with the files fetched and read by a simple BufferedReader on a FileInputStream. While this worked, it was extremely slow. For example, the word "Hello", when searching only for noun senses, would take around 45 seconds, and would fail if the other word types were also searched (due to the one minute request limit). This is due to the fact that to search the database each file had to be fetched and read line by line and processed. The noun index itself has over 100,000 lines, which while not a problem in local testing, is extremely slow on the App Engine (due to the requirement to fetch the file as it is being read).
The next option was to make use of the App Engine's high-replication datastore. This is a database-like feature, hosted on the cloud with the application. It is high-replication in the sense that your data is replicated among a number of data servers, reducing the risk of data loss. It has a very fast access time for application running on the App Engine due to a very clever indexing and caching scheme. The main issue with the datastore is the quotas imposed on free applications. There is a limit of 50,000 write operations a day (with each insert operation requiring 4 writes and 1 read). An attempt to insert the WordNet database as objects (with each line of the index file as an Index object and a similar structure for the data files) hit the quota within second of inserting the first file, which was obviously an issue. To overcome the problem, the datastore bulkuploader function was used. This is a Python application which enables the insertion on csv and xml files straight into the datastore using the remote api. This required a Python instance of the application to be run alongside the full Java version, which enabled the Python bulkloader function to be used to access the live datastore. All of the files were converted into csv files using a simple java program, included as part of this project. All of the four index files were included as one single csv file of a "MasterIndex" type, which enables all of the four types to be retrieved at once. The data files were kept separate, so that they only needed to be searched if they exists in the index. The MasterIndex file has three fields, the word, the offsets of its senses (as a string which is split for processing) and its type (noun, verb etc). The data files have two fields, the offset and the synonyms (also as a string which is split for processing when required).
The basic method of searching for a word in the datastore version of WordNet is to first query the MasterIndex table, which return all the sense offsets for every type of the word. For each word type, if senses exist they are queried in the appropriate data table, using the list of offsets as a parameter and the "IN" filter operator. All of these senses, for all types, are then formed into a "Word" object, which represents all of the senses(and therefore synonyms) for the entered word. These word objects are then used to build the full sense list for the word to be used in the substitution.
The two sets of corpora data were also inserted into the datastore in the same way.