Parallel and Distributed Information Retrieval

Anil Kumar Akurathi

Department of Computer Science

University of Maryland

Outline

- Why Parallel and Distributed IR systems are needed?
- Parallel generation of Inverted Files for Distributed text collections
- Distributed Algorithms to Build Inverted Files
- Performance Evaluation of a Distributed Architecture

Why Parallel and Distributed IR?

- The amount of information is increasing very rapidly with the increase of the size of the Internet
- Searching and indexing costs increase with the size of the text collection
- More and more powerful machines are expensive
- Parallel and Distributed systems provide cheap alternatives with comparable performance

Advantages of distributed systems

- Provide multiple users with concurrent, efficient access to multiple collections located on remote sites
- Use the resources more efficiently by spreading the work across a network
- Easily extendable to include more sites
- Can be created from the products already available

Parallel generation of Inverted Files

- Strongly connected network of processors
- One central coordinator to distribute queries and to combine results, if necessary
- Scalable Algo for parallel computation of inverted files for large text collections
- Average running cost of $O(t/p)$, where
  - t is the size of the whole text collection
  - p is the number of available processors

Distribution of Text collection

- Documents in the collection are evenly distributed in the network
- Each processor roughly holds

$$b = \frac{t}{p}$$

  - b - subcollection size at each processor
  - t - total text size
  - p - total number of processors

Inverted Files

- An Inverted list structure has
  - A list of all distinct words in the text called *vocabulary*, sorted in lexicographical order
  - vocabulary usually fits in the main memory
  - for each word *w* in vocabulary, an *inverted list* of documents in which the word *w* occurs
  - Any portion of the list that needs to be stored or exchanged through the network is *compressed* to keep the disk accesses and network overhead low

Distribution of Inverted Files

- Local index organization
  - each machine has its own local inverted file
  - very easy to maintain as there is no interaction
  - each query should be sent to all machines
- Global index organization
  - global inverted file for the whole collection
  - For simplicity, index distributed in lexicographic order such that all hold roughly equal portions
  - Queries are sent to only specific machines

Global Index Organization

- Even in the local index organization we need to provide the global occurrence information
- Hence computation of the global index is unavoidable
- Also, global index organization outperforms local index organization on TREC collection queries

Phases in the algorithm

- *Phase 1: Local Inverted Files*
  - each processor builds an inverted file for local text
- *Phase 2: Global Vocabulary*
  - global vocabulary and the portion of the global inverted file to be held by each is determined
- *Phase 3: Global Distributed Inverted File*
  - portions of the local inverted files are exchanged to generate the global inverted file

Phase 1: Local Inverted Files

- Each processor reads *b* bytes of data from disk and builds the inverted file
  - words are inserted in a hash table whose entries point to the inverted lists for each word
  - the inverted for a word w has pairs (d, f) where
    - d - document in which w occurs
    - f - frequency of occurrence
  - inverted lists are compressed but hash table is kept uncompressed and unsorted

Cost for phase 1

$$t_1 = b \times ts1 + b \times ts2$$

- where
  - ts1, ts2: average disk access time and cpu time per byte (in sec), these can be derived experimentally
- linearity assumptions are valid for disk access, for hash table with constant access and for Golomb compression algorithm

