Prepare all 5 Assignments from Journal with code of 3 assignments.
Then these few hint questions from internet or book.

Unit 1

1) Difference between Data Retrieval and Information Retrieval.

Ans:

	Data Retrieval (DR)	Information Retrieval (IR)
Matching	Exact match	Partial match, best match
Inference	Deduction	Induction
Model	Deterministic	Probabilistic
Classification	Monothetic	Polythetic
Query language	Artificial	Natural
Query specification	Complete	Incomplete
Items wanted	Matching	Relevant
Error response	Sensitive	Insensitive

2) Precision and Recall.

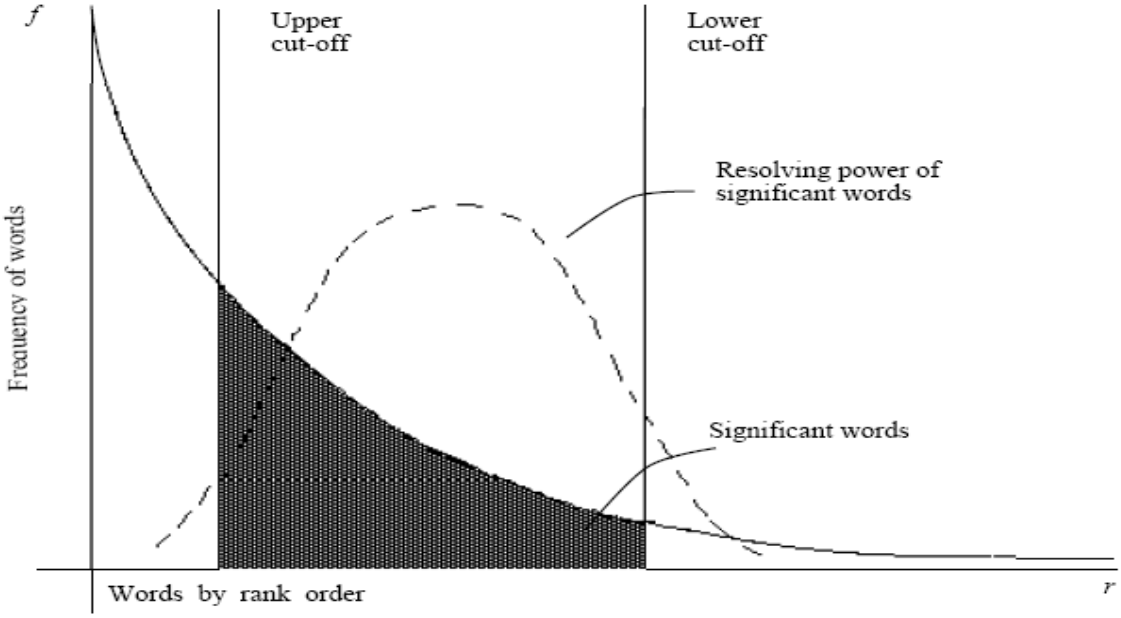
Ans :

Precision: *precision* is the ratio of the number of relevant documents retrieved to the total number of documents retrieved

Recall: *recall* is the ratio of the number of relevant documents retrieved to the total number of relevant documents (both retrieved and not retrieved).

3) Explain Luhn's Idea for finding significant words.

Ans : **Zipf's Law :** the product of the frequency of use of wards and the rank order is Approximately constant.



4) Conflation algorithm

Ans: will usually consist of three parts:
(1) Removal of high frequency words,
(2) Suffix stripping,
(3) Detecting equivalent stems.

5) Indexing exhaustivity and specificity.

Ans***: "Indexing exhaustivity"*** is defined as the number of different topics indexed,
The *index language* ***specificity*** is the ability of the index language to describe topics precisely

6) Five commonly used measures of association in information retrieval.

Ans:

$|X \cap Y|$ *Simple matching coefficient*

$$2\ \frac{|X \cap Y|}{|X| + |Y|}$$ *Dice's coefficient*

$$\frac{|X \cap Y|}{|X \cup Y|}$$ *Jaccard's coefficient*

$$\frac{|X \cap Y|}{|X|^{1/2} \times |Y|^{1/2}}$$ *Cosine coefficient*

$$\frac{|X \cap Y|}{\min(|X|, |Y|)}$$ *Overlap coefficient*

7) Why Normalized versions of the simple matching coefficient are used for measures of association.
Ans:

If $S_1(X, Y) = |X \cap Y|$ $S_2(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|}$

then $|X_1| = 1\ |Y_1| = 1\ |X_1 \cap Y_2| = 1 \Rightarrow S_1 = 1 S_2 = 1$

$|X_2| = 10\ |Y_2| = 10\ |X_2 \cap Y_2| = 1 \Rightarrow S_1 = 1 S_2 =$

1/10

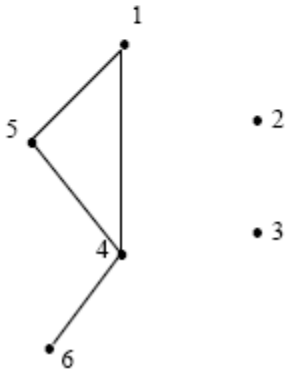
8) Cluster using similarity measures
Ans:

