NoSQL Technologies (Part II)

COMP9313: Big Data Management

Introduction to ElasticSearch

source

Indexing Overview

- Why do we need indexing?
  - Much of the information is represented as text (Web pages, business documents, health records)
  - Searching can be done through linear scan, to a certain extent (e.g., using Unix's grep)
  - Linear scan has its limitations:
    - Scanning large collections of documents (with billions or trillions of words) becomes very slow for most applications (specially interactive ones)
    - More flexible operations might be impractical using grep (e.g. finding words that appear "near" to other words)
    - Ranked retrieval -> Rank retrieval results base on a given matching criteria.

C. Manning et al., "Introduction to information retrieval", Cambridge University Press, Online edition, 2009

Inverted Index

反向

- Key idea -> And index that maps terms (e.g. words) to the documents where they occur

Inverted list

Terms	Documents
act	1, 4, 63, 77, 143, ...
Australia	2, 4, 89, 91, 231...
constitution	4, 8, 99, 107, 431...
...	...

dictionary
(terms)

postings list
(documents identified
by a docID)

Steps to Build an Inverted List

1. Collect documents that needs to be indexed
2. Turn documents in to a list of tokens (tokenization)
3. Perform preprocessing to produce a normalized list of tokens (e.g. stemming)
4. Create list of terms and the corresponding postings (documents) where they occur
5. Sort terms and postings
6. Record (in dictionary) stats such as document frequency

Steps to Build an Inverted List

Doc 1
I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

Doc 2
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

⟹

term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

⟹

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

C. Manning et al., "Introduction to information retrieval", Cambridge University Press, Online edition, 2009

Boolean queries using Inverted Index

- Example task: Locate documents where terms ”Caesar” and “Capitol” occur together.
- Boolean query: `"Caesar" AND "Capitol"`
- Steps:
  1. Locate `"Caesar"` in dictionary
  2. Retrieve postings where it appears
  3. Locate `"Capitol"` in dictionary
  4. Retrieve postings where it appears
  5. Perform the intersection between the two postings lists

C. Manning et al., “Introduction to information retrieval”, Cambridge University Press, Online edition, 2009

ElasticSearch

Elasticsearch

elasticsearch

source

- Open source search engine based on Apache Lucene
- Initial release in 2010
- Provides a distributed, full-text search engine with a REST APIs
  - E.g. GET http://localhost:9200/person/student/8871

Elasticsearch

- Document oriented (JSON as serialization format for documents)

source

- Developed in Java (cross platform)
- Focused on scalability – distributed by design
- Highly efficient search

Elasticsearch Use Cases

- E-commerce
  - Online web stores
  - Fast search for products
  - Autocomplete suggestions
- Storage, analysis and mining of transaction data
  - Trends
  - Statistics
  - Summarizations
- Analytics/Business intelligence
  - Investigation
  - Analysis
  - Visualization
  - Ad-hoc business questions

Elasticsearch Elements

- Cluster
  - An Elasticsearch cluster is a collection of nodes (servers)
  - Identified by a unique name
  - Data is stored in this collection of nodes
  - Provide indexing and search capabilities across all nodes

Elasticsearch Elements

- Node
  - A single server in the cluster
  - Identified by a unique name
  - Stores all or parts of the whole dataset
  - Contributes to the indexing and search capabilities of Elasticsearch

Elasticsearch Elements

- Shard
  - Individual instances of Apache Lucene index
  - Elasticsearch leverages Lucene indexing in a distributed environment
- Index
  - Distributed across shards
  - Collection of documents (e.g. person, employee, etc.)
  - Identifiable by a name
  - Replicas (fault tolerance)
  - Analogy to RDMS: Index → Database

Elasticsearch Elements

- Type
  - Category of documents of the same class (e.g. product, employee)
  - Types have a name and mapping
  - Indexes can have one or more types
  - Analogy to RDMS: Type → Table

Elasticsearch Elements

- Mapping
  - Defines the fields contained in a given Type
  - Describes data type for each field (e.g. String, Integer, etc.)
  - Describes how fields must be indexed and stored
  - Dynamic mapping is possible
  - Analogy to RDMS: Mapping → Schema of Table

