MiniGoogle Search

Project assignment for Cloud Computing
2011/2012

INSTITUTO
SUPERIOR
TÉCNICO

MERC - Tagus Park,
Final Delivery
December 9, 2011

Report By:
Álvaro García Recuero   Mudit Verma   Manuel Bravo Gestoso

1. Introduction

This report details our experiments on cloud and methodologies we used for the Cloud Computing Project “Mini Search Engine”. Project is divided on different modules namely, Indexer, Crawler, Storage, Page Rank Map Reduce application & a front end web search page.

The first implementation was local and ran indexer, crawler, storage, map reduce, and searcher locally. However, for the final delivery we are using diffident cloud services in IaaS, Paas & SaaS domains. Specifically, our indexer & crawler is running locally, Page Rank is running on Amazon Web Services using the S3 Storage and indexer is storing the indexes (words with position) on the storage provided by the service Cloud Xeround which in turns uses Amazon at back end. The search webpage is hosted on Sigma which queries the database hosted on cloud.

2. Implementation

• Cloud platform and infrastructure

In order to provide a virtual location to deploy and test our applications we have used the following cloud computing services existing in the market or provided by internal IST resources:

Amazon web Services: Map Reduce application. We use S3 as storage of the Map Reduce application and Elastic MapReduce from Amazon for running Page Rank on the graph obtained from the crawler. Page Rank output is saved in S3 which is read remotely and stored in another database provided by the following infrastructure.

Xeround infrastructure: We found this the most efficient way of getting a free cloud relational database infrastructure for our project. Since we decided to use MySql as our database backend, we chose this cloud service as it provides the possibility of creating our instance as we would do locally, but instead having the possibility of accessing it remotely from the Internet.

Web browser: We are hosting our search browser at Sigma server so it is accessible from the cloud and able to communicate with the cloud database as well. This is composed of just a php script that performs the search query to the remote database.

Web crawler and Indexer: We use one application for crawling web sites and also indexing the words in them, we then start populating the data into the database tables we defined on to the remote infrastructure. This application also produces a graph that would be used later on as input to the Map Reduce Page Rank application. We use the Amazon S3 API to upload this input file to the corresponding S3 bucket and use it later on when running Elastic Map Reduce.

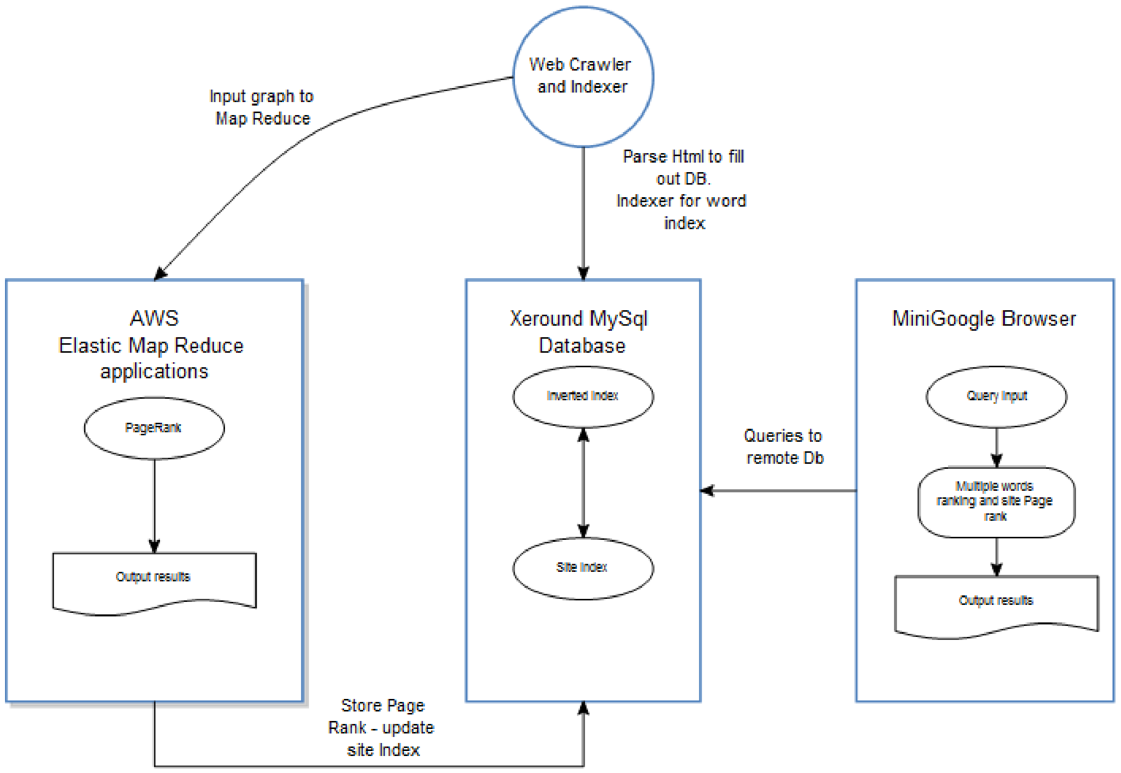
Web Crawler and Indexer
Input graph to Map Reduce
Parse Html to fill out DB. Indexer for word index
AWS Elastic Map Reduce applications
PageRank
Output results
Xeround MySql Database
Inverted Index
Site Index
Queries to remote Db
MiniGoogle Browser
Query input
Multiple words ranking and site Page rank
Output results
Store Page Rank - update site Index

