DM822, Cloud Computing
Project report:

Answering advanced data analysis queries on Google Scholar results using Hadoop and PIG

Nicola Prezza

January 25, 2013

Contents

1 Problem description and motivation

Google Scholar[2] ("GS" in what follows) is the most comprehensive *free* search engine for scientific publications. A query on the search engine results in a list of (mostly) accademic papers associated to the respective authors and, if possible, to the paper's download link. Although the capability to retrieve articles from a huge number of sources and the functionalities offered, the search engine doesn't offer ways to elaborate and analyze the data in complex ways (for example with relational queries). Moreover, the number of results is limited to 1000 articles: if the search is conducted with common keywords is very easy to go beyond this limit, with the result that a lot of articles (maybe of interest) will not be recovered. Another major drawback of the system is that the user cannot export the results of the query for further external analysis.

Freeware tools exist to address (partly) these limitations: an example is Publish or Perish (PoP)[4], a software that allows the user to export the results of a GS query in various formats (CSV, BibTex, ecc...). Unfortunately, all these solutions share the common problem that the number of results is limited to 1000 articles and don't provide methods to further analyze the extracted data. Moreover, sometimes these tools provide incomplete authors lists since the search engine truncates the authors list beyond a certain length.

The aim of this project is to address some of these limitations creating a personal *updated* database of the articles informations (title, authors, year, abstract) indexed by GS with the aim of Hadoop, providing PIG scripts to analyze the database contents and testing the system answering some queries (relational/advanced data analysis) otherwise impossible to answer with the simple instruments offered by GS.

2 Project goals

The project is divided in two sub-projects: data gathering and data analysis.

2.1 Data gathering

The implemented strategy can overcome the 1000 results limitation of GS: since is unlikely that a single author exceedes the 1000 publications limit, the search is conducted by author. The results are then merged by article title to form the database entries. This task is solved with a single MapReduce job: each mapper processes the name of a single author, extracts his publications from GS and for each article and each collaborator author emits key-value pairs of the type $< article\ title, author >$. The reducer then aggregates the authors associated to the same article (removing duplicates) and emits a key-value pair $< article\ title, list\ of\ authors >$ that will be stored in the user's database. This approach has also the advantage that, in the case of an article with truncated authors, the database will contain the list of *all* the authors since the same article will be found for all the authors (this is not the case for example in PoP[4], which provides only the truncated list of authors).

Before to perform the operations above described, the program has to obtain a list of authors. In the first version of the project this file was created with an iterated MapReduce job which at each iteration, starting from an authors list, retrievede from GS the corresponding articles and extended the list with the authors of these articles. This procedure is redundant since the same scraping activity is already done in the articles database builder; the new version of the program retrieves the authors list processing only the previous calculated articles list and thus avoiding the unnecessary data download from the search engine.

According to [3], the graph of (scientific) collaborations is formed by a big connected component and some minor not connected components. Moreover, the average distance between two authors is around 7.64 and the maximum distance is 15. From this statistics it follows that the output of the described algorithm will converge eventually (in less than 15 steps) in the list of almost all the articles indexed by GS (minor isolated components will not be found).

2.2 Data analysis

The second objective of this project is to use PIG and Hadoop to answer specific relational/advanced data analysis queries on the gathered data. The queries implemented are complex enough to be impossible to answer with the only use of GS and are:

1. Computing the collaboration distance between two authors (for example, computing the Erdös number of an author). Calculators of this type exist on the web[1], but are not up-to-date.

2. Trending of specific research arguments over the years. This calculator analyzes the titles and abstract contents computing the frequencies of the keywords (relative to the ground frequencies in all the articles) and computes a trending score of the topic for each year.

3 Mid-project accomplished goals

In what follows the mid-project accomplishments are summarized. The full project accomplishments and results are reported in the following sections.

3.1 Data gathering and algorithm development

To date (20-12-2012), the code for the data gathering objective has been developed and a preliminary list of 18310 articles (6.5 MB) has been obtained (running the code in local mode). The articles database has been built upon a list of 142 authors obtained after 2 iterations of the authors' list algorithm above described starting from one single author (the undersigned). The algorithms have been implemented as above described with an additional refinement due to the Google scraping policies: since the search engine tries to detect and block automatic scraping agents, the developed software submits queries to GS passing randomly through a list of 216 proxies. The list is stored in a file that must be passed to the program by command line. The algorithm changes the proxy randomly at every downloaded page, and with this strategy GS is not able to detect the scraping activity and block the program (as it happened the first time the program has been executed without the use of proxies).

3.2 Further refinement of project goals

Since it has been observed that most of the computation time is spent by the program trying to connect and to download the pages (the use of proxies has the drawback to slow down these operations), another level of parallelization will be added launching for each mapper several threads, each one processing one author. This can be obtained modifying the format of the author's list placing more than one author per line (comma-separated), so that each mapper will obtain a list of authors and process each of them in a separate thread.

