Design of Internet Services - Project: Phase #01

Daniela Vianna, Vasisht Gopalan, Nileema Shingte, Shankar Ram, Ramachand J

October 9, 2011

1 Introduction

This document presents our first experience using Nutch and Hadoop. We will briefly describe our experiments and results, challenges faced—mainly during the configuration of Nutch—, and plans for future work.

2 Basic statistics and output format

Let us first present concisely the output format obtained when running Nutch, which provides a command to generate basic statistics from a crawling database. In other words, we will define each field returned by a run of `bin/nutch readdb crawldb -stats`.

Output:

- total URLs: number of URLs obtained after crawling.
- retries: the maximum number of times a URL that has encountered recoverable errors is generated for fetch (by default, the number of retries is equal to 3).
- min score.
- avg score.
- max score.
- db_unfetched: indicates the number of unfetched URLs. All new URLs are classified with an "unfetched" state. The same happens with redirected URLs that are not immediately fetched and with URLs in time to be refetched.
- db_fetched: number of fetched URLs.
- db_gone: number of URLs with the HTTP 404 (not found) status.
- db_redir_temp: number of URLs with the HTTP 307 (temporary redirect) status.

- db_redir_perm: number of URLs with the HTTP 301 (moved permanently) status.

Fetched URLs will not be available for refetching until after a pre-defined fetch interval. After a fetch interval is completed, the URL is marked as unfetched and is a candidate to be refetched. The new fetch date for each URL can be found in the output generated from a call to the `readdb` command on the crawldb database.

Besides the fetch date, using the `readdb -dump` command on a crawldb database can return a list of URLs that were fetched, unfetched, gone or redirected. Based on that, we can write a parse program to find meaningful unfetched URLs that can be inserted/updated into the crawldb, and so will be candidates to be fetched in a next round of crawling. The URLs parsed will be chosen or not depending on the generator parameters, e.g. `topN`, which guarantees that among all unfetched URLs, only N URLs with the highest scores will be selected for fetching.

A set of commands are available to analyze the crawling output. Besides the two commands presented above, we used `readlinkdb` to get the link database content, and `readdb -topN` to get the top URLs in our crawldb database. Additionally, we have a command to print statistics of a specific URL.

3 Seeds

The seeds used for education ware the ones provided by the TA. For finance, we generate a file with 16 different URLs. These are listed below:

```
http://money.cnn.com/magazines/fortune/
http://www.economist.com/
http://www.nyse.com/attachment/amex_landing.htm
http://www.nasdaq.com/
http://www.bloomberg.com/
http://www.kiplinger.com/
http://www.thisismoney.co.uk/money/index.html
http://www.thestreet.com/
http://www.smartmoney.com/
http://seekingalpha.com/
http://www.forbes.com/fdc/welcome_mjx.shtml
http://www.mymoney.gov/
http://www.investorwords.com/
http://online.barrons.com/home-page
http://www.afajof.org/
http://www.ssrn.com/
```

4 Experiments

4.1 Topic: educational domain

4.1.1 Crawling for two rounds with a small URL sample

For a first experiment we chose to run only two rounds of `generate`, `fetch`, `parse`, and `updatedb` with option `topN` set to 10; it means that only the top 10 URLs generated in the first round can be part of the fetch list for the second round. Our choice was based on our wish to understand each crawling step, which is easier with a small sample of URLs. This was the best way to verify if our configurations were correct and the output was according with our expectations.

Observing the statistics generated for the crawl database, for a total of 640 URLs generated, all of them in a first attempt (retry = 0), 620 URLs were classified as unfetched and 18 URLs were fetched during the two rounds; the other 2 URLs were classified as db_redir_temp. The small amount of fetched URLs is related to the value attributed to the `topN` option, which made possible to add at most 10 URLs into the fetch list.

Another important point to be considered is that, for this experiment, we restricted the search to the domains provided by the initial seed. This was made possible by modifying the file `nutch-site.xml` with:

```
<property>
        <name>db.ignore.external.links</name>
        <value>true</value>
        <description>
        If true, outlinks leading from a page to external hosts
        will be ignored. This is an effective way to limit the
        crawl to include only initially injected hosts, without
        creating complex URLFilters.
        </description>
</property>
```

The output from the crawl database allowed us to verify that all URLs classified as unfetched are immediately available to be re-fetched, while the ones classified as fetched, have their fetch time updated to 30 days. This happens to give a chance for the unfetched URLs to be reselected to be part of the next fetch lists.

