized features. Moreover, many commercial search products are built on the open source projects. For instance, Lucidworks Fusion, Cloudera Search, etc. are built on Apache Solr.

Search engines implement the various complex search-related operations, such as index building and querying. Search platforms use the search engine under the hood and build additional functionalities around it, such as scalability, administration and filtering. There are dozens of open source search engines and platforms [5]. Apache Solr and Elasticsearch are the most popular open source search platforms built on the Apache Lucene search engine [6] at the moment. Since both platforms use a same engine and are very actively developed, it is hard to find any significant advantages of one over the other in their most recent versions. Nevertheless, they are backed by different big data vendors making it much easier to pick if a commercial Hadoop distribution is already used in production. As a consequence, our paper is mainly about a study with Solr.

We give a quick summary of the concern regarding technology maturity and commercial support for production development. Search software and vendors are well developed. Open source search platforms Solr and Elasticsearch are proved production ready and with commercial support available. Solr is integrated and offered in big data platforms of all three major Hadoop vendors (Cloudera, Hortonworks, MapR).

III. SOLR SEARCH PLATFORM STACK

Search platforms are not majorly different from databases or data warehouse systems from a conceptual view. Search platforms are considered NoSQL data stores by many people. In databases, data is stored into structured tables, generally, and then queried from those tables. For search platforms, data is indexed as documents and then the documents are searched from the index.

We classify and present the search platform in the following layers from bottom to top; from the various data sources in the data layer that would like to be searched to allowing the user to submit a search query and displaying results in end user UI applications.

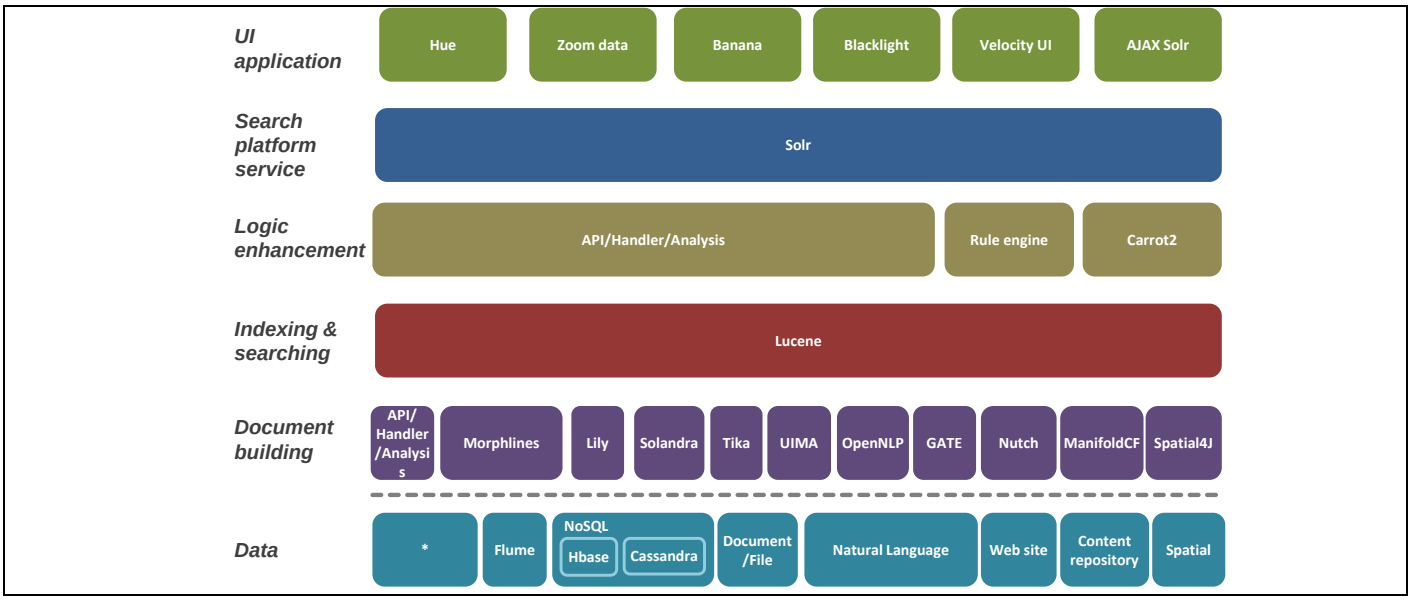

Figure 1: Solr search platform stack

- Data
  - Purpose: Represent various data types and sources
- Document building
  - Purpose: Build document information for indexing
- Indexing and searching
  - Purpose: Build and query a document index
- Logic enhancement
  - Purpose: Additional logic for processing search queries and results
- Search platform service
  - Purpose: Add additional functionalities of search engine core to provide a service platform.
- UI application
  - Purpose: End-user search interface or applications

In the following, we are going to detail each layer and the corresponding software frameworks or components (Fig.1). Since Solr was selected for our study, as discussed in the previous section, only some major software related to Solr will be covered in here.