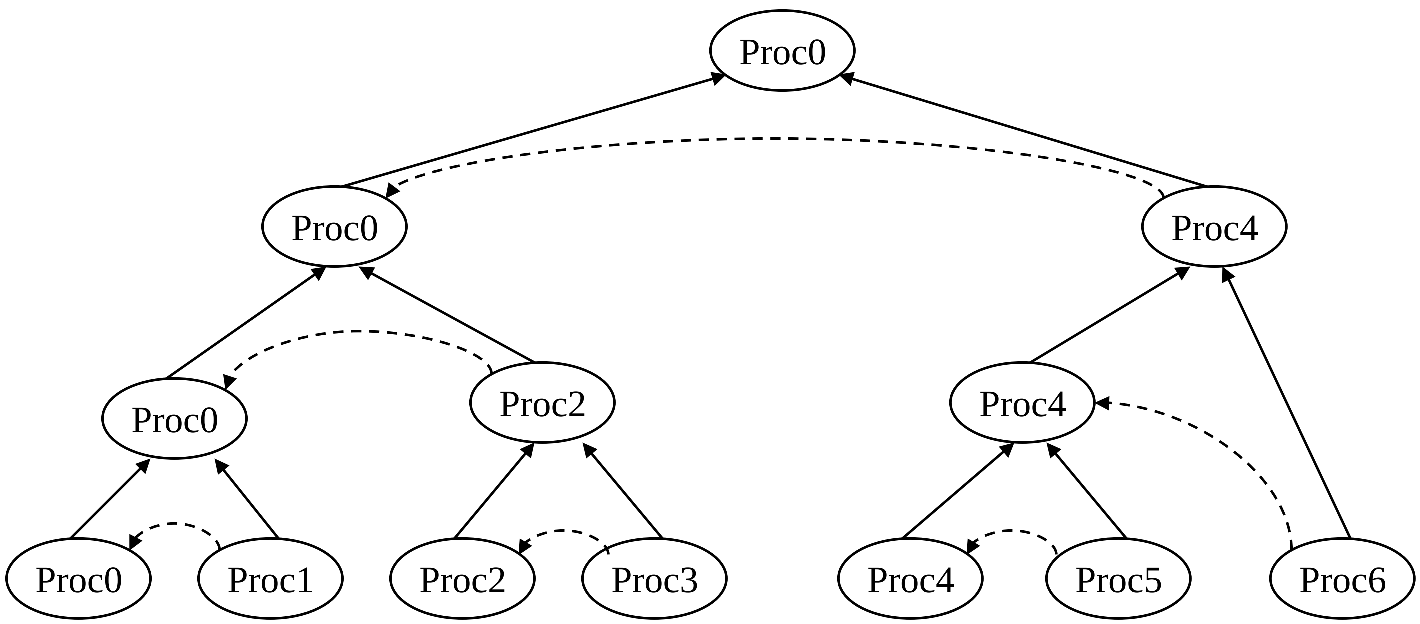
Phase 2: Global Vocabulary

- Processors merge their local vocabularies
  - first, odd numbered processors transfer all their local vocabulary to even numbered processors
  - This pairing process is applied recursively until processor 0 has the global vocabulary (log*p* steps)
  - The size v of the vocabulary can be computed as

$$v = Kt^{\beta} = O(t^{\beta})$$

    - where $0 < \beta < 1$ and K is a constant

Global Vocabulary computation



Global Vocabulary Computation

Cost for Phase 2

$$t_2 = K \sum_{i=0}^{(\log_2 p)-1} S_w (2^i b)^{\beta} \times (ts3 + ts4)$$

- where
  - S_w: average size in bytes of words
  - ts3: average time of network per byte (in sec)
  - ts4: average time of cpu per byte (in sec)

Phase 3: Global Distributed Inverted File

- Processor 0 sorts the global vocabulary and computes the lexicographical boundaries of p equal sized stripes of global inverted file
- This information is broadcast to all processors
- Each processor sorts its local vocabulary
- step-by-step all-to-all communication procedure is followed to exchange the lists

Cost for Phase 3

$$t_3 = K_q \times v_g \log v_g \times ts5 + K_q \times v_l \log v_l \times ts5 + (p-1)K_c \times \frac{2K_i b}{p} \times (ts6 + ts7)$$

- where
  - vl: size (in English words) of the local vocabulary
  - vg: size of the global vocabulary
  - Kq: proportionality constant for quicksort
  - Kc: compression factor
  - Ki: ratio of inverted list size and text size
  - ts5: average cpu time per English word (in sec)
  - ts6, ts7: average network and cpu time per byte (in sec)

Average total cost

$$
\begin{aligned}
&O(b)I + O(b)C + && (Phase 1) \\
&O(t^{\beta})I + O(t^{\beta})C + && (Phase 2) \\
&O(t^{\beta} \log t^{\beta})I + O(b)C && (Phase 3)
\end{aligned}
$$

- where I is the computation internal costs and C is the communication costs
- by observing that b >> t^{β} for common English texts, the average total cost is estimated as

$$O(b)I + O(b)C = O(\frac{t}{p})I + O(\frac{t}{p})C$$

Distributed Algorithms

- Same type of configuration but for a much larger collection
- Total distributed main memory is considerably smaller than the inverted file to be generated
- TREC-7 collection of 100 gigabytes indexed in 8 hours on 8 processors with 16 MB RAM
- Algorithms for inverted files that do not need to be updated incrementally

Design Decisions

- Index terms are ordered lexicographically
- The pairs $[d_j, f_{i,j}]$ for each index term k_i are sorted in the decreasing order of $f_{i,j}$
  - d_j - j^{th} document
  - $f_{i,j}$ - frequency of i^{th} index term k_i in d_j
- The above sorting helps in retrieving less number of documents from disk when there is a threshold for $f_{i,j}$

A sequential disk based algorithm

- In phase a, all documents are read from disk and processed for index terms to create the perfect hashed vocabulary
- In phase b, all documents are parsed again to get the [d_j, $f_{i,j}$] pairs (second access can be avoided if the vocabulary is kept in memory)
- disk-based multi-way merge is done to combine the partial inverted lists

Local buffer and Local lists - LL

- This is similar to what we have discussed before
  - Phase1: each processor builds its own local inverted list
  - Phase2: the global vocabulary and portion of the global inverted file for each processor are determined
  - Phase3: processors exchange the inverted lists in an all-to-all communication procedure