For what regards the data analysis phase, the list of possible analysis procedure to be developed (not all of them will be implemented) has been fixed and summarized in the list in subsection 2.2. Nevertheless, one fixed goal is to implement the collaboration distance calculator in point (1) in a relational way through a PIG script. This script will consist in an iterative procedure which at each step i will obtain the authors X such that the collaboration distance between the main author and X is i. The procedure stops when the second author appears in the collaboration list.

3.3 Software architecting and coding

To date the developed software is organized in 4 packages offering the following functionalities: the package *web* contains the classes *HtmlHashParser* (a self-written html parser which uses a hash data structure to speed-up the content extracting), *Proxy* and *ProxiesVector* (proxies management, including random access to the proxies) and *GSscraper* (the main scraping class which provides high-level functions for the extraction of the data from GS). Package *utils* is designed to contain general-purpose objects used in the rest of the code, and for the moment it contains the class *StringStream*, designed to buffer the downloaded pages and to access them as a data stream. Package *mapred* contains the Map-Reduce code: the classes *AuthorsListCreator* and *DBCreator* implement the algorithms above described to build respectively the authors and articles databases. Finally, the package *db* contains the main class *ScholarDBManager* which implements the command-line user interface and executes the methods.

4 Project Timeline

The project timeline of the entire project is the following:

1. Firstly the authors list creator has been implemented and tested.
2. The articles database creator has been implemented and a small database was created (few hundreds of articles).
3. Due to the GS scraping policies, the program was no more able to download data (GS blocks the IP address for a certain amount of time if scraping activity is detected). A randomized proxy rotating algorithm has been implemented.
4. With the new working software an extended database of 18310 articles (6.5MB) has been obtained.
5. Multithreading has been added to the mappers to speed-up the pages download: multithreading amortizes the cost of the connection to the proxies and allows to exploit the available bandwidth at best.
6. The software has been deployed on Amazon EC2 servers and a list of 1.7M articles (622 MB) has been obtained rotating over a list of 1540 proxies.
7. A set of PIG scripts have been developed for the collaboration distance calculator. The calculator has been tested on small data sets (due to its high resource requirements and costs limits).
8. The topic trending calculator has been developed and tested on some sets of keywords to compare the respective trendings of the arguments.

5 Description of the approaches followed

What follows is a detailed description of the work carried out for this project.

5.1 Proxy rotation and multithreading

As stated before, one of the major bottlenecks of the data gathering phase is the connection to the proxies and the data download. After a request, usually it takes a few seconds for a proxy to answer; moreover, in a great number of times the proxy refuses the connection after the delay. The average number of requests (pages) per author is 2 (each page showing 100 articles for that author); multiplied for the total number of authors ($\approx$ 25K in the biggest search conducted) this leads to an unacceptable computation time. To overcome this problem, multithreading has been added to the program: each authors list's line is in the form

$$author_1 \quad author_2 \quad ... \quad author_n$$

where the authors are tab-separated and n can vary from line to line. In the articles DB creator, each mapper creates n threads, each of them processing one author as described in the following subsection.

One of the biggest problems encountered in the database creation has been the retrieval of the data from GS whithout being detected by the search engine. GS analyzes the rate and distribution of the pages requests to detect automated searches. Once an IP address has been detected, it is put on a black-list and it's no more possible to download data with that IP. To avoid the IP blacklisting, a list of 1540 verified proxies has been used; the program rotates proxy at each request and once all proxies have been used the list is randomly permutated and scanned again from the beginning. This strategy allows to distribute randomly the requests and avoids that 2 consecutive requests are made with the same proxy. Even with this precautions, GS detected some of the proxies, slowing-down the data download. For this reason the biggest database obtained is not complete and contains approximately 1.7M articles (622 MB). The gathering of this information required 7 hours of global computation time on Amazon's servers "M2 high memory double extra large" (high I/O and CPU performances) with 300 threads per Mapper.

5.2 Articles database creation

The articles database creator has been implemented with a single map-reduce job which takes as input the list of authors and builds up a database containing all the authors articles indexed by GS. As described before, in the input file more authors can be grouped in a single line: Each mapper will create a different thread for every different author in the line processed. Each thread downloads the article's data of his author. Every article "*article*" in the results page is associated with a list of co-authors; for each of them (plus the searched author) the following key-value pair is emitted:

$$< "title", "author" \quad "citations" \quad "year" \quad "abstract" > .$$

where "*citations*" is the number of citations, "*year*" is the year of the publication and "*abstract*" is the truncted abstract that GS outputs in the results page. The reducers have simply to merge the authors of every article (removing duplicates). The key-value pairs emitted by the reducers (and then the database entries) are of the form

$$< "title", "author_1, author_2, ..., author_m" \quad "citations" \quad "year" \quad "abstract" > .$$

In the cases where a field is not present in the downloaded data the database creator simply inserts the string "NULL" in the place of that field.

