Modern Information Retrieval

Chapter 10

Parallel and Distributed IR

with Eric Brown

Introduction
A Taxonomy of Distributed IR Systems
Data Partitioning
Parallel IR
Cluster-based IR
Distributed IR
Federated Search
Retrieval in Peer-to-Peer Networks

Introduction

- The volume of online content today is staggering and it has been growing at an exponential rate
- On at a slightly smaller scale, the largest corporate intranets now contain several million Web pages
- As document collections grow larger, they become more expensive to manage
- In this scenario, it is necessary to consider alternative IR architectures and algorithms
- The application of parallelism and distributed computing can greatly enhance the ability to scale IR algorithms

Data Partitioning

Data Partitioning

- IR tasks are typically characterized by a small amount of processing applied to a large amount of data
- How to partition the **document collection** and the **index**?

Data Partitioning

- Figure below presents a high level view of the data processed by typical search algorithms

Indexing Items

Documents

$$\begin{array}{c|cccccc}
 & k_1 & k_2 & \ldots & k_i & \ldots & k_t \\
\hline
d_1 & w_{1,1} & w_{2,1} & \ldots & w_{i,1} & \ldots & w_{t,1} \\
d_2 & w_{1,2} & w_{2,2} & \ldots & w_{i,2} & \ldots & w_{t,2} \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
d_j & w_{1,j} & w_{2,j} & \ldots & w_{i,j} & \ldots & w_{t,j} \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
d_N & w_{1,N} & w_{2,N} & \ldots & w_{i,N} & \ldots & w_{t,N}
\end{array}$$

- Each row represents a document d_j and each column represents an indexing item k_i

Data Partitioning

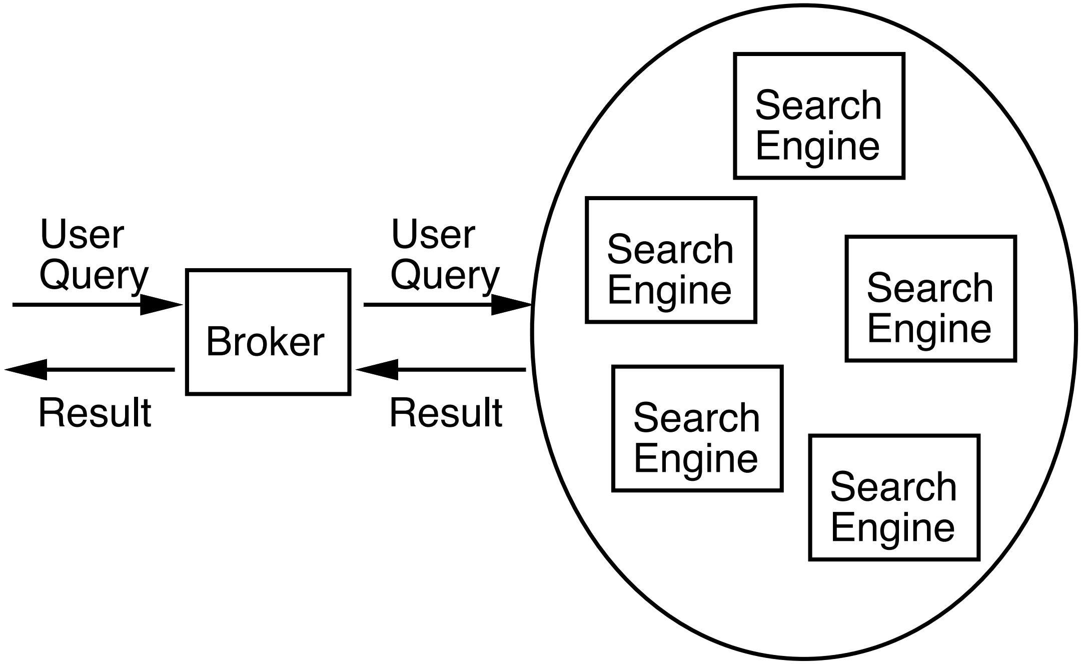
- **Document partitioning** slices the matrix horizontally, dividing the documents among the subtasks
- The N documents in the collection are distributed across the P processors in the system
- During query processing, each parallel process evaluates the query on N/P documents
- The results from each of the sub-collections are combined into a final result list

Data Partitioning

- In **term partitioning**, the matrix is sliced vertically
  - It divides the indexing items among the P processors
- In this way, the evaluation procedure for each document is spread over multiple processors
- Other possible partition strategies include divisions by **language** or **other intrinsic characteristics** of the data
- It may be the case that each independent search server is focused on a particular subject area

Collection Partitioning

- When the distributed system is **centrally administered**, more options are available
- The first option is just the replication of the collection across all search servers
- A **broker** routes queries to the search servers and balances the load on the servers:



