Proposed Method of Retrieving and Visualizing Search Results from the US Federal Register

Matthew J Wiecek
Texas A&M University
College Station, TX
matthewwiecek@tamu.edu

Chun-Chan (Bill) Cheng
Texas A&M University
College Station, TX
aznchat@tamu.edu

Divyesh M Tekale
Texas A&M University
College Station, TX
tekale2@tamu.edu

ABSTRACT

The US Federal Government posts many Notices, Proposed Rules, Final Rules, Presidential Documents, and other documents of consequence on a daily basis. Being able to search and navigate through all of these documents in a rapid fashion is of the utmost importance in many legal and political fields. We propose a system that will graph the search results in a hierarchical structure to clearly indicate which federal agency published the document. We further propose that the results be color-coded to easily identify the type of document retrieved.

CCS Concepts

•**Information systems** → **Presentation of retrieval results;** *Structured text search; Enterprise search;*

Keywords

Federal Register, Graph Output, Information Retrieval, Visualization, Solr

1. INTRODUCTION

The US Federal Register is the official Daily Journal of the United States government. All Federal Notices, Proposed Rules, Final Rules, Presidential Documents, et cetera, are published by the Register. Any document published in the Register carries legal significance as they are considered to constitute legal notice to the public. Being able to rapidly search through the Federal Register and interpret the results is a valuable ability.

In particular, the layout of the search results is crucial. Upon presentation, the user should immediately be able to identify the type of document as well as the the publishing agency. Owing to the hierarchical structure of government, a graph structure can quickly represent which agency published a document and where it lies in the federal bureaucracy. A color coding scheme may be used to help the user identify where they are in the graph.

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

2. PRIOR WORK

2.1 Crawling

Crawling the content is the first step to building any search engine. As the Federal Register has an API available, web crawling becomes simpler. Instead of trying to extract data from HTML, we simply request the data and get a response in an easily parsable format. Crawling RESTful services has a long history. Indeed, there exist systems, such as RESTLER that exist to not only crawl a RESTful api, but to also map it out and visualize the various resources the API made available [1].

2.2 Search

Likewise, search has a long history. Modern search engines can take advantage of inverted indexes and can score documents with tf-idf [6] whilst tools, such as PageRank, use weblinks to rank the prestige of websites based on the webpages that link to them [4]. Indeed, there exist robust open source search engines like Solr, which we use in our work.

2.3 Visualization

The field of visualization is known to most as the method through which information is made understandable. To that end, hierarchical node-link diagrams have been around for quite some time, with research done on attempting to make more than 20-30 nodes intelligible at a time [5].

In the context of web search, much effort has been made to attempt to give the user more information about their search results. ResultMap was an attempt to use treemaps to give the user information about what type of content was found by the search. The treemap was presented alongside a traditional list of links to the content [2]. A method of node-link visualisation was described in a US patent primarily concerned with context management[3]. That is to say, the document retrieved by the search query would be clustered into groups, and these groups would be clustered as a node-link graph. The user interacts with this graph to adjust the weight of the search results displayed.

3. NOVELTY

The novelty of our work lies in the visualization system. As opposed to providing visualization of the search results alongside weblinks, or using a visualization of the topics of the results to be able to adjust which results to display, the results themselves are displayed via a node-link graph to not only help the user rapidly find what they are looking for, but

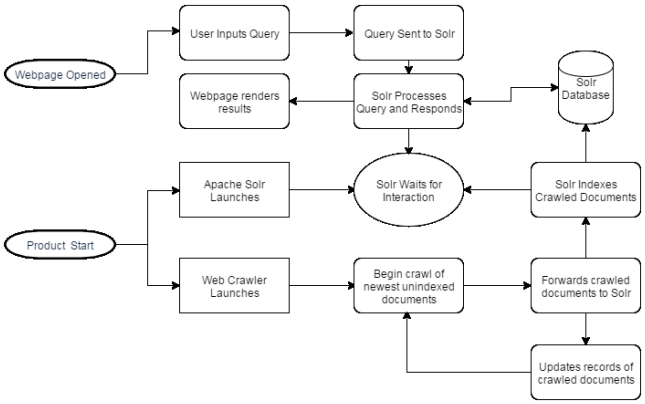


Figure 1: The sequence of events in the application

to also provide the user with context about how and through which agencies the US Federal Government interacts with the topic presented by the user query.