The command to be executed to build the database is

```
hadoop jar SDBM.jar db.ScholarDBManager -new_db authors DB proxies
```

where *authors*, *DB* and *proxies* are respectively the paths of the authors folder, the destination of the database content and the proxies file.

5.3 Authors list creation

The list of authors can be simply obtained from the articles database with one single map-reduced job. Every mapper processes a line of the form

$$< title, "author_1, author_2, ..., author_m" \quad "citations" \quad "year" \quad "abstract" > .$$

and emits the key-values pairs

$$< "author_1", "" >, < "author_2", "" >, ..., < "author_m", "" >$$

The reducers then have simply to remove duplicates and emit one single key-value pair $< "author", "" >$ for every distinct author "*author*".

The command to be executed to build the authors list is

```
hadoop jar SDBM.jar db.ScholarDBManager -new_auth DB authors
```

where *DB* and *authors* are respectively the path of the database folder and the destination folder of articles list.

Globally, the articles database can be built iterating the execution of the two map-reduce jobs above described: firstly the database creator is launched using as input a list containing a few authors. Then a new authors list is created using the output of the database creator. Using this new authors list a new database is built and so on. Experiments show that the database size grows exponentially with the number of iterations: starting from one author, the number of articles obtained in the iterations has been 1, 169, 18301, 1701328.

5.4 Collaboration distance calculator

The collaboration distance calculator has been developed integrating PIG scripts (for the relational part) and bash scripts (for the iterative and conditional control-flow commands which are absent in PIG). The problem to be solved is the following: given a pair of authors and an undirected graph $G = (V, E)$ where V is the set of authors, E is the *labeled* collaboration relation ($(author_1, author_2, article) \in E \Leftrightarrow author_1$ and $author_2$ have written the article *article* together), we want to find the list of edge labels (articles) associated to the shortest path connecting the two input authors.

In local mode, the command

```
pig/find_path.sh "author1" "author2"
```

will execute iteratively all the PIG scripts needed to find the minimum collaboration path between the two authors. Firstly this script prepares the two files *data/extremities/extremities*, which contains the following line:

`author1 author2`

and *data/transitive_closure/transitive* , which contains the following line:

`author1 author1`

At each step i the database in *data/transitive_closure/* will contain records of the type

`author_j author_k article_1,article_2,...,article_i`

where $article_1, article_2, ..., article_i$ is the list of articles associated to a path connecting $author_j$ and $author_k$. This database at the end of each step will be compared with the database in *data/extremities/* searching for matches; the algorithm will stop when there will exist a record in *data/transitive_closure/* having as authors the extremities in *data/extremities/*.

After having prepared these two files, the bash script executes the PIG script

`pig/scripts/collab_relation.pig`

which computes the collaboration relation having as records lines of the type

`author_j author_k article`

(i.e. the labeled edges in E) and stores it in *data/collab_relation/*. At each iterative step the database in *data/transitive_closure/* is updated extending the paths with the collaboration relation in *data/collab_relation/* with the PIG script

`pig/scripts/transitive_closure.pig`

The matches between the paths in *data/transitive_closure/* and the extremities in *data/extremities/* are stored in *data/path/*, which is the output of the process.

All the relevant operations are executed with the only use of relational operators. The presence of joins makes some operations extremely memory and time consuming, and the intermediate data (especially *data/transitive_closure/*) can easily reach the size of several GB even with the database contaning only 18K articles. For this reason the tests whose results are shown in section 7 have been conducted on the smallest database containing 169 articles.

5.5 Trending of topics over the years

The topic trending calculator processes the articles database and calculates the trending per year of a given set of keywords. The main idea is to extract for each article all the words contained in the title and in the (truncated) abstract, to sort them using the year of publication and to calculate the trending of the set of keywords given as input. The trendiness χ^2_y for the year y is calculated using the chi-square value

$$\chi^2_y = \sum_{w \in Keywords} \begin{cases} \frac{(Obs_y(w) - Obs_{y-1}(w))^2}{Obs_{y-1}(w)+1} & if\ Obs_y(w) > Obs_{y-1}(w) \\ 0 & otherwise \end{cases}$$

where $Keywords$ is the set of keywords in input and $Obs_y(w)$ is the number of words w counted in articles published in the year y. In the indicator the counter of the words not in $Keywords$ has been omitted from the sum since the number of words is extremely variable

from year to year and thus the inclusion of this term biases heavily the final result. This indicator uses as expected value of the distribution the counters at the previous year. A positive value for the indicator means that there have been a positive trending of some of the keywords in that year, while a 0 value means that all the keywords have been used less than in the previous year. The 1-smoothing at the denominator prevents a division by 0 in the cases where $Obs_{y-1}(w) = 0$.