Collection Partitioning

- The second option is random distribution of the documents
- This is appropriate when a large document collection must be distributed for performance reasons
- However, the documents will always be viewed and searched as if they are part of a single, logical collection
- The broker broadcasts every query to all search servers and combines the results for the user

Collection Partitioning

- The final option is explicit semantic partitioning of the documents
- Here the documents are either already organized into semantically meaningful collections
- How to partition a collection of documents to make each collection “well separated” from the others?
  - Well separated means that each query maps to a distinct collection containing the largest number of relevant documents

Inverted Index Partitioning

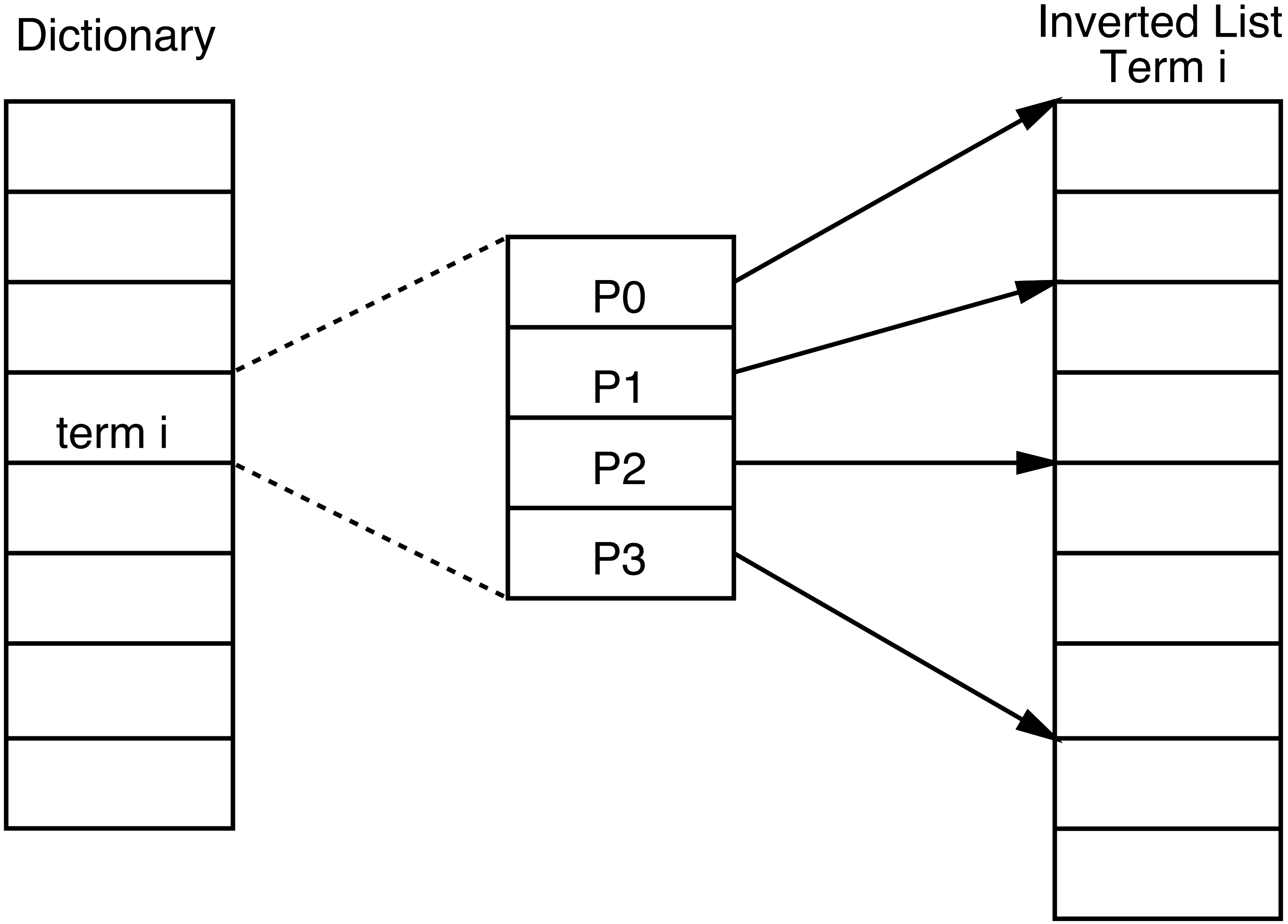
- We first discuss inverted indexes that employ document partitioning, and then we cover term partitioning
- In both cases we address the indexing and the basic query processing phase
- There are two approaches to document partitioning in systems that use inverted indexes
  - Logical document partitioning
  - Physical document partitioning

