

Summa White Paper Technical Section

The Summa Project Group,
The Digital Resources and Web Group,
The State and University Library of Denmark
`summa@statsbiblioteket.dk`
`http://www.statsbiblioteket.dk`

October 4, 2007

The Summa search system can be divided into the presentation layer and the index and storage layer. The index and storage layer imports data from various data sources and provides a number of web services as illustrated in figure 1. The main Summa web services available to the presentation layer are

- Search index,
- Fetch full record from storage,
- Calculate facet browser result.

Summa also offers did-you-mean and suggest web services, and the Summa website module further makes use of a number of external web services.

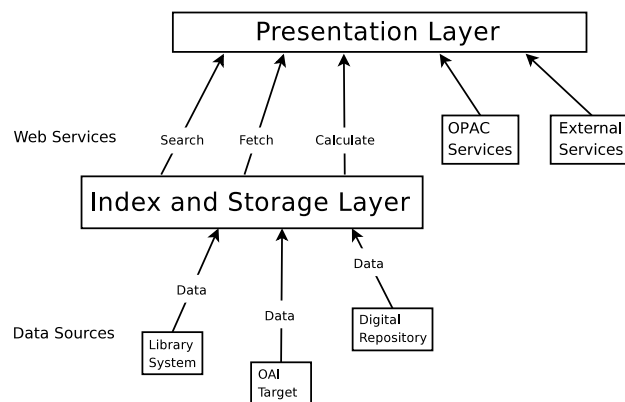


Figure 1: The Summa architecture as a black box.

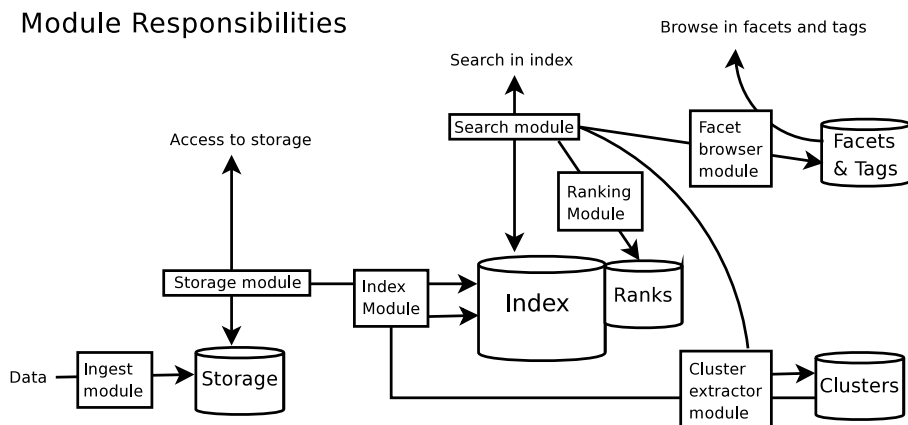


Figure 2: Summa data-flow and module responsibilities.

The index and storage layer provides the Summa web services through the central search engine, the central storage access and the central facet browser calculator. Underneath Summa is a modular design to accommodate easy adaptation, and the Summa index works as a distributed architecture to meet scalability challenges.

1 Modular Design

The Summa search system consists of a set of independent modules all developed in Java. The heart of the search system is the indexing module, which builds the Summa index (based on Lucene [2]) and the search module, which provides the search functionality. The data-flow and responsibilities of some of the modules are illustrated in figure 2.

2 Distributed Index

The index used by the State and University Library in September 2007 contains approximately eight million library metadata record. The goal for spring 2008 is 100 million. Two challenges of working with a large index is presentation and performance. The challenge of presenting *useful* large search results is met by ranking and facet browsing as well as 'did-you-mean', 'suggest' and spell checking among other things. The performance challenge is met by distributing the index (see figure 3), introducing incremental update work-flows and introducing new hardware.