Figure 1. Applications flow

Detailed Description of Modules and Features:

a. Web Crawler

For the first delivery we had a list of artificial pages made by ourselves, uploaded on sigma. We went page by page and found out links on those pages. In the end we had a graph which contained the adjacency list formed by the crawling. However, for the 2nd delivery, we have crawled the real internet deeper and deeper where graph contains thousands of nodes in its adjacency lists.

b. Web Indexer

For the first delivery we get a static list of pages and we index their content by saving the inverted indexes in local database. We use java threads to parallelize this process. In 2nd delivery we use the list of pages generated by Web Crawler, to index their content in parallel and store it in a remote database. We also save the position of each word in the page. In the end while searching for more than one word, we consider the proximity along with the Page Rank and display the pages accordingly.

c. Map Reduce application computing Page Rank on Amazon AWS

In our Map Reduce application we have two Mappers and two Reducers. First Mapper takes input from the graph, which contains a list of rows for each site ordered by: Site_id, Page_rank, Previous_page_rank, Degree (number of incoming links).

In the first Mapper and Reducer Job we perform the iterations where we calculate the new page rank using the current page rank using the page rank algorithm.

Later in the second Mapper and Reducer we check when the values converge. We do this ultimately in the Reducer by comparing the previous Page Rank with the current for all the key and value pairs.

At the end of the iterations, we define a set of properties to use during the process as follows,

- Threshold: we define a value to know when the difference if the previous output iteration is small enough so we assume Map Reduce has converged. If there is only one value not small enough according to our value then we perform a new iteration.
- Maximum Iterations: as input to the application we set a number of iterations to be run.

d. Multiple word search with ranking based on proximity, occurrence & Page Rank

As an added feature, we are now able to do multiple words search (2 words) and rank the pages based on the proximity of the words in the string that is being searched, the page rank & the no of occurrences of a word in a given page. We use the following formula to calculate the new ranking based on the page rank, no of occurrence and the proximity.

For all the pages which have all the words in the given search String:

Proximity Avg.(Page i) = Avg (all possible (Position(word(x)) – Position(word(y)))
Factor f = max (proximity avg(page i)) / max page rank (page i)
New Rank (page i) = (1-m) * PR (page i) + m*(Max(PAi/f) – PAi/f)

PAi = proximity average of page i
0<m<1

When a website has just one of the word from the search string but contains multiple occurrences, we do similar calculations as above but in-state of using the proximity as a factor we use number of occurrences.

The factor d can differ based on the requirement that whether we want to give more importance to the page rank or to the no occurrences & proximity. However, we have considered the proximity to be most important followed by page rank and the occurrence in that order.

3. Experiments

We have categorized our experiments in the following terms,

- Page Rank : Local vs. Cloud

  Running the Page Rank on local machine with small graph takes less time when compared to running it on cloud with two instances. As per our understanding this happens because the graph is not big enough to get the benefit of multiple resources in the cloud. We are sure that, if we have a large graph (hundreds of thousands of nodes) and more instances for Map Reduce, we would have gotten faster results.

- Searcher and Result ordering

  We ran our search experiment using the different values of m (as defined above). With that we could notice the difference in the search results which is based on the fact that whether we want to give importance to page rank or to the proximity and occurrences of word in a page.

- Changing d factor in Page Rank

  By the changing the value of d we would see the difference in page rank of each page, for example if we decrease d, the websites which does not have lots of incoming links, will get better page rank and vice versa.

4. Challenges faced in different experiments after going to cloud

- Now our crawler is crawling the websites deeper and deeper, however when we performed our experiment on cloud we faced difficulties as it is highly un deterministic to find the size of each newly crawled website and the number of new websites found in each level. For example, we ran our experiment with Wikipedia, starting from one random article; however, with just 2 level deep crawling we found 97000 new web pages. When indexer started indexing all those 97000 websites, it was taking huge time, storage and bandwidth as all the content (index) was being stored on cloud. We performed the similar experiment with number of other websites such as google.com and academic websites. With just google.com which has very less content on its pages, with just 800 crawled pages we could find 600000 new rows to be inserted in the cloud storage. Since we have the limited and trail resources on cloud infrastructure this process was no so fast.

- Another monetary challenge we faced was with elastic map reduce on cloud, since we were having paid service, more we ran the map reduce more we were getting paid. It was also dependent on the number of nodes to be used in map reduce.

Keeping the above constraints in mind, we performed all our experiments using just two instances.

- With our multiple words search, we realized that the way we are storing the words and the positions in the database is not one of the best options. Actually for huge storage, instead, we should have used the other storage data structures where we could perform fast searches of words in order to calculate the proximity which is computationally complex.

5. Conclusion

Overall, it was a great learning curve for all of us as none of us had any back ground in cloud computing before this course. We learned in detail about the new technologies, the available cloud platforms and different resources we used. In this process we faced few difficulties which only increased our knowledge and if we work on Cloud again in future, this experience would come handy.