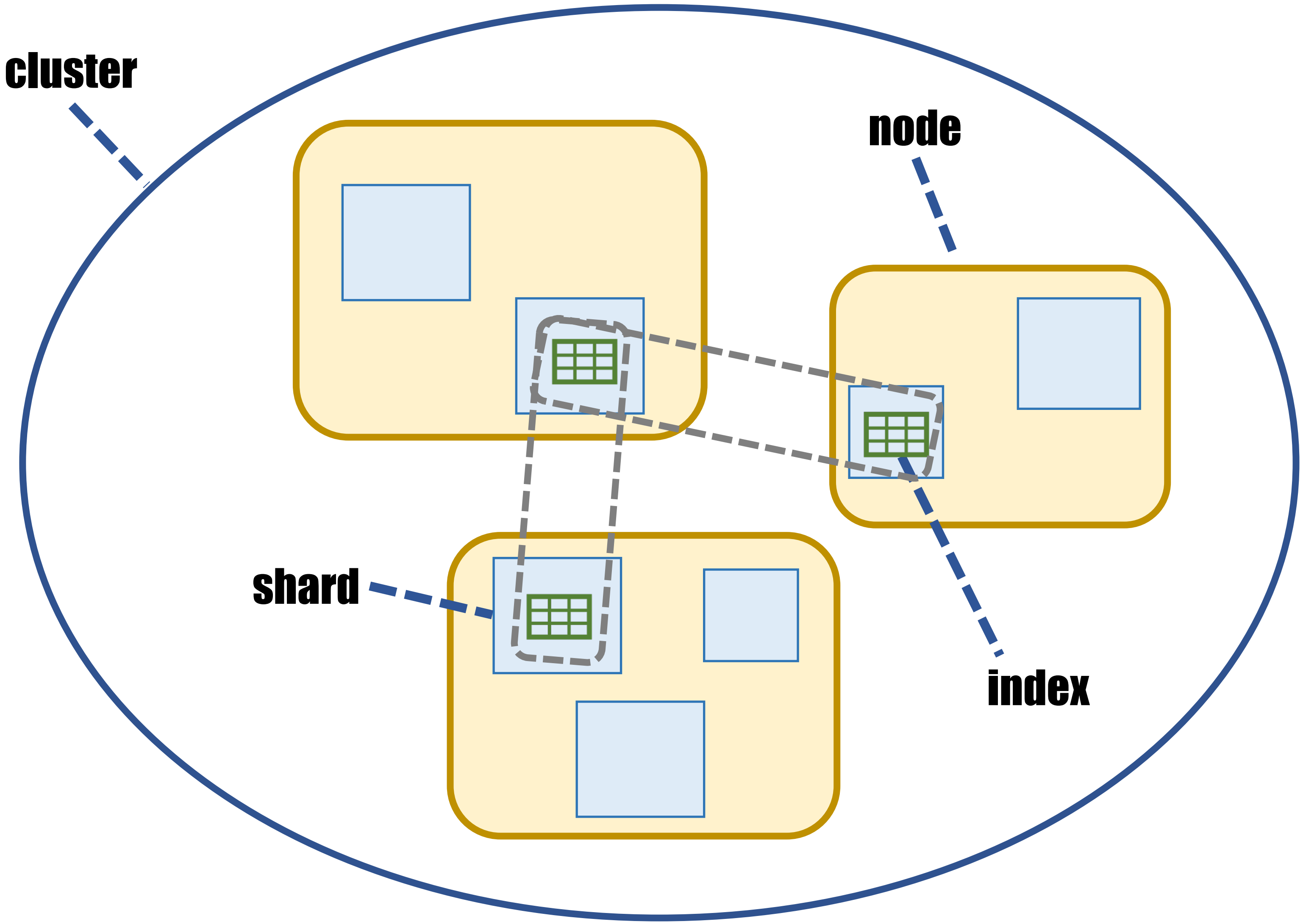
Elasticsearch Elements

- Document
  - Basic unit of information
  - Documents contain fields (key/value pairs)
  - ElasticSearch uses JSON to represent documents
  - Analogy to RDMS: Document → Tuple

Elasticsearch Elements

- Replicas
  - Copy of a shard
  - Provides fault tolerance (shards and node failures)
  - Scalability -> Queries can be executed in parallel
  - Default ElasticSearch configuration:
    - 5 primary shards
    - 1 replica for each index

Elasticsearch Ecosystem



Search APIs: Query String

- Querying using **query strings** (HTTP request)
  - Search the twitter index:

    ```
    GET /twitter/_search?q=user:kimchy
    ```

  - Search all indices

    ```
    GET /_all/tweet/_search?q=tag:wow
    ```

  - Search within specific types

    ```
    GET /twitter/tweet,user/_search?q=user:joe
    ```

- Not all search options are available using this mode

https://www.elastic.co/guide/en/elasticsearch/reference/6.1/index.html

Search APIs: DSL

- Querying using ElasticSearch DSL

```
GET /_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "title":   "Search"        }},
        { "match": { "content": "Elasticsearch" }}
      ],
      "filter": [
        { "term":  { "status": "published" }},
        { "range": { "publish_date": { "gte": "2015-01-01" }}}
      ]
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/6.1/index.html