Distributed Index & Storage

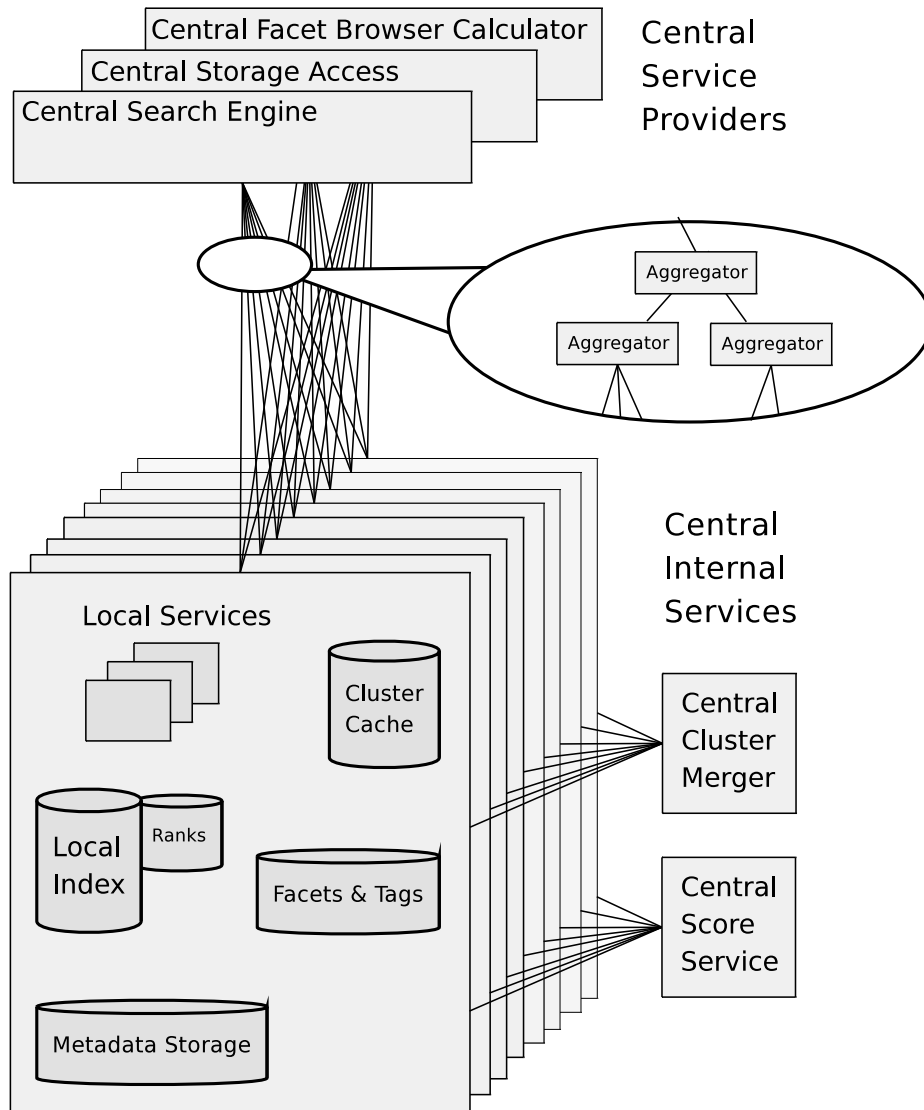


Figure 3: The Summa distributed index architecture.

3 Modules

The Summa modules are:

Website module (presentation layer)

Search modules The search modules provide a central search service, search aggregators, local search nodes and local core search functionality.

Search service Offers the central search web service.

Search aggregator A search aggregator merges search results from the local search nodes and passes the merged result to the central search service.

Search node A thin wrapper around the core search functionality offered by the search module.

Search Offers the core search functionality of a local index.

Index module Indexing is a local service. The basic operation of the indexing module is adding metadata documents to the local index. The module retrieves the individual metadata records from storage, extracts the information to be indexed from the different formats using different xslt's, creates and enriches index documents and updates the index.

Storage module The storage module offers both a central and a local service, which provide access to a storage. An external storage can be used as a local service by Summa if it provides the access API defined by Summa. Summa currently offers a storage service implemented against a JDBC database, and a storage service implemented against Derby is planned. The ingest service can be used to ingest data into the storage accessed by the storage module.

Ingest module The ingest module supplies a local service, which can ingest data from multiple sources into the same storage. The module is responsible for checking for new incoming data, and ingesting into storage if any.

DICE (DIstributed Computing Environment) module DICE is a simplified implementation of the MapReduce model [1] and is used by the index module and can be used by other modules as well.

Common module The common module contains functions common to a number of the other modules.