The calculator has been implemented with a single map-reduce job. The command to be executed to calculate the trending of a set of words is

```
hadoop jar SDBM.jar db.ScholarDBManager -topic_trending keywords DB output
```

where *keywords* is a comma-separated list of keywords, *DB* is the path of the articles-database and *output* is the path of the output. All the words are divided in K categories, where K is the number of keywords: one for each keyword and one for the remaining words. Each mapper processes one article and counts the number of words falling in each category, then emits two Key-values pairs:

$$< y, Obs_y(w_1) \quad Obs_y(w_2) \quad ... \quad Obs_y(w_K) \quad 0 \quad 0 \quad ... \quad 0 >$$

$$< y+1, 0 \quad 0 \quad ... \quad 0 \quad Obs_y(w_1) \quad Obs_y(w_2) \quad ... \quad Obs_y(w_K) >$$

where y is the publication year of the article, $Obs_y(w_1), ..., Obs_y(w_K)$ is the number of words falling respectively in the K categories and in each line there are K zeroes. The meaning of each key-value pair is the following: the first K values are the counters relative to the current year, while the next K counters refer to the words in the previous year. Each reducer receives a set of key-value pairs having the same key y and sums up all the counters, obtaining the key-value pair

$$< y, Obs_y(w_1) \quad Obs_y(w_2) \quad ... \quad Obs_y(w_K) \quad Obs_{y-1}(w_1) \quad Obs_{y-1}(w_2) \quad ... \quad Obs_{y-1}(w_K) >$$

which contains the counters of all the categories of words for the year y (observed values) and the year $y-1$ (expected values). At this point the reducer computes the chi-square value as described above and emits the key-value pair

$$< y, \chi^2_y >$$

These key-value pairs can be used directly to print charts showing the trending of an argument (represented by a set of keywords) over the years.

6 Software architecting and coding

The Map-Reduce code is divided in 4 packages: *db,mapred,utils,web*. Package *db* contains the main class *ScholarDBManager* which is used to execute and configure the Map-Reduce jobs and to manage the command-line interface with the user. Package *mapred* contains the three Map-Reduce classes developed for this project: *DBCreator*, *AuthorsListCreator* and *TopicTrending*. The first of these tree classes uses the facilities implemented in the remaining two packages *utils* and *web*: the only classes added with respect of the description in section 3.3 are *ProxiesVector* and *Proxy* in package *web*, and are used to manage the list of proxies (load from file, random permutation, get next proxy). All these classes can be found in the project directory *ScholarDBManager/*.

Two PIG scripts have been written for the collaboration distance calculator described in 5.4: *pig/scripts/collab_relation.pig* and *pig/scripts/transitive_closure.pig*. These scripts respectively compute the collaboration distance between the authors in the articles database and extend the paths from the source in a breath-first-search fashion. The two scripts can be automatically executed in local mode using the bash script *pig/find_path.sh*. The other bash scripts in *pig/* simply execute automatically the PIG scripts in *pig/scripts/* in local mode. The folder *pig/UDFs/* contains the sources and executables of three implemented UDFs gathered under the package *myudfs*. *myudfs.MERGE* takes in input two articles paths with the corresponding sources and destinations, where the destination of the first equals the source of the second:

```
author_1 author_2 list_of_articles_1
author_2 author_3 list_of_articles_2
```

and merges them, returning the tuple:

```
author_1 author_3 list_of_articles_1,list_of_articles_2
```

This UDF is used to extend the paths in *pig/scripts/transitive_closure.pig*. The UDF *myudfs.FILTERPATH* simply removes the first two elements from the input tuple and it is used in *pig/scripts/transitive_closure.pig* to remove the duplicate sources and destinations from the output of the script. Finally, *myudfs.COMMABAG* takes in input a comma-separated list of authors and converts it in a bag containing one author (in a tuple) per line; this UDF is used in *pig/scripts/collab_relation.pig* to generate all the possible authors pairs for each article, building in this way the collaboration relation.

7 Tests and results

In this section all the test performed and the corresponding obtained results will be described. According to the 3 parts of this projects, the section is divided in 3 subsections: database creation, collaboration distance calculator and topic trending calculator.

