Information Retrieval Methods

Helena Ahonen-Myka
Spring 2007, part 12
Parallel and distributed IR

In this part

- Parallel information retrieval
- Distributed information retrieval

Parallel and distributed IR

- The amount of electronic information is huge
  - Web
  - Commercial collections
  - Corporate intranets
- Disk space becomes cheaper and electronic content becomes easier to produce, download and store

Parallel and distributed IR

- As document collections grow larger, they become more expensive to manage with an IR system
  - Searching and indexing costs grow with the size of the underlying document collection
  - Larger collections result in longer response times
- As more documents are added to the system, performance may deteriorate to the point where the system is no longer usable
- à parallel and distributed architectures and algorithms are needed

Taxonomy of parallel architectures

- SISD: single instruction stream, single data stream
- SIMD: single instruction stream, multiple data stream
- MISD: multiple instruction stream, single instruction stream
- MIMD: multiple instruction stream, multiple data stream

MIMD architectures

- A MIMD computer contains N processors, N instruction streams, and N data streams
- Each processor has its own control unit, processing unit, and local memory
- MIMD systems usually include shared memory or a communication network that connects the processors to each other
  - A high degree of interaction: tightly coupled
  - A low degree of interaction: loosely coupled

MIMD architectures

- Multitasking
  - Each of the processors runs a separate, independent search engine
  - Search engines do not cooperate to process individual queries, but they may share data
  - A broker accepts search requests and distributes them among the available search engines
  - Throughput is increased, as more requests can be processed, but the response time of individual queries remains unchanged

7

MIMD architectures

- Challenges of multitasking
  - How to balance hardware resources: when the number of processors grow, also the number of disks and I/O channels has to grow
  - If the inverted index does not fit into main memory à the processors compete for disk access à bottleneck at the disk could eliminate the throughput gains from the addition of more processors

8

MIMD architectures

- To improve query response time, the computation required to process a single query can be partitioned into subtasks and distributed among the multiple processors
- The broker accepts a query and distributes it among the search processes
- Each of the search processes evaluates a portion of the query and transmits an intermediate result back to the broker
- The broker combines the intermediate results into a final result for presentation to the end user

9

MIMD architectures

- Typical in IR computation: a small amount of processing per data item applied to a large amount of data
- How to partition the computation à how to partition the data
- Two approaches:
  - Document partitioning divides the documents among the subtasks
  - Term partitioning divides the index terms among the processors

10

MIMD architectures

- Document partitioning
  - The N documents in the collection are distributed across the P processors in the system
    - P subcollections of N/P documents each
  - During query processing, each processor evaluates the query on the subcollection assigned to it
  - Results from each of the subcollections are combined into a final result list
- Term partitioning
  - Divides terms among the P processors such that the evaluation procedure for each document is spread over multiple processors in the system

11

Partitioning

- Logical document partitioning
- Physical document partitioning
- Term partitioning

12

Logical document partitioning

- The inverted file is extended to give each parallel process direct access to that portion of the index related to the processor's subcollection of documents
- Each term dictionary entry is extended to include P pointers into the corresponding inverted list
  - j^{th} pointer indexes the block of document entries in the inverted list associated with the subcollection in the j^{th} processor

13

Logical document partitioning

- When a query is submitted to the system, the broker first ensures that the necessary term dictionary and inverted file entries are loaded into shared memory
  - All of the parallel processes can access a single shared copy
- The broker initiates P parallel processes to evaluate the query
- Each process executes the same document scoring algorithm on its subcollection, using the extended dictionary to access the appropriate entries in the inverted file

14

Logical document partitioning

- The search processes record document scores in a single shared array of document score accumulators and notify the broker when they have completed
- After all the search processes have finished, the broker sorts the array of document score accumulators and produces the final ranked list of documents

15

Physical document partitioning

- The documents are physically partitioned into separate, self-contained subcollections (one for each processor)
- Each subcollection has its own inverted file, and the processes share nothing during the query processing
- The broker distributes a query to all of the search processes
- Each process evaluates the query on its portion of the document collection and produces a local, intermediate result list
- The broker collects the intermediate lists from all the processes and merges them into a final result list