Logical Document Partitioning

- In this case, the data partitioning is done logically using the same inverted index as in the original algorithm
- The inverted index is extended to give each processor direct access to their portion of the index
- Each term dictionary entry is extended to include P pointers into the corresponding inverted list
- The j-th pointer indexes the block of document entries in the inverted list associated with the sub-collection in the j-th processor

Logical Document Partitioning

- Extended dictionary entry for document partitioning



Logical Document Partitioning

- When a query is submitted to the system, the broker initiates P parallel processes to evaluate the query
- Each process executes the same document scoring algorithm on its document sub-collection
- The search processes record document scores in a single shared array of document score accumulators
- Then, the broker sorts the array of document score accumulators and produces the final ranking

Physical Document Partitioning

- In this second approach, the documents are physically partitioned into separate sub-collections
- Each sub-collection has its own inverted index and the processors share nothing during query evaluation
- When a query is submitted to the system, the broker distributes the query to all of the processors
- Each processor evaluates the query on its portion of the document collection, producing a intermediate hit-list
- The broker then collects the intermediate hit-lists from all processors and merges them into a final hit-list
- The P intermediate hit-lists can be merged efficiently using a binary heap-based priority queue

Physical Document Partitioning

- Each process may require global term statistics in order to produce globally consistent document scores
- There are two basic approaches to collect information on global term statistics
  - The first approach is to compute global term statistics at indexing time and store these statistics with each of the sub-collections
  - The second approach is to process the queries in two phases
    1. Term statistics from each of the processes are combined into global term statistics
    2. The broker distributes the query and global term statistics to the search processes

Physical Document Partitioning

- To build the inverted indexes for physically partitioned documents, each processor creates its own index
- In the case of replicated collections, indexing the documents is handled in one of two ways
  - In the first method, each search server separately indexes its replica of the documents
  - In the second method, each server is assigned a mutually exclusive subset of documents to index and the index subsets are replicated across the search servers
- A merge of the subsets is required at each search server to create the final indexes
- In either case, document updates and deletions must be broadcast to all servers in the system

Comparison

- Logical document partitioning requires less communication than physical document partitioning
  - Thus, it is likely to provide better overall performance
- Physical document partitioning, on the other hand, offers more flexibility
  - E.g., document partitions may be searched individually
- The conversion of an existing IR system into a parallel system is simpler using physical document partitioning
- For either document partitioning scheme, threads provide a convenient programming paradigm for creating the search processes

Term Partitioning

- In term partitioning, the inverted lists are spread across the processors
- Each query is decomposed into items and each item is sent to the corresponding processor
- The processors create hit-lists with partial document scores and return them to the broker
- The broker then combines the hit-lists according

Term Partitioning

- The queries can be processed concurrently, as each processor can answer different partial queries
- However, the query load is not necessarily balanced, and then part of the concurrency gains are lost
- Hence, the major goal is to partition the index such that:
  - The number of contacted processors/servers is minimal; and
  - Load is equally spread across all available processors/servers
- We can use query logs to split the index vocabulary among the processors to achieve the goal above
- A complementary technique is to process the query using a pipeline of processors

Overall Comparison

- Document partitioning affords simpler inverted index construction and maintenance than term partitioning
- Assuming each processor has its own I/O channel and disks, document partitioning performs better
- When terms are uniformly distributed in user queries, term partitioning performs better
- In fact, Webber *et al* show that term partitioning results in lower utilization of resources
- More specifically, it significantly reduces the number of disk accesses and the volume of data exchanged

Overall Comparison

- The major drawback of document partitioned systems:
  - Many not needed operations are carried out to query sub-collections possibly containing few relevant documents
- The main disadvantage of term partitioning:
  - It have to build and maintain the entire global index, which limits its scalability
- In addition, term partitioning has a larger variance regarding answer time and fixing this needs more complicated balancing mechanisms

Suffix Arrays

- We can apply document partitioning to suffix arrays in a straight forward fashion
- As before, the document collection is divided among the P processors and each partition is treated as an independent collection
- The system can then apply the suffix array construction techniques to each of the partitions
- During search, the broker broadcasts the query to all of the search processes
- Then the intermediate results are merged into a final hit-list

Suffix Arrays

- If all of the documents will be kept in a single collection, we can still exploit the parallel processors to reduce indexing time
- In the suffix array construction algorithm for large texts, each of the merges of partial indexes is independent
- Therefore all of the $O((n/M)^2)$ merges may be run in parallel on separate processors
- After all merges are complete, the final index merge may be performed

Suffix Arrays

- In term partitioning for a suffix array, each processor is responsible for a lexicographical interval of the array
- During query processing, the broker distributes the query to the processors that contain the relevant portions of the suffix array and merges the results
- Note that when searching the suffix array, all of the processors require access to the entire text
- However, on a single parallel computer with shared memory, the text may be cached in shared memory