7.1 Database creation

After numerous parameter tuning and testing the database creator described in section 5.2 has been successfully used to build a database containing the information about more than 1.7M of articles. The greatest difficulty in building such a database is to adopt a good proxy rotating algorithm and to use a proxy list as big as possible. Before to obtain the biggest database, other 2 small databases and the corresponding authors lists have been obtained. Table 7.1 shows some characteristics of the gathered data. The mentioned files can be found in *data/all_computed_data*. An interesting fact that can be deduced from this

DB file name	#articles	#authors	authors file name	size
169Articles	169	150	150Authors	63.7 KB
18KArticles	18301	25203	25KAuthors	6.5MB
1.7MArticles	1701328	1037013	1Mauthors	622.5MB

Table 1: Informations about the gathered data.

table is that the collaboration graph encoded in each database is not dense, as it has $O(n)$

edges (articles), where n is the number of authors mentioned in the database. This is due to the fact that each of the computed graphs does not contain the list of all the articles for the listed authors; the list of articles is complete only for the previous list of authors (for example, *1.7MArticles* contains *all* the articles written by the authors in *25KAuthors*, and is not complete with respect to the authors in *1Mauthors*). As said before, the construction of the biggest database required 7 hours of global computation time on Amazon's servers "M2 high memory double extra large" (high I/O and CPU performances) with 300 threads per Mapper and using 3 nodes: one master and 2 slaves. Increasing the number of threads and/or nodes is a risky operation since the frequency with which each proxy is used is increased and thus is more likely that GS detects the scraping activity. The only way to increase the download speed is to use a bigger list of good proxies (i.e. not already blacklisted by Google). These benchmarks can be used to estimate the computation time needed to build the next step of the database, i.e. using as input the list of 1M authors. Since the software required 7 hours to gather the information regarding 25000 authors, the time needed to build the next database can be estimated to be approximately of 280 hours.

The built database can be compared in size with DBLP[6], a database containing informations about more than 2.1M articles in XML format. Differently from DBLP, the built database reports for each article also a part of the abstract (useful for keywords-guided searches) and the number of citations to the article. The disadvantages of the built database are that for some articles some fields can be missing or wrong (sometimes the results of GS are not well-formed) and, as pointed out in [5], that homonyms authors (or authors with similar names) cannot be distinguished and thus some authors are attributed wrong articles. Anyway, the developed software demonstrated to be a valid tool to gather interesting and useful information that can be compared in size and content with existing databases.

7.2 Collaboration distance calculator

The implemented collaboration distance calculator discussed in section 5.4 has been tested on some random instances (pair of authors). During the tests it turned out that, since heavy join operations are used, the implemented calculator is extremely memory and time consuming. The calculation of the collaboration relation is an example of this kind of bottleneck: since for each article and each pair of co-authors there will be an entry in the collaboration relation, the size of this intermediate data can be extremely large (several GB even if small article databases are used). The following lines show some output results on the 169Articles database with random authors pairs:

```
Authors:
  R Shrivastava, T Jesse
Path:
1) (Distinct size distribution of endogenous siRNAs in maize: Evidence from
  deep sequencing in the mop1-1 mutant,E De Paoli,M Pillay,R Shrivastava)
2) (Toward the conifer genome sequence,E De Paoli,M Morgante)
3) (Automated fingerprint background removal: FPB,A Policriti,M Morgante,
  S Scalabrin)
4) (Physical mapping in highly heterozygous genomes: a physical contig map
  of the Pinot Noir grapevine cultivar,A Policriti,C Segala,G Valle,T Jesse)
```

```
Authors:
  C Taviani,J Zhai
Path:
1) (Synergies between Engineering Solutions in QFD Analysis,C Taviani,F Vezzi,
  G Fantoni,G Santoni)
2) (ERNE-BS5: aligning BS-treated sequences by multiple hits on a 5-letters
  alphabet,A Policriti,C Del Fabbro,E De Paoli,F Vezzi,N Prezza)
3) (Massive analysis of rice small RNAs: mechanistic implications of regulated
  microRNAs and variants for differential target RNA cleavage,E De Paoli,J Zhai,
  S Park,SGR Gurazada)
```

The distance calculator finds all the possible paths of minimum length connecting the two authors; in the previous examples only one path is showed for brevity. Since the authors database is not complete, the distance given as result is always an upper bound to the real collaboration distance between the authors.

7.3 Topic trending

The topic trending calculator has been tested on four topics, each one represented by a set of 7 keywords:

1. **Artificial intelligence**. Keywords: AI, intelligence, artificial, NN, neuron, network, perceptron.
2. **Algorithms**. Keywords: algorithm, algorithms, turing, machine, church, programming, instruction.
3. **Cloud computing**. Keywords: cloud, hadoop, mapreduce, map-reduce, PIG, amazon, EC2.
4. **Databases**. Keywords: database, sql, relational, algebra, data, storage, dbms.
5. **Bioinformatics**. Keywords: dna, alignment, sequence, bioinformatics, bioinformatic, genome, algorithms.