16

Physical document partitioning

- The merge procedure assumes that the parallel search processes produce globally consistent document scores
- Depending on the ranking algorithm, each parallel search process may require global term statistics, e.g. document frequency (df)
- The global term statistics can be collected
  - during indexing, or
  - during query processing: first global term statistics are computed, then a query is distributed to the processors

17

Logical vs. physical document partitioning

- Logical document partitioning requires less communication than physical document partitioning (with similar parallelization) à likely to produce better overall performance
- Physical document partitioning offers more flexibility, and conversion of an existing IR system into parallel IR system is simpler

18

Term partitioning

- A single inverted file is created for the document collection
- Inverted lists are spread across the processors
- During query evaluation, the query is decomposed into terms and each term is sent to the processor that holds the corresponding inverted list
- The processors create result lists with partial document scores and return them to the broker

19

Term partitioning

- The broker combines the result lists according to the semantics of the query
  - Boolean query: union, intersection, or subtraction
  - Ranked query: the result lists contain term scores that must be combined acording to the ranking formula

20

Distributed IR

- Distributed computing is the application of multiple computers connected by a network to solve a single problem
- A distributed computer system is like a MIMD parallel processor with
  - a relatively slow inter-processor communication channel
  - a freedom to employ a heterogenous collection of processors in the system

21

Distributed IR

- Distributed systems typically consist of
  - A set of processes, each running on a separate processing node (server)
  - A broker process is responsible for
    - accepting client requests,
    - distributing the requests to the servers,
    - collecting intermediate results from the servers, and
    - combining the intermediate results into a final result for the client

22

Distributed IR vs. parallel IR

- In distributed computing, the subtasks run on different computers and the communication between the subtasks is performed using a network protocol such as TCP/IP
- It is also more common to employ a procedure for selecting a subset of the distributed servers for processing a particular request rather than broadcasting every request to every server in the system

23

Algorithmic IR issues

- How to distribute documents across the distributed search servers?
  - Collection partitioning
- How to select which servers should receive a particular search request?
  - Source selection
- How to combine the results from the different servers?
  - Merging the results

24

Collection partitioning in a decentralized system

- In a system comprising independently administered, heterogenous search servers, the distributed document collections will be built and maintained independently
  - There is no central control of the document partitioning procedure
  - It may be that each search server is focused on a particular subject area

25

Collection partitioning in a centralized system

- The collection can be replicated across all of the search servers
  - Appropriate when the collection is small enough to fit on a single search server, but high availability and query processing throughput are required
  - The parallelism in the system is being exploited via multitasking, and the broker's job is to route queries to the search servers and balance the loads on the servers

26

Indexing of partitions (in a centralized system)

- Indexing the documents is handled in one of two ways
  - Each search server separately indexes its replica of the documents
  - 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 server to create the final indexes

27

Updates (in a centralized system)

- Document updates and deletions must be broadcast to all servers in the system
- Document additions may be broadcast, or they may be batched and partitioned depending on their frequency and how quickly updates must be reflected by the system

28

Collection partitioning in a centralized system

- The second option: random distribution of the documents
  - Appropriate when a large document collection must be distributed for performance reasons, but 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 of the search servers and combines the results for the user

29

Collection partitioning in centralized system

- The third option: explicit semantic partitioning of the documents, which are either
  - already organized into semantically meaningful collections, such as by technical discipline, or
  - an automatic clustering or categorization procedure is used to partition the documents into subject-specific collections

30

Source selection

- Source selection is the process of determining which of the distributed document collections are most likely to contain relevant documents for the current query (and therefore should receive the query for processing)
- Simple approach: assume that every collection is equally likely à always broadcast the query to all collections
  - Appropriate when documents are randomly partitioned, or there is significant semantic overlap between the collections

31

Source selection

- The collections can also be ranked according to their likelihood of containing relevant documents
- This is appropriate
  - if documents are partitioned into semantically meaningful collections, or
  - it is prohibitively expensive to search every collection every time
- The basic technique:
  - Treat each collection as if it were a single large document
  - Generate a collection vector for each collection
  - Evaluate the query vector against each collection vector to produce a ranked listing of collections

32

Source selection

- A standard cosine similarity measure can be used: to calculate a tf*idf term weight in the collection vector,
  - term frequency $tf_{i,j}$ is the total number of occurrences of term i in collection j,
  - and the inverse document frequency idf_i for term i is $\log(N/n_i)$, where N is the total number of collections and n_i is the number of collections in which term i appears

