Index Building Performance Improvements

This page discusses performance improvements to the multisite search index building. We hope to consider changes to the code that are candidates to improve the performance, assess the likelihood that they will improve the performance, implement some of them and assess if they did improve the performance.

In general we are likely to think of performance as measured by how long it takes to index all the individuals for some site. Hopefully we can ignore memory space and disk space considerations for now. Another aspect of performance to consider is resources required by the LOD site being indexed. This aspect could be measured by the count of HTTP request to the site, load average on site over the time the index is being built, difference between response time during indexing and not during indexing.

Caching of HTTP Request Results

The run time of index building process is likely to be dominated by time spent requesting linked data via HTTP. In many cases we may need the data for a URI in different times or processes during the run of the index builder. Caching these requests is a standard way to improve performance in situations like this, so much so that it is part of the HTTP standard.

If we create the most naive implementation of the IndexBuilder, we might wind up asking for the same RDF chunk many, many times. Faculty belong to the same department, Co-authors write the same document, etc. This might not be a significant problem. We should try the naive way and see how much duplication there is, as a percentage of the total load.

Convoluted

When we go through the "Discovery" phase, we presumably get unique URIs. We could keep track of them, so we don't fetch them again if they are co-authors on each other's documents, etc. Of course, the "Linked Data Expansion" phase can reveal many additional URIs for which we must fetch data, and we would need to keep track of them also. Ideally, of course, this would work with Hadoop, so if one node fetched RDF for a URIs, no other node would fetch for that same URI. This becomes challenging.

Caching in HttpClient

The HttpClient can be configured to add a caching layer. See the manual page: http://hc.apache.org/httpcomponents-client-ga/tutorial/html/caching.html

If we use the standard memory-based cache, we might reduce the duplication in RDF requests. If a single Hadoop node fetched the RDF for a particular URI, it would not fetch it a second time. This would require no logic in our code, since our code would make the repeated HTTP request, and the HttpClient would simply find the result already in the cache. There are tuning questions here: how many bytes can we devote to caching? How much will the cache overflow? We will also be caching many pages that will not be asked for a second time, but with a memory-based cache, there is very little penalty for that.

Again, each Hadoop node would maintain its own cache, so some duplication would undoubtedly persist.

Taking advantage of Hadoop when Caching with HttpClient

The HttpClient caching layer allows us to simply implement our own storage. We only need a mechanism by which the HTTP response can be stored, using an arbitrary String as a key, and can then be retrieved or deleted by that same key: http://hc.apache.org/httpcomponents-client-ga/httpclient-cache/apidocs/org/apache/http/client/cache/HttpCacheStorage.html

If we implemented the HttpCacheStorage to use the HDFS (Hadoop Distributed File System) as its storage area, our cache could be effective across all Nodes. Further, it could even persist from one run of the program to the next, although we would expect that many or most of the responses would have expired by then.

Efficiencies and Penalties

First thing to determine – are we seeing a significant percentage of duplicated requests? Can we reason about this, or do we need to instrument it?

Memory-based cache:

- How big should the cache be?
  - We can set limits on the number of files to save, and the maximum size of the files.
- What happens when a file that is too large is retrieved? Is it just not saved?
- What happens when we retrieve more files than the cache can hold? Is it FIFO or LIFO or by some other scheme?
- Storing things in memory and retrieving them has no significant overhead. However, it we suck up too much memory, then we may impact the application's performance.

HDFS-based cache:

- There are constant sources of overhead that may or may not be significant
- Each page we store in the cache will be written to disk - perhaps on a different node, requiring a network transfer.
- Each time we want to retrieve a page, we must first check whether it is cached. This also may require looking on a different node.

Layered cache:

The caching layer acts as a decorator on HttpClient, making it easy to combine both a memory-based cache and an HDFS-based cache. When a page is retrieved, it would stored in both caches, which is no improvement. However, when we check to see whether a page is in the cache, we first check in memory. If we find the page there, we look no farther. If we have significant repetition, this could improve efficiency.

Is this worth doing?

The memory-based cache is easy to do. Very easy. The HDFS-based cache is more work.

The big question is, what do we save? If we have very little repetition, we might actually see a loss of performance by introducing a cache, or by using a cache that was improperly tuned.