4. HIGH LEVEL OVERVIEW OF SYSTEM

4.1 Subsystems

The system shall be made up of four distinct subsystems: the crawler, the search engine, and the user interface. The crawler will interact with the Federal Register API to fetch the documents, the indexer will process them into a format that is acceptable to the search engine and the search engine shall serve user queries and return a list of documents that match the query. The user interface shall be responsible for handling user interaction and displaying the search results returned by the search engine.

4.1.1 Crawler and the Federal Register API

The Federal Register maintains an API to allow developers access to federal documents.[1] As the API has a limit of 1000 documents per request, the crawler shall request the full set of each day's documents one day at a time. It is not expected that there shall be more than 1000 documents published by the Federal Government on any given day. The response shall be parsed and returned to the search engine for indexing.

4.1.2 Search Engine and Apache Solr

The search engine used will be Apache Solr.

4.1.3 User Interface and the Web Browser

Users will interact with the search engine via a web interface. The user will be free to input a query via a text box. There will be further filtering options available, such as restricting the search to certain federal departments or agencies. The user query will then be passed to the search engine via an HTML GET request. Upon receipt of the response from the search engine, the web page will be responsible for drawing the graph and color coding the elements.

4.2 Timeline of Events

There are two events during the application's life. The initial launch of the application, and a user query.

[1]The documentation to the API may be found at https://www.federalregister.gov/learn/developers

4.2.1 Launch of Application

Upon launch of the application, both Solr and the Web Crawler will launch (see Figure 1). Due to the 1000 item limit per API call imposed by the Federal Register API, the crawler will request documents one day at a time. Once the documents have been fetched and processed, the crawler shall store the documents for future indexing. The crawler will the update its own internal record of which days have already been crawled, and begin crawling the earliest day that has not yet been crawled. This will continue indefinitely until program termination, an error occurs within the crawler, or a user defined amount of days has been crawled.

4.2.2 User Query

Once the user enters a query, said query is forwarded via an HTTP GET request to Solr for fulfillment. Solr will return a list of potential documents that match the result with associated meta-data. The web page will render the results and display any spelling recommendations that have been executed. At this point the user will be free to interact with the search results, such as expanding them to get the abstract, or a link to the document itself.

5. LOW LEVEL OVERVIEW OF SUBSYSTEMS

5.1 Web Crawler & Federal Register API

The webcrawler will be written in *Java* and will be composed of several individual subsystems.

5.1.1 Query Generation

The Federal Register maintains an API that is accessible via standard HTTP GET requests. The request has been crafted to request a JSON object containing the links to the XML versions of each published document. As the API sets a maximum limit of 1000 items per request, a query will be generated to request each day's published documents; the assumption is that there will never be more than 1000 unique documents published in any given day. The class *Generate_Queries* is responsible for generating the web requests. It implements the runnable interface, allowing other threads to run as it generates the queries. The generated query is then pushed into a *BlockingQueue* which is thread-safe so that other threads may execute the queries.

5.1.2 Fetching & Parsing JSON Response

Once the queries have been generated, an HTTP request has to be served over the network to retrieve the JSON object from the Federal Register API that contains the links to the XML versions of that days documents. This is the responsibility of *Get_List_Docs*. It fetches the queries from a *BlockingQueue*, sends the GET request, and parses the JSON response. The extracted URLS are put into a different *BlockingQueue* so that other threads may fetch the documents themselves. This class also implements the runnable interface to parallelize the fetching of the JSON responses. As there is no interaction necessary between different *Get_List_Docs*, many threads can be spawned to ensure that the network is saturated.