After having calculated the trendiness for each topic on the biggest database (1.7MArticles), the values in the year interval 1950-2012 have been extracted (the calculator provided results back to 1600, but as expected the trends values are trascurable before 1950) and a bar plot has been produced with the statistical tool R. For a better visualization the plot has been splitted in the four images 1, 2, 3, 4. The figures show the chi-square values grouped by year for a fast comparison of the trendiness between the four categories. An expected result obtained is that all the categories have experienced a growth of trendiness across the considered years, with biggest values in the recent years. There doesn't seem to be a correlation between the trendiness of the four categories across the years; this could be explained by the fact that the calculator merely counts the number of appearance of the words in a separate way and doesn't consider possible correlation between them (for example the fact that the words appear in the same context or in the same article). The trendiness calculator has demonstrated to be extremely fast: even using only one node and the *1.7MArticles* database, the process takes only one minute to complete on Amazon's EC2 "T1 Micro" instances.

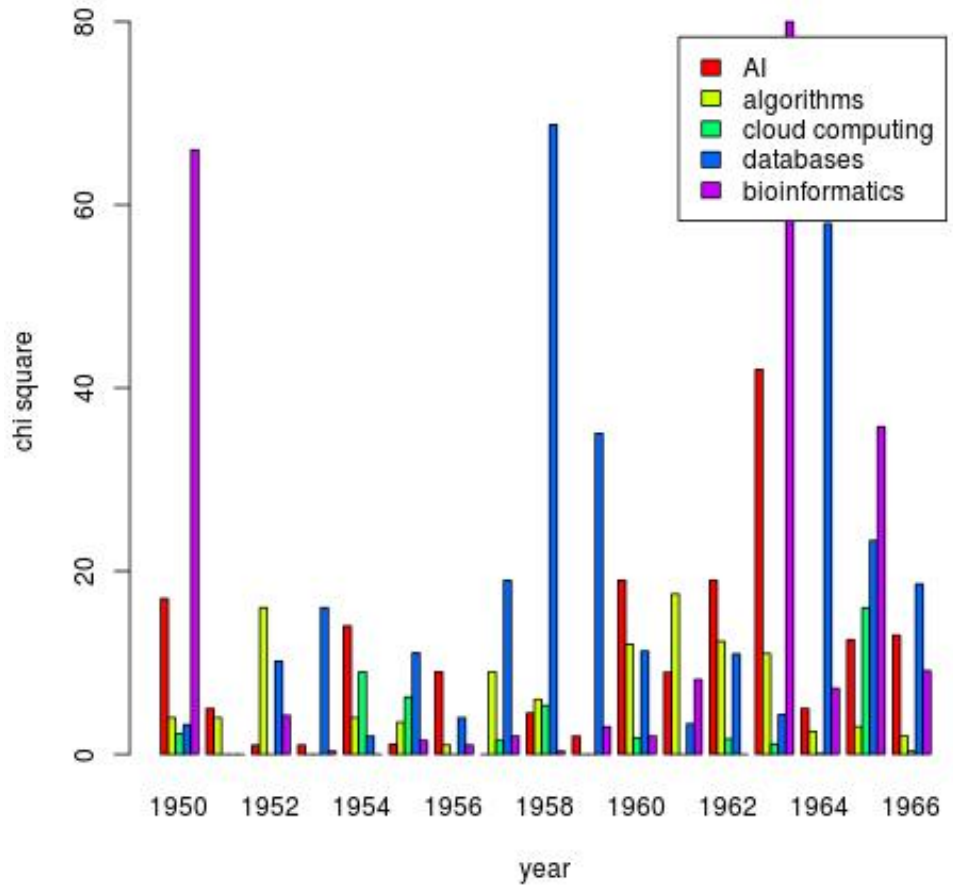

Figure 1: Trendiness of the four topics in the period 1950-1966

8 Conclusions

The work done in this project has permitted to exploit Amazon's cloud computing facilities to build a big database containing information about more than 1.7 millions of articles. The data has been gathered using as source the web search engine Google Scholar[2] and organized in a database in such a way that it can be analyzed with complex relational queries otherwise impossible to answer with the mere use of GS. Even if the built database doesn't contain all the articles indexed by GS, it can be compared in size with existing and well-known articles databases such[6]. For this project two instruments have been developed to analyze the built database: a relational collaboration distance calculator and a trendiness calculator. The first instrument turned out to be extremely memory and time consuming since heavy join operations are used on a huge amount of data. Although there are many ways to improve the PIG scripts developed for this purpose, the most efficient way to develop such a calculator in a distributed way probably would be implementing a Map-Reduce version of Dijkstra's Algorithm. The implemented trendiness calculator, on the other hand, turned out to be extremely fast even on the biggest built database. With this tool the database has been analyzed to compute the trendiness of four computer