A. *Data*

One of main advantages of the search platform compared to a database is that it can handle both structured and unstructured data. This means the search platform has the capability to index and search data from existing database, file repository, etc. The data layer represents these various data types and sources.

Since the data sources are external or not tightly coupled to search platforms and there are a huge number of data storage software solutions, we will not detail the software component for the data layer. However, search functions might be only able to retrieve search results to work with external data storage together if data itself is not stored inside the search engine. Depending on the design of the search application, the search engine could store only indexes of the data while the data itself can be stored separately outside the search engine with links to the indexes. Search results can be retrieved from the external data storage based on the link information. For example, web search engines normally do not save the full content of Web pages from external web servers but instead just store page indexes and the URL of pages. Search engines used as secondary index of databases such as Hbase, are similar examples.

B. *Building documents*

For common database systems, a Table is used as a data structure to store related information. It consists of fields (columns), and rows. Multiple tables can be formed for different topics in a logic database, such as employee (name, department, role…) table, customer (name, address, sex…) table.

In the concept of a search engine, documents are the unit of indexing and search. A Document is a set of fields. Each field has a name and a textual value. It is just like a paper document, which has title, author, date, etc. Unlike databases, which can have multiple tables in a logical database, all data in a search engine must be de-normalized in to a single defined document schema in a logic document collection (Fig.2).

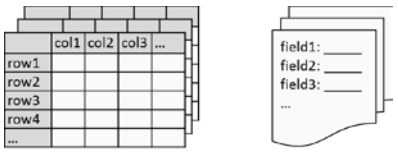

Figure 2: Tables in a Database (left) V.S. Documents in a Collection (right)

Hence, for a piece of data, regardless of if it is structured or unstructured, all searchable information of the data must be extracted and converted to a document, so that it can be indexed and searched afterward. Due to the complexity and variety of data, additional software components to the Solr API, handler, etc. might be used to build documents, for example, for extracting information from audio files or web pages; or for language translations.

Nutch

Highly scalable web crawler

- An open source Apache project
- Tightly integrated with Solr
- Modular, extensible architecture that allows adding of plugins
- Distributed to provide scalability and reliability
- Supports different storage back-ends, such as Hadoop, Hbase, etc. (Hadoop was a spun out subproject from Nutch)
- Supports parsing with Tika

TIKA

Toolkit for detecting, parsing and extracting metadata and text content from files for indexing

- An open source Apache project
- Integrated into Solr via a plugin that comes, by default, with Solr.
- A simple unified interface for all parsers.
- Allow adding of new parsers as plug-ins.
- Support over a thousand different file types, such as HTML, PDF, XML, audio, video, etc.
- Able to detect the language of a document.

UIMA

UIMA stands for Unstructured Information Management Architecture. Framework for transforming unstructured information, such as text, audio and video into structured information

- Apache project, originally developed by IBM and used in the Watson project.
- Native support for distributed computation for scale out
- Define custom pipelines of Analysis Engines (annotators) which incrementally add metadata to the document via annotations.

OpenNLP

Machine learning based toolkit for the processing of natural language text

- Apache project, originally developed by IBM.
- Supporting many NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, etc. with machine learning processes.
- Can be run under UIMA as a plugin

GATE

A suite of tools for text processing

- Open source project, originally developed by University of Sheffield
- Similar to UIMA, has a GATE-UIMA interoperability layer for combining GATE and UIMA

Kuromoji

A Japanese morphological analyser that provides the Japanese language support in Solr

- Apache project originally developed by Atilika
- Tokenisation of Japanese text
- Supports part-of-speech tagging

Smartcn

A library for analysing Chinese text that provides the Chinese language support in Solr

- Part of the Apache Stanbol project
- Tokenisation of Chinese text

Spatial4j

A geospatial Java library with Solr integration

- Open source library
- Provide common shapes that can work in Euclidean and geodesic (surface of sphere) world models
- Provide distance calculations and other geospatial mathematical functions
- Read shapes from WKT formatted strings

Lily HBase Indexer

A tool for indexing data stored in HBase

- An open source project developed by NGDATA and Cloudera
- Allows you to define indexing rules
- Designed to scale
- Indexes asynchronously so that HBase performance is not affected

Solandra

Integration of Solr and Cassandra

- An open source project developed by DataStax
- Uses Cassandra as storage to allow Solr to scale to huge numbers of documents
- Supports most Solr features
- Data is available as soon as the write succeeds

Morphlines

Framework for easily developing Hadoop processing applications for loading data into Solr, HBase, HDFS or other data warehouse.