5.1.3 Retrieval of Published Documents

Once the URL of the XML version of the document has been retreived, the actual document must be retrieved over

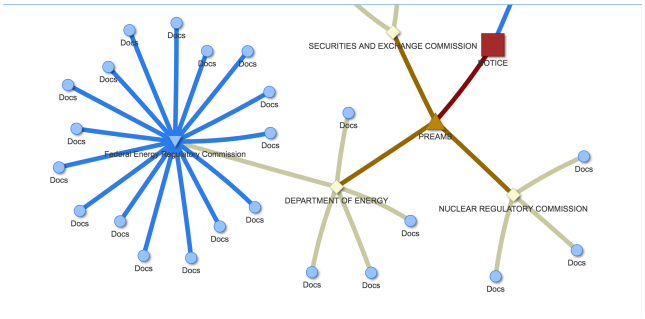


Figure 2: Zoomed in UI

the network. The class *Get_Doc* retrieves the actual document itself. It, also, implements the runnable interface. The number of threads can be increased to ensure that the network is saturated. It fetches the URL to the XML version of the document from a *BlockingQueue*. After fetching the document, this thread will save the fetched document to disk.

5.2 Indexer

The indexer will be written in *Java* and will interface with Apache Solr via the *Solrj* library.

5.2.1 Raw Data & Crawler Intergration

It is clear that before data may be indexed, it must first have been crawled. The web crawler dumps each crawled document into its own text file. The indexer will then read the text files from the directory in which the crawler has put the text files; these are stored as strings in a *BlockingQueue*. The indexer threads will parse the XML and extract the relevant fields. The reason for such separation is that it allows indexing and crawling to be done independently. This minimizes web traffic to the Federal Register and maximizes politeness. If the documents have to be re-indexed due to a change in the schema, we do not need to burden the Federal Register with a request for already crawled documents.

5.2.2 Indexed Fields & SolrJ intergration.

Solr takes in various fields for indexing. There are four well defined types of documents published in the Federal Register: Notices, Proposed Rules, Final Rules, and Presidential Documents. The indexing threads pull the raw strings from the *BlockingQueue* and parse the string as an XML document. First, the type of document is identified. Once the document has been identified, the appropriate XML fields are extracted as defined by the schema in section 7. The fields are then added to a *SolrInputDocument* and are then passed to the Solr server by the *HTTPSolrServer* class provided by the *Solr* Java API.

5.2.3 XML Parser

All the pages we crawled from the federal register was in xml format, so we had to read through the xml data with org.w3c.dom.NodeList library. We parsed the key words `TYPE`, `AGENCY`, `AGENCY TYPE`, `SUBAGY`, `SUBJECT`, and for `PRESDOC` we parsed `HD`, `FP` to obtain the information to satisfy the schema in section 7.

5.3 Application

5.3.1 User Interface

The results will be drawn graphically using *JavaScript*. The United States Federal government forms a hierarchical structure with the federal cabinet departments at the highest level, and various executive agencies reporting to them. To represent the publishing agency of a document, we draw out a graph with the Federal Register in the middle and types of documents branching off of it (see schema in Section 7). The cabinet departments will branch off of the document type and executive agencies branching off from their respective departments, with documents as leaf nodes branching off from their publishing agency.[2]

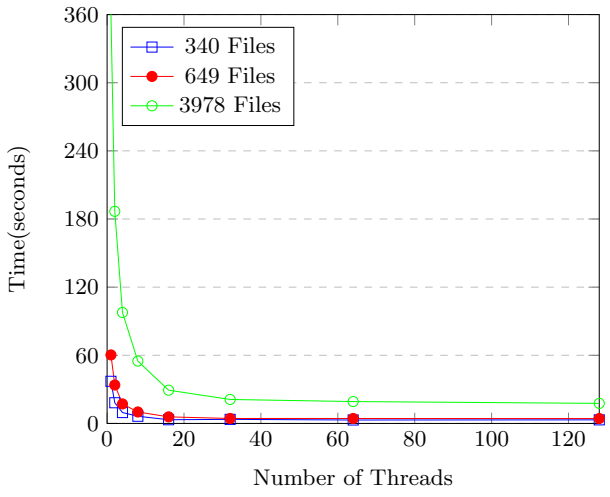
As there may sometimes exist joint notices and rules posted by various executive agencies, a single leaf node document may have several parents. To help make the tree readable, different colors will be used to represent different levels of the tree. For instance, a cabinet department will be differently coloured than the agencies withing them, see figure 5.2.3. Users may hover over any node to get flavour text of the abstract of the document. Clicking on a document will open a new page to the document on the Federal Register's website.

