IMPROVING USER EXPERIENCE

BY IMPROVING SEARCH RESULTS

Can the speed of your website really affect its bottom line?

When it comes to page load speed, faster is always better. Page loading time is the major contributing factor in website abandonment. In simple terms, in today's online environment, the average user has no patience for a website that takes too long to load.

Both Amazon and Google reports a significant drop off in user interaction with every increment in page load time. When it comes to searching, Google reports a 20% drop off in traffic with just a half a second's delay[1]. If a user is searching for something on your site, it already means they can't find what they're looking for. A slow site search will just compound their frustration and drive them away from your site and business, losing you customers and badly affecting your bottom line.

SEARCH THE OLD WAY

Traditional (or relational) databases weren't designed and aren't built for search.

They do well in situations where data can be neatly partitioned into columns, and where everything is correctly spelled, however real world data, and your client's brains, don't work that way. Search results can vary from nearly correct to wildly off target for various reasons:

- People struggle to remember the exact term or spelling they were looking for.
- The term they're looking for is embedded in 1000 word documents with any number of comments.

- The ambiguity of natural languages causes a plumber to see results for the skate park's (more popular) halfpipe than for the pipe he's looking for.

The reason for this inaccuracy is that traditional database searching uses any number of combinations of the LIKE '%term%' clause, where large text fields are scanned for a specific term. What makes it even worse is that the indexing optimizations normally available to relational data is not available when searching like this.

There are other strategies to try and improve your search speeds in a relational database, but it will only address one of a number of issues:

- Slow searches due to wildcard matching
- Different forms and variations of a word is ignored. Searching for "moved" won't return results for "moving" or "mover".
- The context of the document is discarded, causing irrelevant results to be ranked highly.
- Misspelt words in the documents are ignored and searches for misspelt words return little or no results.

SEARCH THE NEW WAY

To address the shortcomings of searching through relational databases, techniques around full text searching has been developed.

Traditional databases like MySQL and Postgres have retrofitted full text capabilities onto their relational database engines. If you have a solution that relies strongly on relational data but need a proper search solution you have the option to use the retro fitted full text search functionality in addition to the relational functionality.

Since the amount of information available is only getting larger, being able to find relevant information becomes more and more important. This realisation lead to the development of databases focused specifically on improving searching. Some solutions are classified as NoSQL and focuses on unstructured or loosely structured documents. These include databases like MongoDB and CouchDB. Other solutions are classified as document stores where a document is stored in it's entirety but indexed to enable efficient searches. Document stores includes Lucene, SOLR, and various other databases like Elasticsearch which is built on Lucene.

These databases give you a number of advantages over traditional databases right out of the box:

- Searches are specifically optimized for large documents containing a lot of text.
- Fuzzy searching is supported through stemming and other techniques.
- Misspelt words are handled correctly by checking the Levenshtein distance between words.
- The relationships between documents can easily be highlighted.

Word Stemming

Word	Prefix	Stem	Suffix	Ending
understatements	under	**state**	ment	s
reactions	re	**act**	tion	s
disabilities	dis	**able**	ity	s
unrealised	un	**real**	ize	ed
incredibly	in	**credit**	able	y

Most words can be broken into a prefix, stem, suffix and ending. Elasticsearch allows you to search on the stem of both the search term and the search subjects.

Building your searching functionality on top of such a document store will go a long way towards returning results that are:

- Delivered more quickly.
- More accurate.
- More relevant.
- More fault tolerant.

Example

A good working example of the application of these techniques are Amazon's highly effective cross-sell/ up-sell suggestions to customers:

EagerELK

USING ELASTICSEARCH TO IMPROVE SEARCH RESULTS

Elasticsearch is one of the new generation databases that provides a number of features to help you improve your search results:

Problem	How Elasticsearch solves it
Speed	• Elasticsearch allows optimization in both indexing (getting the data into the database) and searching (retrieving the data). • With its bulk API large numbers of documents can be indexed in a relatively short time. • Caching, filtering and various other optimizations allow for significant decreases in time taken for search queries. See the Growth Intelligence use case[4] for more information.
Word variations	Elasticsearch provides various language analyzers to aid in improving search results. Each of these analyzers provide various techniques, from stemming to tokenization, to help improve search results. These need to be tweaked depending on the implementation and the default language. See the Elasticsearch guide on Dealing with Human Languages[8].
Context	When returning search results, Elasticsearch gives you the option to give some context around the results, such as other categories or ranges that contain results that might be of interest to the user. This allows the system to guide the user to results that are there, instead of the user just making uninformed searches that lead no where. See the Quizlet use case[9] for more information.
Misspelt words	Elasticsearch has a fuzzy query[7] to address the issue where users mistype the term they are searching for. You can specify the fuzziness to allow for a greater or smaller range of misspelt words, and rank your results according to how closely the word matched. See the Globo use case[6] for more information.
Scalability	To ensure data reliability, Elasticsearch has redundancy built in. No extra configuration required. The data is distributed and replicated from the word go. This also allows the system to very easily scale horizontally, ensuring that you'll always be able to meet your customers needs, without having to sacrifice on performance. See the Github use case[5] for more information.

EagerELK

WHAT YOUR BUSINESS GAINS

Your clients and users might have experienced the issues discussed while searching on your site or through your content. As discussed, searches in your application can be improved in a number of ways, and with each improvement comes an increase in user satisfaction, stickyness and a potential revenue retention and increase for your business.

More about EagerElk

EagerELK is focussed on assisting its clients with rolling out Elasticsearch solutions. We provide a full stack service around the ELK stack - Elasticsearch, Logstash and Kibana.

For more information or to read our blog visit us on the web: **http://www.eagerelk.com**

REFERENCES

1. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html
2. https://lucidworks.com/blog/full-text-search-engines-vs-dbms/
3. http://en.wikipedia.org/wiki/Levenshtein_distance
4. https://www.elastic.co/use-cases/growth-intelligence
5. https://www.elastic.co/use-cases/github
6. https://www.elastic.co/use-cases/globo
7. http://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-fuzzy-query.html
8. http://www.elastic.co/guide/en/elasticsearch/guide/current/languages.html
9. https://www.elastic.co/use-cases/quizlet