IndexPartitioning

ABOUT THIS DOCUMENT

This document was created as a brainstorming exercise, and a road map towards possible implementations – as of 2008-10-29 it does not describe any particular features of Solr.

some of the basic ideas expressed here can be achieved by using multiple indexes (either on separate machines, or as separate Solr Cores on the same machine) with the same schema and DistributedSearch over these indexes when you need "cross partition" searching.

Overview of Idea

In addition to the goal of allowing a single Application Server instance (ie: port) to run multiple Solr webapp instances (ie: the same war, duplicated several times with different names, each with their own configs/data) there has been some discussion about the idea of allowing a single Solr webapp to support "partitioning" of the index – either based on a particular field value, or based on an explicit "partition name" used when issuing updates.

Purpose

The motivation for this would be to allow "faster" queries over a subset of the data, when that is the "common case" (ie: if you have a single index containing homogeneous data for multiple clients, and sometimes it's neccessary to query across all of the data at once, but the majority of the cases you only want to query one client's subset of the data)

Detailed Description

solrconfig.xml would have new information indicating there are index partitions, something like this perhaps...

```
    <partitions default="foo">
      <partition>foo</partition>
      <partition>bar</partition>
      <partition>baz</partition>
      <partition>bax</partition>
    </partition>
```

...which indicates that there are 4 partitions, which can be refrenced by name. Each partition would be backed by a physical index on disk in the data directory (the directory name would be the same as the partition name (or maybe the syntax would be `<partition dir="f_o_o">`).

The `<add>` command would support a new `partition="..."` attribute indicating which partition the document(s) should be added to, if no partition is specified, they would go in the default partition. If the index has a uniqueKey field and allowDups is false, then old docs with the same key should be deleted in *all* of the partitions. (this is neccessary to allow data to be moved from one partition to another ... but perhaps a new `overwriteOtherPartitions` attribute on the add would help here?)

Deleting by id should (probably) delete across all partitions (see above) but deleting by query could also be confined to a single partition using a similar `partition="..."` attribute.

`SolrQueryRequestion.getSearcher()` sould continue to behave as it allways has, returning a Searcher acoress the entire "index" – by making an IndexSearcher over a MultiReader of all partitions. but a new method could be added: `SolrQueryRequestion.getSearcher(String partitionName)` which would allow request handlers to confine their searches to a single partition.

Alternate Dynamic Partitioning Idea

Partitions could be created on the fly, based on field values. Considering confguration like this...

```
    <partitionField default="foo">objectType</partitionField>
```

...which would indicate that anytime a document is added, the `objectType` field should be inspected, and used as the name of the parition (and directory of the underlying index). if a document does not have a value for the objectType field, the default partition of "foo" should be assumed.

Things that need to be considered

- SolrQueryRequestBase currently grabs a searcher as soon as it is constructed so that it's garunteed to have a consistent view of the index. would it need to grab a seearcher across every paritition to ensure this without knowing in advance which partition(s) the plugin wants to look at it?
- how do the cache configurations work regarding the various searchers/indexreaders?
    - does each searcher on each partition have it's own caches with the same config as the main searcher?

    - is there a way to specially config the caches on a per partition basis?
- how would the replication scripts need to change?
    - should it be possible to replicate individual partitions seperately?
- The main motivation is performance, based on past experience showing that using partitions with lucene indexes is faster then uber-lucene indexes, but Solr already has a lot of caching improvements that may already make the issue moot – and adding support for this could acctaully harm the performance compared to using a single index (depending on how the change would impact the caching framework). Some serious Performance testing should be done before expending a lot of effort on this idea.

Alternate Dynamic Partitioning Idea (number 2)

Similar to the previous Dynamic Partitioning Idea, this variant provides the constraint that the factors that go into partitioning data are known in advance. Considering confguration like this...

```
<partition>
  <partitionField name="creationYear">
   <value>2004</value>
   <value>2005</value>
   <value>2006</value>
  </partitionField>
</partition>
```

This would indicate that any time a document is added, we inspect the value of the objectType field to determine to which index to write data. The default partition is used whenever a value for objectType is either not provided or does not match one of the provided values. The data directory would follow the convention of partitionFieldName/partitionFieldValue (e.g. data/creationYear/2004/index).

Additional considerations

- Options should be considered for how defaults are handled:
    - Does data exist in exactly one place in the index (i.e. either in the default or a partition)?
    - Can we have an option to always keep a copy of the data in the default partition, possibly with a copy in a partition?

Query Integration

One suggestion is to integrate partition selection into the query parser. For example, assume that we have an index storing data about users, partitioned by the year in which the user was created in the system. If we issue a query like "+email:mbryzek +creationYear:2005", we may want to consider extending the query parser to identify that a query has been issued that can be rewritten to use a partiiton. Doing this provides a simple external API that should be easily understood.