5.4 Obtaining Search Results

When a user types in a search query, it is passed to Solr via a simple HTTP get request. Solr returns the search result as a JSON object, which is then drawn as a hierarchical tree. The amount of search results returned is limited to 25 to avoid overwhelming the user by producing an unintelligible graph. The user will have standard next and previous buttons to be able to go through the search results in chunks of 25.

6. STATISTICS ON GETTING THE DOCUMENTS

The following figure is the statistics we ran on different number of threads to fetch the data from the webpage.



[2]Independent executive agencies fall somewhat outside of this hierarchy. At present they will be displayed along cabinet departments.

In the figure, we have three lines indicating different numbers of data we crawled with the x-axis as the number of thread and y-axis as the system time. It can be seen that there is a limit in increasing the threads to increase the speed of fetching the webpages. The slope gradually slows down when the thread numbers gets to 32. In conclusion, we fetched 1400 days (11000 documents) in less than three minutes with 100 threads running.

7. INDEXING SCHEMA

As there are four distinct types of documents published in the Federal Register, it is necessary that a different schema is in place for indexing each type of document.

7.1 Notice

The notice is perhaps the most common document published in the Federal Register. When indexed, documents of this type will contain the following fields:

Type of Document shall always be "Notice"

Cabinet Department the federal cabinet department which issued the notice

Agency the agency within the department which issues the notice

Action description of the type of publication

Summary a brief summary of the notice

Document ID the unique ID code assigned by the Federal Register for easy lookup

7.2 Proposed Rule

A proposed rule is issued as a guidance document to the public. It describes the current draft of a rule an executive agency is considering imposing. The publication of the draft rule gives the public a chance to provide feedback about the rule as well as giving the public time to prepare for the effects of the potential rule.

Type of Document shall always be "Proposed Rule"

Cabinet Department the federal cabinet department which issued the proposed rule

Agency the agency within the department which is issuing the proposed rule

Action description of the type of publication

Summary a brief summary of the notice

Document ID the unique ID code assigned by the Federal Register for easy lookup

7.3 Rule

A rule is issued as notice that a proposed rule has been finalized and is now in effect. It serves as both a means to communicate to the public about the new rule, as well as being the legal source of the rule, which may be referenced in court.

Type of Document shall always be "Rule"

Cabinet Department the federal cabinet department which issued the rule

Agency the agency within the department which is issuing the rule

Action description of the type of publication

Summary a brief summary of the notice

Document ID the unique ID code assigned by the Federal Register for easy lookup

7.4 Presidential Document

A presidential document is one which has been issued by the President of the United States. They may be executive orders, notices, or other noteworthy documents the President has decided to publish. These do not conform to the same structure as other documents published in the Federal Register, and thus, are harder to index.

Type of Document shall always be "Presidential Document"

Header the header of the document

Title from which Title of the US Code does this document derive it's authority (if applicable)

Document ID the unique ID code assigned by the Federal Register for easy lookup

8. REFERENCES

[1] R. Alarcón and E. Wilde. Restler: Crawling restful services. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1051–1052, New York, NY, USA, 2010. ACM.

[2] E. Clarkson, K. Desai, and J. Foley. Resultmaps: Visualization for search interfaces. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1057–1064, Nov 2009.

[3] A. Ershov. Context-based search visualization and context management using neural networks, Jan. 6 2009. US Patent 7,475,072.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[5] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[6] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.