LL algorithm merging procedures

- In phase 1, when the main memory is full, the inverted list is written to disk.
- If there are R such runs, at the end of the phase, an R-way merge is performed
- Similarly, in phase 3, a p-way merge is performed after receiving the portions of the inverted lists from other processors

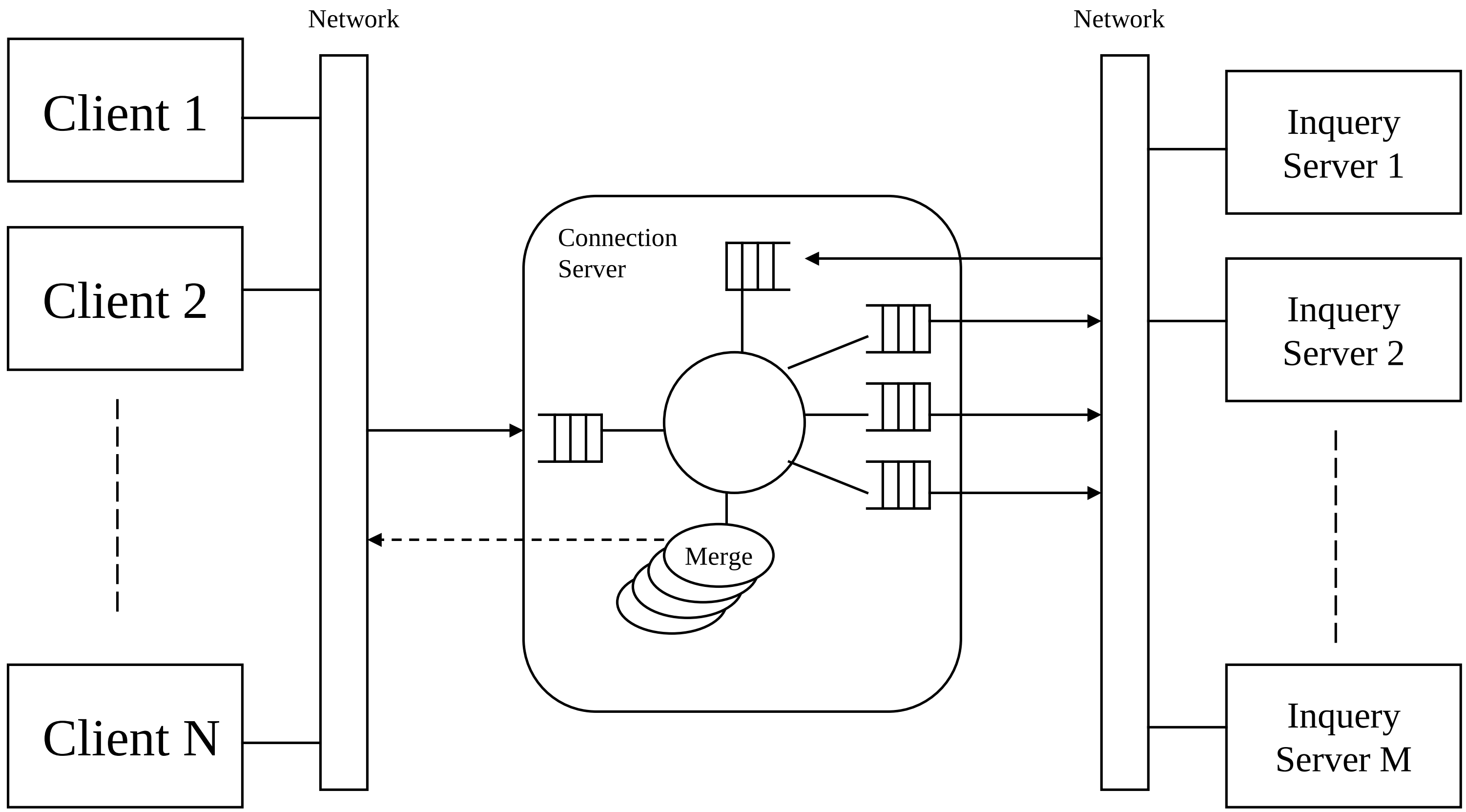
Local buffer and Remote lists - LR

- This assumes that the information on global vocabulary is available early on
- To avoid the R-way merging done in LL, the portions of the inverted lists are directly sent to the other processors (now a pR-way merging is needed)
- This avoids the disk I/O associated with R-way merging procedure

Remote buffer and Remote lists - RR

- An improvement over LR is to assemble the triplets in small messages early on and to send them to avoid storage at local buffer
- These messages need to be large enough to reduce the network overheads
- Transmission through network and reading of local documents from disk can be overlapped
  - Very little cost associated with network transmission

