LexEVS 5.1 Performance Best Practices

This is intended to be a sent of guidelines or 'best practices' for getting the best performance out of LexEVS 5.1

- Using Iterators

With 5.1, ResolvedConceptReferencesIterators are populated with Lucene Document data on demand, meaning no data is pulled from the Lucene Documents until 'next' is called on the iterator. This means the initial resolve of the iterator will be much faster.

- Sorting

Sorting is expensive, because in order to sort, the entire result set needs to be resolved either from the Lucene Documents or the database. The exception to this is sorting by Lucene score (matchToQuery sort). This sort is very fast in all cases.

If sorting is a requirement:

- Consider creating a new Search Plugin that will better match the results to the user input. This way you can rely on the Lucene score to sort by the best results.
- Resolve to a List and use a 'post' sort. This means: First resolve to a list of a set number of results. Then sort -- the 'post' sorts will only sort on the results that are included in the list. For instance, suppose the user searches for 'heart' and these hits are returned from Lucene:

'heart' 'heart attack' 'heart failure' 'bleeding heart'

now, resolve to a list of say, 2 results. resolve -> \'heart','heart attack'\

this essentially gives you the top n results of your query.

Now you can 'post' sort these two results -- that way extra calls to Lucene and/or the database don't have to be made.

- Questions:
  - Would the Search Plugin mentioned above be a client-side plugin, or a server side plugin?
  - To implement pagination properly, the 'post' sort would have to be performed on the entire list of resolved concepts. Would there still be a performance issue?
    - All sorts (except matchToQuery sort) will require either a database lookup or lucene lookup for all the results that are to be sorted. That is why 'post' sort was introduced. If you resolve to a list and limit it, only those results that included in that list will be sorted. This allows you to get the top n results from a query and sort them, without having to sort the entire result set.
    - Sorts are going to be performance bottlenecks if used with large result sets -- Lucene document retrieval is only so fast and there are only so many transactions per second we can push to the database. Even more, if you sort large result sets by anything but the score, the first results displayed to the user may in fact have a very low lucene score.