- An open source project developed by Cloudera
- No need to code, just simple configuration

C. Indexing and searching

Indexing is the process of converting document fields into a format or index that facilitates rapid searching. A simple analogy is an index that you would find at the end of a book: That index points you to the location of topics that

appear in the book. These functions are implemented by a search framework or library which is the core engine of a search platform.

Lucene

Java search engine library for indexing and searching

- Apache project.
- Supporting field-specific indexing and searching, sorting, highlighting, and wildcard searches, etc.
- With some state-of-the-art ranking algorithm implementations, such as Vector Space Model (VSM).

D. Logic ehancement

It is sometimes necessary to apply additional processing logic to search queries, search results, etc. in many application use cases. For example, making searches return the first paragraph of a text document field if the text content is too long. In addition to the Solr API, handler, etc., there are other Solr add-on components, which developers can use that allow them to specify processing logic.

Business Rules for Solr

A module for Solr that integrates with rules engines.

- The feature was not completed in Apache Solr, but a proprietary module is available from LucidWorks
- Allow modification of queries, search results or documents before they are indexed by business rules.
- Integrate with Drools and other rules engines

Carrot2

Search results clustering engine

- An open source project included with Solr as a plugin
- automatically cluster small collections of documents,
- With specialized document clustering algorithms implementations

E. Search platform service

Beside core indexing and search functions provided by search engine library, i.e. Lucene, additional functionality like HTTP APIs, administration interface; scalability, content parsing, etc. are also commonly required for building search applications. As a consequence, applications, such as Solr, are built on top of the Lucene core engine to provide a search platform service.

Solr

The search engine interface to the Apache Lucene search library

- Apache project. Originally developed by CNET networks.
- Based on Lucene and backed by Lucid imagination
- Has a simple REST based query interface
- Includes plugins for many frameworks described, such as Tika, by default.
- Scalable using SolrCloud, which provides sharding and replication

F. UI application

An end user application requires a UI for submitting search queries and browsing search results, similar to a database browser.

Velocity UI

Built-in default Solr search UI

- Based on Apache Velocity project
- Simple customizable UI based on Velocity templating

Hue

Hadoop Web UI contains Search UI for Solr

- Open source project developed by Cloudera
- Dynamic search dashboards
- Many customizable search widgets, e.g. data table, bar chart, time lines, map etc.

Zoomdata

Visualization & Analytics Platform

- Proprietary software
- Built-in Solr connector
- Advanced visualization & analytics dashboard

VuFind

A library resource portal

- Open source project built on Solr
- Search and browse library's resources

Blacklight

Search interface for Solr

- An open source written in Ruby on Rails
- Customizable interface via the standard Rails (templating) mechanisms

AJAX Solr

A JavaScript library for creating user interfaces to Apache Solr.

- Open source project
- Some basic pager, tag cloud, map, etc. web widgets are available

IV. ADVANCED SEARCH FEATURE DEVELOPMENT

As we discussed, an important part of a search system is being able to quickly find the ‘right’ information to answer users’ questions or search queries. As a result, this was the innovation focus of our team and two advanced search features were developed for the context of network management systems.

In the following, due to the space and focus of this paper, we only give an overview of two innovative search features we developed for PoC. The detailed implementation and evaluation will be given in our future publications.

A. User experience based search recommendation

Browsing and searching network information for observation, analysis and troubleshooting is an inherent part of using the features and functions of any Network Management System. Improving the efficiency of the manner in which users can operate network management systems reduces the time taken to carry out processes and resolve issues, increases customer satisfaction and reduces operator costs.

Users of modern management systems must have a substantial amount of knowledge and experience to discover the information they need to carry out observation, analysis and troubleshooting activities, since both network and network management systems are becoming large and complex. For example, if a user knows that a range of a KPI values is commonly related to a problem, then they can quickly filter out network nodes which have values outside the ranges when the user performs a node data search. As a result, if this KPI value filter can be recommended to other users when they perform a similar search, then the search function becomes more efficient and the length of time taken to resolve a problem may be significantly shortened. Recommendation techniques are common in e-commerce applications [7] [8], for example, recommending news or movies to users and thus promote sales. However, it is a novel concept for search applications in telecom domain.