Performance evaluation of a Distributed Architecture



Distributed Information Retrieval System

Architecture

- *Inquery* server, a full-text information retrieval model is used
- Clients connect to a *connection* server, a central administration broker which intern connects to Inquery servers
- Clients provide the user interface to the retrieval system

IR commands

- Query commands
  - set of words or phrases and a set of collection identifiers
  - response includes document identifiers with estimates
- Summary commands
  - set of document identifiers and their collection identifiers
  - response includes title and first few sentences of the document
- Document commands
  - a document and its collection identifier
  - response includes the complete text of the document

Connection Server

- Forwards the clients commands to appropriate Inquery servers
- Maintains the intermediate responses from the servers until it receives responses from all
- Merges the responses from the servers
  - It is assumed that the relative rankings between documents in independent collections are comparable

Simulation Model

- User configures a simulation by defining the architecture using a simple command language
- CPU, disk and network resources used for each operation are measured
- Utilization percentage of the connection server and Inquery servers is measured
- Evaluation time of a query is computed by adding the evaluation times of individual terms in the query

Evaluation times

- Document retrieval time
  - A constant (0.31 sec) measured after calculating the average retrieval time for 2000 random documents
- Connection server time
  - time to access the connection server (0.1 sec)
  - time to merge the results (17.9 msec for 1000 values)
- Network time
  - sender overhead, receiver overhead and network latency

Simulation parameters

- Number of Clients/Inquery servers (C/IS)
- Terms per Query (TPQ)
- Distribution of terms in queries (QTF)
- Number of Documents that match queries (AR)
- Think Time (TT)
- Document Retrieval / Summary Information (DR/SO)

Transaction sequence

- Evaluate a query
- Obtain summary information of top ranking documents
- *think*
- retrieve documents
- *think*
  - Only natural language queries are modeled
  - structured query operations such as phrase and proximity operators are not modeled

Experiments and results

- Two kinds of experiments
  - Equally distributing a single database among the servers
  - Each server maintains a different database and the clients broadcast to a subset of servers
- Both small and large queries are used
- Performance deteriorates if *connection server* or *Inquery servers* are over utilized
- Architectures with two or four connection servers to eliminate the bottleneck are also used

Distributing a single text collection

- Exploits parallelism by operating simultaneously
- Each client needs to connect to all servers
- Small queries (TPQ = 2)
  - As the number of clients increases, average transaction time increases
  - Going from 1 to 8 servers, improves the performance since the size of the database decreases
  - For more than 8 servers, performance degrades as the connection server becomes over utilized (size of the incoming queue at connection server also increases)

Single text collection, cont.

- Large Queries (TPQ = 27)
  - Performance degrades rapidly as the number of clients increases since the system places greater demands on the Inquery servers
  - For more number of Inquery servers, extremely high utilization of the connection server and Inquery servers causes the degradation
  - Contrast to small queries where Inquery server is highly utilized only for single Inquery server

Multiple text collections

- In the simulation, each client searches half of the available collections on the average
- Hence, work load increases both as a function of the number of Inquery servers and the number of clients
- Small queries (TPQ = 2)
  - connection server utilization increases with the number of clients causing a degrade in the performance
  - Inquery server utilization decreases as the number of Inquery servers increases (size of the incoming queue at connection server also increases)

Multiple text collections, cont.

- Large Queries (TPQ = 27)
  - Performance of the system does not scale for large queries
  - Inquery servers cause a bottleneck as the number of Inquery servers increases
  - Connection server remains idle for most of the time since query evaluation takes most of the time

Multiple connection servers

- Additional connection servers reduce the average utilization of a connection server and increase the performance for small queries
- For 2 connection servers, speadup of 1.94 over single connection server using 128 Inquery servers and 256 clients
- For 4 connection servers, system scales very well for large configurations using small queries

Conclusions

- The architecture provides scalable performance for small queries
- Over utilization of connection server or Inquery servers degrades the performance
- For large queries and extremely high workloads, Inquery servers do not provide good response times
- Adding more connection servers gives good performance for small queries

References

- B.Ribeiro-Neto, E.S.Moura, M.S.Neubert and N.Ziviani. Efficient Distributed Algorithms to Build Inverted Files. In SIGIR'99, Berkley, USA
- B.Ribeiro-Neto, J.P.Kitajima, G.Navarro, C.Santana and N.Ziviani. Parallel generation of inverted files for distributed text collections. In Proc. of Int. Conf. of the Chilean Society of Computer Science, (SCCC'98) pages 149-15, Antofagasta, Chile, 1998
- B.Cahoon and K.S.Mckinley, "Performance Evaluation of a Distributed Architecture for Information Retrieval," ACM SIGIR, Switzerland, Aug., 1996