Path for the crawl database: `/user/dvianna/crawldbD2TN10`, with options `depth 2, topN 10, filtering`.

4.2 Filtered vs unfiltered executions

For this experiment we ran two separate executions of `nutch`. Both executions ran for 5 rounds (`-depth 5`) selecting 1000 URLs with the highest score to be fetched in the next round (`-topM 1000`). The difference between them is in the

filter: for the first execution, we kept the filter off, whereas for the second, we set the filter to ignore outlinks for URLs external to the specified domains. We could consider the applied filter a little extreme; however, our intention was to inspect the impact of any kind of filter applied to each round of our crawling. We could change the file `regex-urlfilter.txt` and use the option `-filter` to apply a more relaxed filter.

Comparing the statistics generated for both executions, we can clearly see the impact caused by the filter. In the execution without the filter, the total number of URLs were 46465 against 21903 from the execution with the filter. The number of fetched URLs was almost the same, as we expected, since the fetch list was limited by the parameter `-topN 1000`. Considering the huge difference between the total number of URLs in those two executions, we can only conclude that the number of unfetched URLs was extremely high in the execution without the filter. This made us believe that this execution should be the one taking the longest time to finish, but this theory proved incorrect: the execution with the filter activated was, in average, twice as slow. It is, however, hard to make any conclusions since we were sharing the cluster with a considerable amount of students, though we still feel that the filter can be the one responsible for the longest execution. We know from online forums that the addition of regular expressions in `regex-urlfilter.txt` can slow down the execution considerably; however, we did not verify this affirmation.

The size of the database is proportional to the total number of URLs for each execution. It means that the database generated by the execution without the filter presents a database with twice the size of the database generated by the execution applying the filter.

Path for the crawldb databases used above:

- `/user/dvianna/crawldbA5` for options depth 5, topN 1000, no filter; and
- `/user/dvianna/crawldbD5F2` for options depth 5, topN 1000, filtering.

4.3 Increasing the number of rounds

In this analysis, we present three different executions of `nutch`: for all of them, no filter was applied, and we define `-topN 1000`. The difference between the 3 executions is the depth (number of rounds): in the first, only one round of crawling was executed; in the second, 5 rounds; and in the third, 10 rounds.

Figure 4.3 presents the statistics for each executions. The x-axis defines the number of rounds and the y-axis defines the number of URLs. The output from the commands `readdb -stat` and `readb -dump` were as expected. As the number of rounds increased, the total number of links increased proportionally to the number of rounds. The same happens with the number of fetched and unfetched pages. The size of the database also increased proportionally.

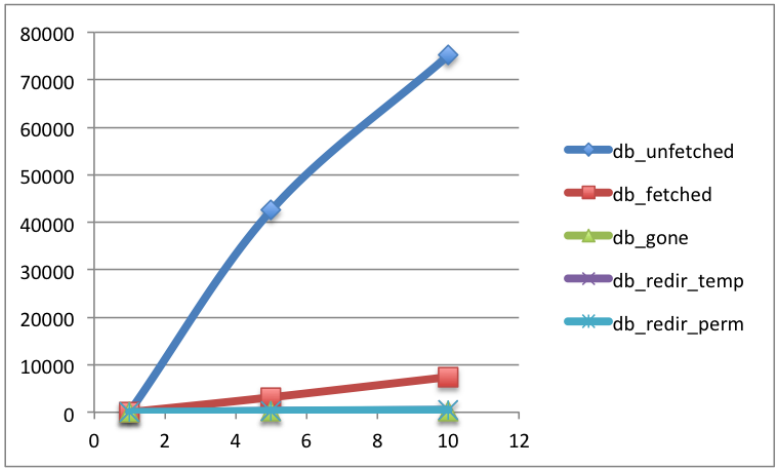

Figure 1: Statistics for executions with depth 1, 5 and 10

Path for the crawlbd databases used above:

- `/user/dvianna/crawldb` - depth 1, no filter (mistakenly deleted from the file system)
- `/user/dvianna/crawldbA5` - depth 5, topN 1000, no filter
- `/user/dvianna/crawldbA10` - depth 10, topN 1000, no filter

4.4 Trying something big

For this experiment we decided to run something big compared to all our previous executions. We then ran 20 rounds of `generate`, `fetch`, `parse`, and `updatedb` without any filter and with `-topN 10000`. The execution took almost one entire day; however, we cannot make any conclusion for the running time, since we did not know the load of the cluster during the execution. Only to give an idea about the data generated, we have a total of 1,189,046 URLs, most of them retrieved in the first attempt (only 4,707 retried in the second attempt). From this total, only 150,689 URLs were fetched and 11,436 were marked as gone.

The size of the crawldb database is 122.65 MB, compared with the 4.39 MB generated by an execution with 5 rounds and `-topN 1000`, and with the 8.15 MB generated by an execution with 10 rounds and `-topN 1000`. Neither of these executions applied any type of filter. The huge difference between them is not related exclusively to the number of rounds, but to the value of `-topN`, which defines the number of URLs in the fetch list.

Path for the crawl database used above: `/user/dvianna/crawldbA20` - depth 20, topN 10000, no filter.

4.5 Topic: financial domain

4.6 Educational vs Financial domain

For this experiment we ran the Nutch crawling with `depth 5` and `topN 1000`. Comparing the results from the financial seeds with the ones from educational seeds, we concluded that even though the first one has generated more urls, the number of fetched urls are almost the same for both of them. This result was expected since the size of the fetch list was the same for the two executions.

4.7 Filters and Plugins

For the financial domain we concentrated mostly in adding filters and plugins during the crawling steps. However, we could not get the results we expected from them. We have to learn more about plugins before we can use it in our executions.

5 Scores

We have spent some time trying to understand the Nutch algorithm to compute the score for each link, since it plays an important role when selecting the top N URLs for the next fetch step.

Below is a brief description of the Scoring algorithm:

```
1. Count the total number of links.
2. Initialize rankOneScore as 1/numberOfLinksInWebgraph
3. Initialize score of all links with rankOneScore
4. Create an inverse mapping of the links so that you know
   the incoming links (similar to linkdb)
5. Compute the aggregate score of all the incoming links.
   A page with zero inlinks has a score of rankOne
6. Calculate the linkRank score using: linkRankScore =
   (1-this.dampingFactor) + (this.dampingFactor*totalInlinkScore)
7. These scores are stored in a NodeDb, and used for the next
   iteration of score calculation.
8. The above steps are iterated over a user specified number of
   times. Multiple iterations allow the linkRankScores to converge
```

Observations:

- the scoring algorithm ignores duplicate links and pages
- damping factor is a configurable value, defaults to 0.85. Why is this value selected? Is it based on some heuristics? What does it indicate?
- scoring is independent of query, it is purely based on the graph structure of the fetched links.

- crawl strategy is breadth first
- the score is used by the generator to select the top N links to fetch

6 Discussion

6.1 Output format

The output format was discussed together with the experiments analysis.

7 Lessons learned and challenges faced

We had a hard time playing with the configuration files, mainly because most of the time we could not see clear the result for the changes we were applying.

Another problem we faced was that, after getting so many data from a variety of executions, we did not know how was the best approach to analyze our data in order to draw interesting conclusions from it.

We learned:

- how to play with the configuration files to get different execution patterns from our crawler;
- what behavior and results to expect from each execution;
- how to deal with the output generated in each round;
- how flexible `nutch` is.

Reading on online forums about the huge number of unfetched URLs obtained for each execution, we found an interesting discussion related to the number of map and reduce tasks. Some affirmed that increasing the number of map and mainly reduce tasks could reduce the number of unfetched pages. However, we did not have a chance to look at this matter.

Our script was prepared to accept different values for the option `-threads` used by `fetch`. However, we did not test it yet. But this is something that we are planning to do soon.

7.1 Future work

There is much to be done. We are planning to play with threads, map/reduce tasks, implement different filters, maybe implement our own parser. We did not go further about the scores and index part—we have to learn how this is done and see what we can do to improve it.