Score module The score module provides an internal central service responsible for managing the overall work-flow. This module keeps track of both the local machines, the local services, the aggregators, the central services and a number of small 'folder-watcher' services.

Cluster extractor module The cluster extractor module commands a number of local build services, a central merge service and a number of local provider services. The cluster data structure is build locally, merged centrally and moved back to all the local index machines to be provided locally.

Facet browser module The facet browser module is responsible for services both for building and updating the facet & tag data structure, and for calculating and merging the facet browser result for a given query.

Ranking module The ranking module is responsible for building and updating an external ranking data structure in which the index can look up 'corrected' ranks for the index documents, i.e. ranks modified to match the complete index (the complete index is the 'sum' of all the local indexes).

Did-you-mean module Provides did-you-mean functionality.

Suggest module Provides suggest functionality.

Spellchecker module Provides spell checking functionality.

4 Work-flows

Distributing the index means that we are working with a number of local indexes, and when performing a search, we search in all the local indexes and combine the results. This means that the work-flows become a little more complicated. The two central work-flows of the Summa search system is adding data to the system and searching in the data.

4.1 How is data added to the Summa search system?

When we want to add a new library meta-data record to Summa, the typical data-flow is the following ('base' tells us the format of the original file and thus determines the appropriate xslt):

- Place new record in designated folder (on designated machine).

- New record is found by a 'folder-watcher' and moved to appropriate folder on appropriate local machine.
- New record is found by local ingest service and ingested into storage; 'base' is set according to folder.
- Local index module asks assigned storage modules for news and receives new record (the storage is not necessarily on the same machine).
- The record is transformed to an index document using the xslt determined by the 'base'.
- The document is enriched by calling the existing enrichment services, such as the cluster provider from the cluster extractor module.
- The document is added to index.
- The document is sent to all modules responsible of updating external data structures, such as the facet browser module, which updates the external facet and tag data structure.

The add operation is almost the same for full (re-)indexing and for incremental updates. It is however faster to build a new external facet & tag data structure from scratch than by incremental updates.

Incremental updates can also be deletes, and an actual update can be done as a delete and an add operation.

After performing an incremental update, the ranks and clusters of the 'old' documents in index are no longer up-to-date. Also the xslt's may have been updated, and we do a full re-indexing at least weekly. The re-index work-flow is:

- Build new cluster data structure.
- Build index: For each record in storage:
 - Transform to index document,
 - Enrich document (new clusters etc.),
 - Add document to index.
- Build new ranking data structure.
- Build new facet & tag data structure.

Full re-indexing is very expensive, and 'rolling' updates will probably be introduced.

4.2 How is a search performed?

When the user enters a query there are a number of parallel work-flows. First the user may get help from the suggest service, and then the user receives both a search result and a facet browser result. If the search result is empty, the user gets a did-you-mean result. The search work-flow is the following:

- The central search service receives and parses the query.
- The query is send (via the aggregators) to all the local search nodes.
- Each of the local search nodes use the local core search functionality to search in the local index and return a search result to an aggregator.
- Each aggregator receives search results from a number of nodes, merges the results and returns this result to another aggregator or to the central search engine.
- Finally the central search engine receives a number of search results, merges these and then returns the final result.

The facet browser work-flow is very similar to the search work-flow:

- The central facet browser service receives and parses the query.
- The query is send to all the local facet browser nodes.
- Each of the local facet browser nodes search in the local index using the local core search functionality to obtain a 'slim' search result.
- The slim result and the local facet & tag data structure is used to count the exact number of all tags in all facets occurring in the result, and the 'top tags' of all the facets are returned.
- The central facet browser service receives a number of facet & tag results, merges the results¹ and returns the final result.

When a user requests a single record, the record is retrieved from the appropriate storage and similar items are suggested by the search modules.

¹The merged result is not necessarily exact, but we are working on a new merge algorithm inspired by the dissertation *TopX* by Martin Theobald [3].

References

- [1] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
- [2] Otis Gospodnetić and Erik Hatcher. *Lucene in Action*. Manning Publications, 2005. See also lucene.apache.org/java/.
- [3] Martin Theobald. *TopX: Efficient and Versatile Top-k Query Processing for Text, Structured, and Semistructured Data*. PhD thesis, Universität des Saarlandes, April 2006.