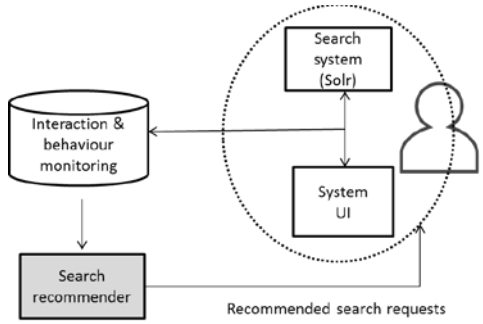

Figure 3 : search recommendation

The idea is that system learns or tracks user search experience as search transactions or logs (Fig.3). When a new search request is performed by a user, the system is able to recommend search requests based on past user experiences. The data captures the user search experience and is modelled as follows:

- A user search request/action part $V = [v_t, v_l, v_f]$, that can be further divided into three components; a term, a filter, and a feature. These three parts cover the user requirement of different aspects of a search. v_t is a user query which is composed of a number of keywords or terms. v_l is a set of filters applied on the search results such as a specified time range or restrictions on the data source, etc. v_f is a feature containing different setting on how the result is presented. For example, the result is sorted by time or by ranking score, the result is faceted by a RBS or/and severity, values are displayed in bar chart or pie chart, etc.
- A rating/feedback part r is a user rating of the results returned from the query part. It is measured by user interaction with search result. For example, mouse over results, clicking on results, viewing different result pages, scrolling windows, time spent on the page, etc. More user interactions on a search result page indicate that the results are more interesting to the users.
- A knowledge source part c represents where the knowledge comes from, such as a generic system user or a domain expert. It gives different weights for different user background and expert's experience is given higher weights to improve recommendation results.

When a user performs a new search request V', the recommend score of past searches in search log can be calculated based on $sim(V, V') * r * c$, where sim is a similar measure function, e.g. Pearson correlation coefficient commonly used for recommender systems [8] [9]. Search requests with top scores will be recommended to the user for the current search request. Hence, the user is able to apply the filters, etc. directly by clicking recommended search requests.

As a very simple example, when a user searches for a network node "RNC9", the system will recommend a time filter for last 7 days and a bar chart which facets the data results at a network Cell level. This is because the system learnt that pervious users seem more interested in cell level summary information of last one week when they search for a cell management node. In our study with more than ten network management engineers as users on prototype evaluations, all users gave positive feedback and thought that the feature is very useful to quickly zoom in on the important information. It also can avoid ignorantly search queries from users wasting system resources.

The search recommendation provides recommendations that draw on the experiences of previous users of the system in the form of recommended search, supporting data source filters, important time windows, relevant data fields or KPIs, or significant graphical charts to allow. It improves the search efficiency by leveraging past user experiences for network information discovery, analysis and troubleshooting.

B. Anomaly detection enhanced search ranking

Very large amounts of network data also create a great challenge for network analysis and troubleshooting. It is very hard to find valuable insights from large amounts of relevant data even if a user is able to retrieve all relevant information by search. For example, in our case study, there could easily be hundreds pages of related data records or events returned by searching for a network element with just a few minutes time range before a problem occurs. In such a case, the Solr features such as simply sorting data by time, or relevance based result ranking becoming inadequate. The Solr relevance based ranking returns the top ranked documents which closely relate to the search terms, or the network element we searched, but they do not necessarily have any association to the problem with the network element. There are many alternative ranking algorithms that exist, such as

web ranking [10] [11], but none of them addresses our problem for network troubleshooting.

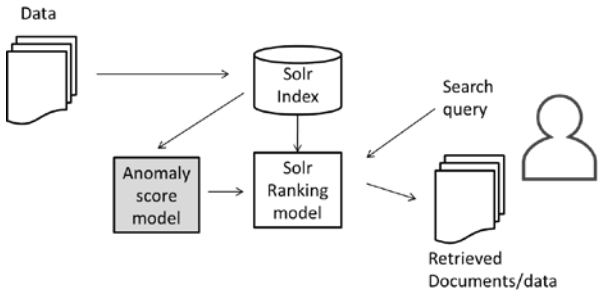

Figure 4 : Search ranking for network troubleshooting

Our solution targeted the problem by taking the anomaly score into account when ranking data for user search queries (Fig.4). For example, when a user submits a search query for a network element (e.g. RNC17), protocol (e.g. FTP) or any terms (e.g. CORBA), the top ranked search results are not only highly relevant to the search query, but also are highly abnormal compared to the rest of the retrieved data. Anomaly detection is not new for network problem detection [12], but integrating the technique to enhance network data search ranking is a novel concept.

