LargeIndexes

An index can be large in several dimensions: number of entries, sizes of fields, number of fields in most records, number of total unique fields, total number of terms in a field across all records.

Number of entries

An index can have hundreds of millions of small records. For example, Flickr has billions of records, but there are not many data fields per entry and the caption and description fields tend to be very short.

Large individual fields.

It is possible to store megabytes of text in one record. These fields are clumsy to work with. By default the number of characters stored is clipped. There are some strategies available.

- Index-only: store the text in a file or database and only index the field in Solr
  - Highlighting is not available
- Break the documents into pages and index each page.
  - The pages will be ranked individually. There is no feature to group the rankings of pages found against a document and create a score per document.
    For example, some legal text corpuses have book-length contracts mixed with 1-paragraph memos.

Large number of fields per record.

Some indexes can have hundreds of fields in every record. Financial contracts can be very complex, and an index could contain hundreds of fields per transaction.

Large number of unique field names.

In the "electronics store" example used in the Solr configuration file examples, each product type may have 5-10 unique metadata items (megapixels or printer paper). But since the entire store can have hundreds of different products, the total metadata name space can be in the thousands. (The wildcard field feature is the right way to handle this.)

Total number of unique terms across all records
Memory use for a facet query uses a counter for every unique term in the index, for every field used. A facet query on a boolean field (or strings "true" and "false") will use almost no RAM, while a facet query on a field with millions of total terms may run out of memory.

Very large numbers of unique terms in an index segment can cause significant memory use and lead to OOMs. Setting the termInfosIndexDivisor can significantly reduce memory used. An alternative that affects both indexing and search memory use is to change the index time setting termIndexInterval to a larger value.

Tips and tricks

Hash Key Identity
See UniqueKey about using a cryptographic hash UUID for the unique key.

Solr-stoppers
SpellCheckComponent and LukeRequestHandler walk the entire term database instead of vectoring term sets through a query. This can make a large index unavailable for hours.

In solrconfig.xml, change the dismax parameter "q.alt" to something besides '*:*'. Hitting this query by accident can also make a large index unavailable for minutes.

Sorting

With many records, sorting on a field is problematic. Memory use varies for different tasks: searching words in the text fields needs X amount of RAM. Sorting on a field takes Y > X RAM, and faceting on a field with many values takes Z > Y > X RAM. In an index with many short records, the user could search but not sort, and sort but not facet.

DistributedSearch
Faceting

DistributedSearch merges facet results (see the page for limitations). *(Please comment on implications for huge facet result sets. It seems like memory usage in merging by count and merging by name have an equal upper bound, but the average case will require very little memory for merging by name.)*

Horizontal v.s. Vertical Partition

In database jargon a "horizontal partition" splits record sets into multiple stores, while a *vertical partition* splits each row into multiple pieces and stores each piece in a separate database, cross-connected by the primary key for the record. DistributedSearch provides a *horizontal partition*. There is no implementation of a vertical partition across indexes.

MultiTiered

If one places too many servers per proxy/shard multiplexer, then the proxy server can become overloaded. Multiple tiers of Solr distributed proxy search servers, scales the per server load out to N tiers. See SOLR-1477

KattaIntegration
Hadoop based RPC enables potentially better scalability compared with HTTP. See SOLR-1395

HadoopIndexing
SOLR-1301 implements Hadoop based indexing for Solr.