Building an index for PubMed data

Introduction

PubMed is an important biomedical literature database produced by the National Library of Medicine (NLM) in Bethesda, Maryland, USA. It contains over 25 millon citations. In this paper, I discuss ways to build a local copy of the data. I have three requirements on my local copy.

First, data on a the local machine should by as current as possible. It should guarantee that it is up-to-date for at least the previous day.

Second, it should be fast to retrieve records by identifier.

Third, the collection efforts should give me a trace on when the record was available locally on my system.

PubMed do provide an API. The API is reliable and fast. However the API requires a search term. A term that would retrieve all data entries does not appear to be available, from my reading of the API documentation. Reader who are aware of the API may dismiss my problem as trivial.
There should be a way to query records every day to get only those

Vendor data

Individuals or organizations can apply to become PubMed vendors. The term is a bit misleading as NLM don't expect vendors to actually sell the data. Candidate vendors have to enter a license agreement. They have to explain the aims they access the data for. Once NLM have agreed to give access to the data, vendors have access to PubMed data. Technically, this means that the vendor can specify five ip addresses. These addresses have access to an ftp server. There are two separate directories. Both directories contain compressed XML files. The first directory has the “baseline” files. The second directory has the “patch” files. The baseline files are updated once a year in mid-December. At that time, a set of baseline files is replaced by another. To distinguish both sets, the files are name medline*yy*n*xxxx*.xml.gz. Here *yy* is a two-digit indicator of the year and *xxxx* is a four-digit sequential number. These files remain unchanged during the year. No new files are added to the baseline directory. In the course of the year, year NLM keeps adding “path” files in the patch. The sequential number starts where the patch file numbers have ended. Usually, NLM add a file every day at 12:00 local time, or, if the day is a Sunday, at 6:30 local time. Sometimes there are two files, sometimes none, but usually there is one. Patch files contain additions of new records as well as corrections to records found in the baseline files or in other patch files distributed earlier. Once a patch file is released it is not changed. Patch files don't contain format changes. They don't contain indications of deleted records. These two data events is what the baseline files are for. Licences are required to reload the baseline data every year to flush out invalid records.

This organization is ingenious. NML have been using it in production for many years. Records within the file start of differernt lines. Thus is it possible to avoid XML parsing for faster record processing. Let me refer to the union of both set of file as the “meci” files.

Since meci files don't change, incremental processing is easy. First, I uncompress each file. Each file contains XML record. Each record has a unique identifier known as a PubMed id, pmid in the

following. For each uncompressed medline*yy*n*xxxx*.xml meci file, I can create an index file medline*yy*n*xxxx*.xml.idx. The index file contains line each describing a record in the meci file. Thus, for the first file in the 2016 dataset

```
$ head -2 medline16n0001.idx
16691646 260 1496
16691647 1756 1377
```

This means that the file medline16n0001 contains the record wit pmid 16691646 starting at byte 260 with a length of 1496 bytes. This is the first record in the file, then comes the second record, on the second line. Thus if we know it what meci file a record is, we can look for the pmid of the record in the first column of the corresponding index file. To build a fast lookup, I need to index file the correct meci file for a pmid. To do this, I split the set of all pmids into 1000 parts of roughly equal size. I do this by simply takeing the last three digits of the pmid. When the pmid is less then 1000, I add zeros to the left. The result is what I call the pemi associated with the pmid. Then in a first step, I write each line of the index file into “pmidex file”

To do that, I build a set of 1000 files label them 000 to 999. For each pmid, I take last three digits. We can find the first record in the the 2016 baseline files at in pemi 646

pubmed@elneg:~/data/barf$ head -2 646.raw
16n0001 16691646 260 1496
16n0001 17231646 2187190 1283

Each line contain the meci file name, the pmid, the offset and the length in that order.

The pmidex file is just an intermediairy state to look up records by pmid. There are two issues. First, recall that subsequent meci files will contain records that overwrite earlier ones. Thus we are likely to see the same pmid repeated. Thus we have to scan the entire form. We can make the search shorter in three ways. First we can write the pmid at the start of every line. This will allow faster access to a line oriented tool. Second, we can concatenate all the record locations, meaning the triplse [meci, offset, length] pertaining to a fixed pmid after that pmid. Finally, we can sort the file by pmid. The result is a sorted index file I will call the “sortdex” file. There are a thousand sorted files, one for each pemi.
To update the data, we can incrementally process every meci file on the day that it arrives. I can decompress it. I can build the append build an index for it and conca

Let us summarise what happens when we have to find the record associated with a pmid. First, we find the pemi of the pmid. Then we used lookup -b on the sorted file to find the line that start with the pmid. We split the line to find the “offset length meci” combination with the latest meci. We go to the uncompressed meci file to find the record. Typically this will take less then 0.2 secords.

“lookup” is a standart unix utility that will look for the pmid at the start of each line. It's “-b” flag implements binary search. This makes for a fast lookup.

So things look fine at this point.