33

Source selection

- A danger of this approach is that although a particular collection may receive a high query relevance score, there may not be individual documents within the collection that receive a high query relevance score
- The problem can be avoided by indexing each collection as a series of blocks, where each block contains B documents
  - The query is evaluated against each block
  - The score for a collection is computed from the scores of its blocks

34

Source selection

- Alternative approach to indexing collections: training queries
- A set of training queries are used to build a content model for each collection
- When a new query is submitted to the system, its similarity to the training queries is computed and the content model is used to determine which collections should be searched and how many documents from each collection should be returned

35

Query processing

- Query processing in a distributed IR system:
  1. Select collections to search
  2. Distribute query to selected collections
  3. Evaluate query at distributed collections in parallel
  4. Combine results from distributed collections into final result

36

Query processing

- Step 1 may be eliminated if the query is always broadcast to every document collection in the system
- Otherwise, one of the selection algorithms is used and the query is distributed to the selected collections
- Each of the participating search servers then evaluates the query on the selected collections using its own local search algorithm
- Finally, the results are merged

37

Merging the results

- A number of scenarios
- If the query is Boolean and the search servers return Boolean result sets
  - the final result set = union of the result sets
- If the query involves free-text ranking, a number of techniques are available ranging from simple to complex/accurate

38

Merging the results

- Simplest approach: combine the ranked result lists using round robin interleaving
  - 1: 1st document from the 1st list,
  - 2: 1st document from the 2nd list,
  - ... N: 1st document from the Nth list,
  - N+1: 2nd document from the 1st list,...
- Likely to produce poor quality results, since hits from irrelevant collections are given status equal to that of hits from highly relevant collections

39

Merging the results

- Improvement: merge the result lists based on relevance score
- Unless proper global term statistics are used to compute the document scores, we may get incorrect results
- If documents are randomly distributed such that global term statistics are consistent across all of the distributed collections, the merging based on relevance score is sufficient

40

Merging the results

- If the document collections are semantically partitioned or maintained by independent parties, then reranking must be performed
- Reranking, e.g. , by weighting document scores based on their collection similarity computed during the source selection step
- The weight for a collection can be computed as

$$w = 1 + |C| \cdot (s - \hat{s}) / \hat{s}$$

  - where |C| is the number of collections searched, s is the collection score, and ŝ is the mean of the collection scores

41

Merging the results

- More accurate technique for merging ranked result lists is to use accurate global term statististics
- If the collections have been indexed for source selection, that index will contain global term statistics across all of the distributed ollections
- The broker can include these statistics in the query when it distributes the query to the search servers
- The servers can use these statistics in their processing and produce relevance scores that can be merged directly

42

Merging the results

- If a collection index is unavailable, query distribution can proceed in two rounds of communication
- In the first round, the broker distributes the query and gathers collection statistics from each server
- These statistics are combined by the broker and distributed back to the servers in the second round.

43

Merging the results

- The search protocol can also require that the servers return global query term statistics and per-document query term statistics
- The broker is then free to rerank every document using the query term statistics and a ranking algorithm of its choice
- The end result is a list that contains documents from the distributed collections ranked in the same order as if all of the documents had been indexed in a single collection

44

Parallel and distributed IR

- Many parallel IR algorithms are well suited to both multiprocessor and distributed implementations
- By using an appropriate abstraction layer for inter-process communication, we can easily implement a parallel system that works well on both multiprocessor and distributed architectures with relatively little modification

45

Parallel and distributed IR

- Challenges
  - How to measure retrieval effectiveness on large text collections?
    - How to generate relevance assessments for queries?
    - Pooling techniques used in TREC may not work
  - How to build distributed IR systems from heterogeneous components (=meta-search)?
    - Lack of term statistics from the back-end search servers: reranking of results not possible
    - Each server may have its custom query language: meaning of the query may change

46

In this part

- Parallel IR
  - Multitasking
  - Multiple processors for a query
    - Document partitioning (logical and physical)
    - Term partitioning
- Distributed IR
  - Collection partitioning
  - Source selection
  - Merging the results

47