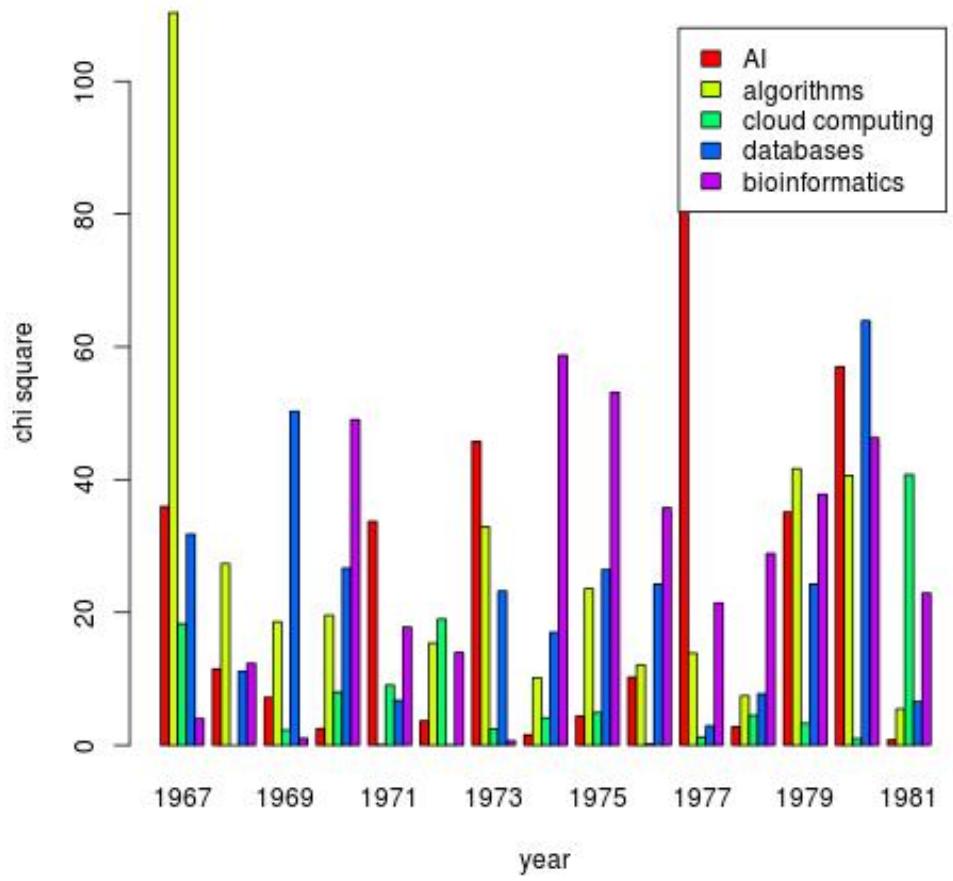

Figure 2: Trendiness of the four topics in the period 1967-1981

science topics using as "guide" a set of 7 keywords for each topic. The results have been displayed in a plot showing for each year the trendiness of each topic. This kind of analysis helps to highlight the relative growth of importance of an argument over the years; it worth noticing that with the mere use of GS such an analysis is impossible to conduct for the lack of instruments offered by the search engine and for the impossibility to sistematically download from it huge amounts of organized data ready to analyze.

References

[1] Erdös number calculator. Website: http://www.ams.org/mathscinet/collaborationDistance.html.

[2] Google Scholar search engine. website = http://scholar.google.com/.

[3] The Erdös Number Project. Website: http://www.oakland.edu/enp/trivia/.

[4] A. Harzing. Publish or Perish, available from http://www.harzing.com/pop.htm, 2007.

[5] A.-W. Harzing. Google Scholar - a new data source for citation analysis. Dec. 2008.