API requests

When I first constructed the index, I asked Ross Mounce to send me pmid of papers he is interested in. For the majority of these pmids, my index would not find a record. A painfully slow grep searches of the pmid revealed that they were not in the meci data. I was aware of the fact that the meci data is not complete. The precise wording of the documentation, at https://www.nlm.nih.gov/databases/journal.html as read on 20 December 2015 is

> NLM now exports over 98% of PubMed records to MEDLINE/PubMed licensees. The approximately 2% of the records not exported to MEDLINE/PubMed licensees are those tagged [PubMed - as supplied by publisher] in PubMed. These are mostly records recently added directly to PubMed via electronic submission from a publisher, most of which are soon to proceed to the next stage, at which time the record will be exported. Most other non-exported records are either: older citations for the relatively few non-MEDLINE journals in PubMed; citations to MEDLINE journals prior to their date of selection for MEDLINE which were submitted electronically by the publishers before late July 2003; or citations electronically submitted for articles that appear on the Web in advance of the journal issue's release (i.e., ahead of print citations). Following publication of the completed issue, the item will be queued for issue level review and released in In-Data-Review status. All "publisher-supplied" citations have not been reviewed for accurate bibliographic data.

The text is confusing at the point were it says “most other non-exported” since this may suggest that some the non-included records are not tagged. I contacted the NLM using te licencee email address and I was told that there the relevant API query would be “publisher [sb] ”, which, when we add the API URL and URI encode yields http://www.ncbi.nlm.nih.gov/pubmed/?term=publisher+%5Bsb%5D. It turns out that running regular requests to the API is necessairy.

API requests

I contacted the hotline for the licencees, and I was told that the query … would “”. While this does not seem to completely cover all the aspects of why a record is not in the meci, it seems a reasonable start. Thus every day, shortly after midnight Bethesda time, I can run a query of the form. Let *yyyy-mm-dd* be a represention of yesterday's date I use the API at http://eutils.ncbi.nlm.nih.gov/entrez/eutils/eftech.fcgi with the parameters db=pubmed, datetype=edat, mindate=*yyyy-mm-dd*, maxdate=*yyyy-mm-dd*, term=publisher+%5Bsb%5D, retmode=xml and retmax=99999. This will return identifiers. The API can then be used to download records for these identifiers. I call these records “fapi” records.

Since 2% of 25000000 is just a mere 500000 records, it is possible to create one directory per pemi, and store each fapi record in a separate files in there. Thus the pmid to pemi lookup can be recycled to find the location of the file that contains the fapi data. Access to these records is even faster that access to meci records. However, there is a problem. I have no renewal strategy for records while they are in a fapi state. At this time, while a record is a fapi, it is not being renewed at all. Thus, if NLM corrects a fapi record, the correction is not reflected in the data on disk. This could only be done by periodically reviewing all the fapi, look up what date they correspond to using the file modification date and then downloading them again. To be really sure we should do this for all fapis every day. That's a bit of a tall order.

Nevertheless, we need to periodically clean up the fapis. That is, at the time we process mecis, I look at all the fapis available. If the pmid of the fapi is found in a meci, the fapi is surplus to requirements. I delete it.

Thus when the meci search, as outlined in the previous section fails, I can look whether the fapi file exists. If it does, I return the record which is simply the contents of the file. As a last restort

Daily inputs

I have a requirement to produce a daily digest of incoming records. A records is incoming if it has a pmid not seen before. There is no requirement for sub-day precision in timing of arrival. For meci files, the timestamp on the file in the ftp server, in local Bethesda time apparently, seems to correspond to the time they are actually available. If I manage to download them the same day, I have an indication of the date in the files' timestamps. For the fapis, even though it is not the day I get them, I impute them on the day that I place in the API search request. After the download of the fapis' early in the US morning, I can combine the fapi harvests with the mecis of the previous day into a daily input. This is the “dain” file. The structure of the dain file is very simple. If the record is present as the fapi, I write a line containing the pmid. If the record is a meci, I put the combination “pmid offset start mecif”. Dain files need regular maintanence. Each time we get a new version of a PubMed record, we need to update the mecif/start/offset triple for the pmid to make sure we use the latest version.

Dain files are are the only parts of the index actually needs preserving over time. There are two reasons for that. First, fapi cleansing implies that when a fapi is lost, the arrival date of the corresponding record would be incorrectly imputed to the meci date. Second, an the baseline update day, all files arriving will have the same meci date. That means that this information would also be lost, unless it could be reconstructed from the data in the actual records. I have not yet investigated how meci records relate to obseved meci file times.

Baseline file updates

Baseline file update times seems to be not known in advance. But NLM send an email to licencees when a new set of files is available. At that point it is useful to mirror them. The meci processing is really the same as for patch files. I store the uncompressed files. Then I index every file. Then I add lines for the new patch files to the pmidex files. When this is done, the pmidex files will contain roughly twice the amount of lines than before. I then proceed to created the sortdex files. When this is done, the sortdex files will have a line length that is roughly twice the previous line length. Thus, it is desirable to do some further processing to reduce disk space. Next we need a general update of the dain files. We need to find the latest meci for each pmid referenced in the dain. If it is not available in the current data we can try to get the record data using the API thus reverting to a fapi record.
For pmidex files, the records that refer a previous year mecif can be removed. For sortex files, we can remove all