The technique is briefly described in the following. Document results are divided as $D = \{D_{w1}, D_{w2}, \dots, D_{wn}\}$ a finite series of document bags in different sliding windows, e.g. 3 minutes. Then, we can have a data statistic collection $E = \{E_{D_{w1}}, E_{D_{w2}}, \dots, E_{D_{wn}}\}$ based on document bags. E_{D_w} is a data statistic model for each term indexed, such as document frequency or inverse document frequency . Given a data space with multiple terms $t_1, \dots, t_x$ and a data statistic collection $E_{D_{w1}}, \dots, E_{D_{wn}}$ for these terms, by using various anomaly detection techniques [13] [12], such as statistical test and regression. We are able to calculate the abnormal score for each term in different windows. If a data document contains an abnormal term, then the document is considered as a possible anomaly. The anomaly score of each term is used to enhance the original Solr document ranking model. i.e., documents/data records that contain terms with high abnormal scores are expected to be ranked at the top. If b is a set of abnormal terms that was detected for a window and d_i is a document, the calculation of ranking score for the document with a user query q can be briefly described as:

$$score(q, d_i, b) = \frac{V(d_i) \cdot V\left(q \cap (\cup\, b)\right) \cdot S(b)}{|V(d_i)||V(q \cap (\cup\, b))|}$$

Where V is a vector function for converting any term set as vectors based on index data, and $|V|$ is Euclidean norm of the vector. S function is to boost score based on abnormal scores of terms in b.

To explain with a simple example, when a user searches for a node to troubleshoot some service downtime that happened the previous night and a number of records contain the terms “high”, “temperature”, “restart”, etc., documents containing the highlighted abnormal terms will be returned as top ranked results to give the user insight.

As a result, rather than simply sorting research results by time, term frequency, etc., top ranked search results are related to causes of network problems and give better results for network analysis or troubleshooting. It makes search more effective for network troubleshooting.

V. Conclusions

Browsing and searching network information for observation, analysis and troubleshooting is an inherent part of using the features and functions of any Network Management System. As enterprise search has capabilities of handling various data types and sources and big data scalability, it is becoming an emerging technology for such network management functions development.

In this paper, we have given an overview of work done in our research and prototype team regarding advanced search project. We provide a brief report on search fundamental knowledge and study of Solr search platform stack. It answers common questions from management and development team regarding adopting search technology for production development and gives a Solr reference card for developers. We described two advanced search features, user experience based search recommendation and anomaly detection enhanced search ranking from our research work. These features were developed to make network searches more efficient as it can help a user quickly locate the most valuable search results, but the concept can be adopted for other search applications. More detail of these two features and prototypes will be described in our future work.

References

1. Libes, D.E., E.L. Morse, and J.C. Scholtz;. A Study on Search Engine Use by Intelligence Analysts. in IASTED International Conference on Parallel and Distributed Computing Systems. 2006.
2. Ben-Yitzhak, O., et al. Beyond Basic Faceted Search. in International Conference on Web Search and Data Mining 2008.
3. Apache Solr. Available from: http://lucene.apache.org/solr/.
4. Grainger, T. and T. Potter, Solr in Action. 2014: Manning publications.
5. Middleton, C. and R. Baeza-Yates. A Comparison of Open Source Search Engines. in Technical report, Universitat Pompeu Fabra, Department of Technologies. 2007.
6. Apache Lucene. Available from: http://lucene.apache.org/core/.
7. Po-Huan Chiua, G.Y.-M. Kaob, and C.-C. Loa, Personalized blog content recommender system for mobile phone users. International Journal of Human-Computer Studies, 2010. **68**(8): p. 496–507.
8. Huang, Z., D. Zeng, and H. Chen, A Comparison of Collaborative-Filtering Recommendation Algorithms for E-commerce. Journal of IEEE Intelligent Systems, 2007. **22**(5): p. 68-78.
9. Herlocker, J.L., et al., Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems, 2004. **22**(1): p. 5-53
10. Bendersky, M., W.B. Croft, and Y. Diao. Quality-Biased Ranking of Web Documents. in ACM international conference on Web search and data mining 2011.
11. Manning, C.D., P. Raghavan, and H. Schütze, Introduction to Information Retrieval. 2008: Cambridge University Press.
12. Bhuyan, M.H., D.K. Bhattacharyya, and J.K. Kalita, Network Anomaly Detection: Methods, Systems and Tools. IEEE communications surveys & tutorials, 2014. **16**(1): p. 303 - 336.
13. Gupta, M., et al., Outlier Detection for Temporal Data: A Survey. IEEE Transactions on Knowledge and Data Engineering, 2014. **26**(9): p. 2250 - 2267