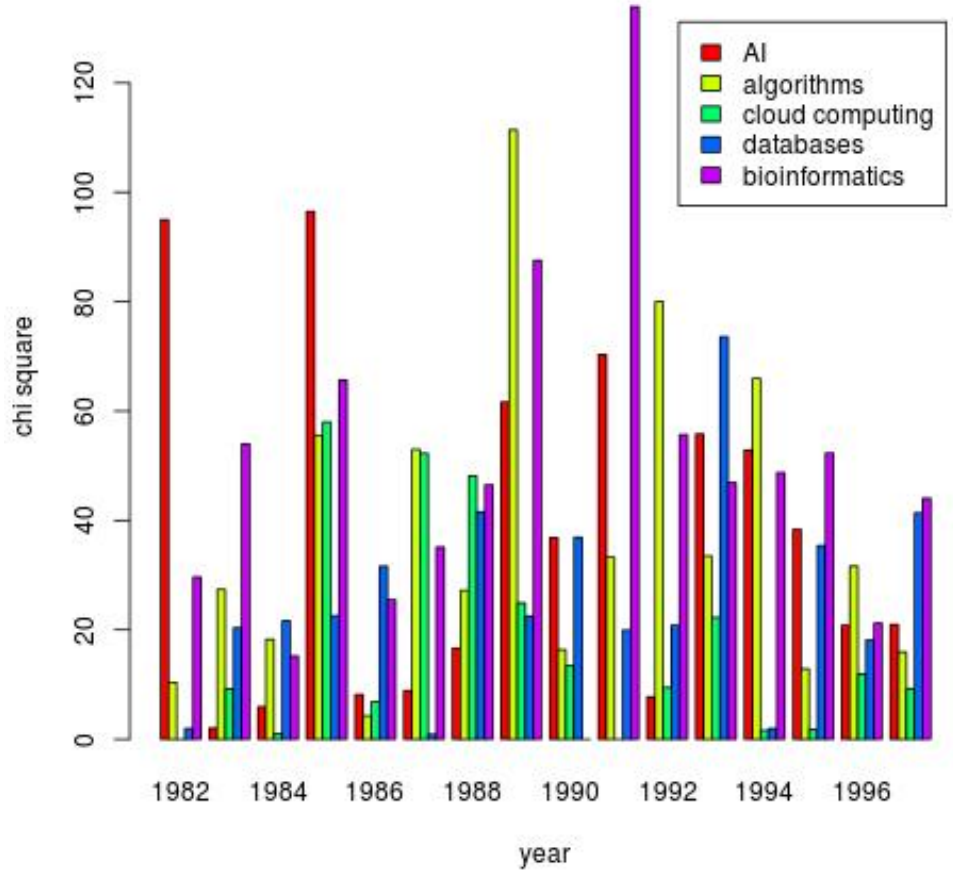

Figure 3: Trendiness of the four topics in the period 1982-1997

[6] M. Ley. DBLP — Some Lessons Learned. 2009.

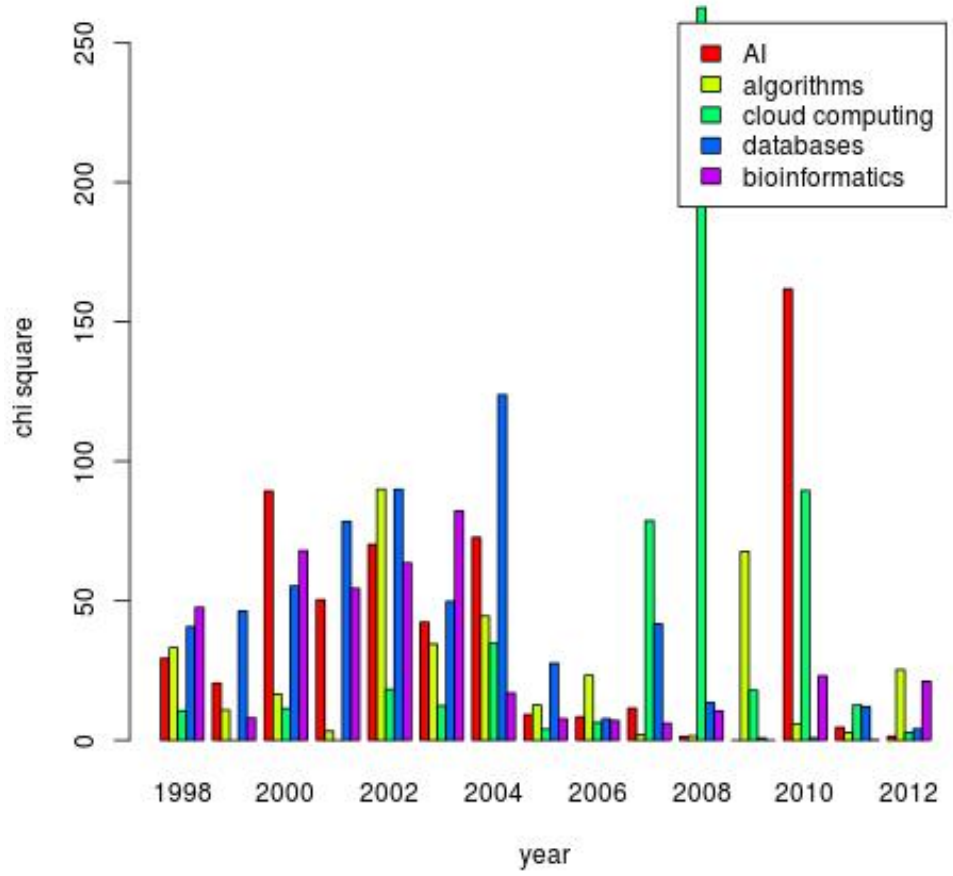

Figure 4: Trendiness of the four topics in the period 1998-2012