Similarity matrix

1				
2	.6			
3	.6	.8		
4	.9	.7	.7	
5	.9	.6	.6	.9
6	.5	.5	.5	.9
	1	2	3	4

Threshold: .89

Graph:



9) 'Single-Pass Algorithm' for clustering.
Ans:
(1) The object descriptions are processed serially;
(2) The first object becomes the cluster representative of the first cluster;
(3) Each subsequent object is matched against all cluster representatives existing at its processing time;
(4) A given object is assigned to one cluster (or more if overlap is allowed) according to some condition on the matching function;
(5) When an object is assigned to a cluster the representative for that cluster is recomputed;
(6) If an object fails a certain test it becomes the cluster representative of a new cluster.

10) Explain Clustering using dissimilarity matrix. Also explain effect of threshold on clustering.
Ans:

Dissimilarity matrix:

2	.4			
3	.4	.2		
4	.3	.3	.3	
5	.1	.4	.4	.1
	1	2	3	4

Binary matrices:

2	0			
3	0	0		
4	0	0	0	
5	1	0	0	1
	1	2	3	4

Threshold = .1

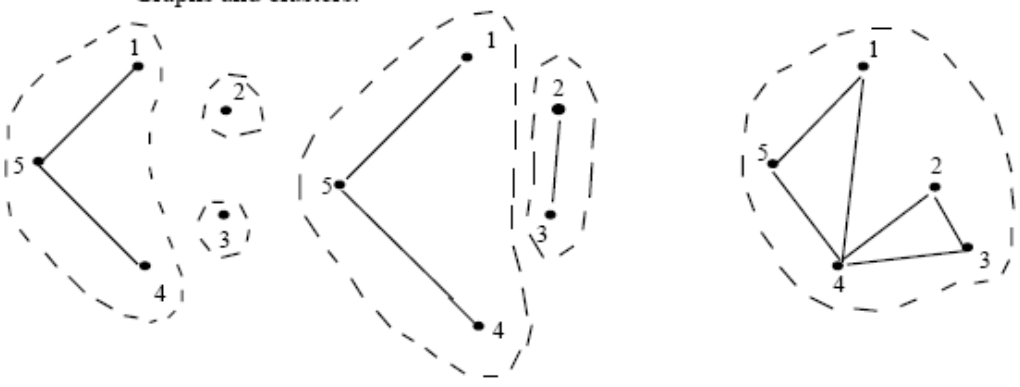
2	0			
3	0	1		
4	0	0	0	
5	1	0	0	1
	1	2	3	4

Threshold = .2

2	0		
3	0	1	
4	1	1	1
5	1	0	0
	1	2	3

Threshold = .3

Graphs and clusters:



Unit 2

1) Explain K-list.
   Ans:
   K-list as a set of records containing K such that:
   (1) The *K*-pointers are distinct;
   (2) Each non-null *K*-pointer in *L* gives the address of a record within *L*;
   (3) There is a unique record in *L* not pointed to by any record containing *K*; it is called the *beginning* of the list; and
   (4) There is a unique record in *L* containing the null *K*-pointer; it is the *end* of the list.

2) Definition of a *directory* of a file.
   Ans: Let *F* be a file whose records contain just *m* different keywords *K*1, *K*2. . . *Km*. let *ni* be the number of records containing the keyword *Ki*, and *hi* be the number of *Ki*-lists in *F*. Furthermore, we denote by *aij* the beginning address of the *j*th *Ki*-list. Then the *directory* is the set of sequences
   (*Ki, ni, hi, ai*1, *ai*2 . . . *aih i*) *i* = 1, 2 . . . *m*

3) Inverted files.
   Ans: *ni* = *hi* for all *i*,
4) Index-sequential files
   Ans: *ni* = *hi* = 1 and *a*11 <*a*21 . . . <*am*1.
5) Multi-lists
   Ans: *hi* = 1.
   Cellular multi-lists
   Ans:
6) Cellular Multi –Lists.
   Ans: The directory for a cellular multi-list will be the set of sequences
   (*Ki, ni, hi, ai*1 . . . *aih i*) *i* = 1, 2. . . *m*
   Where the *hi* have been picked to ensure that a *Ki*-list does not cross a page boundary
7) IR Models
   Ans: Basic concepts, Boolean Model, Vector Model, Fuzzy Set Model.
8) Boolean Search.
   Ans:
   Like query *Q* = (*K*1 AND *K*2) OR (*K*3 AND (NOT *K*4)).
9) Boolean Search on Inverted file
   Ans:

K_1 -list : D_1, D_2, D_3, D_4

K_2 -list : D_1, D_2

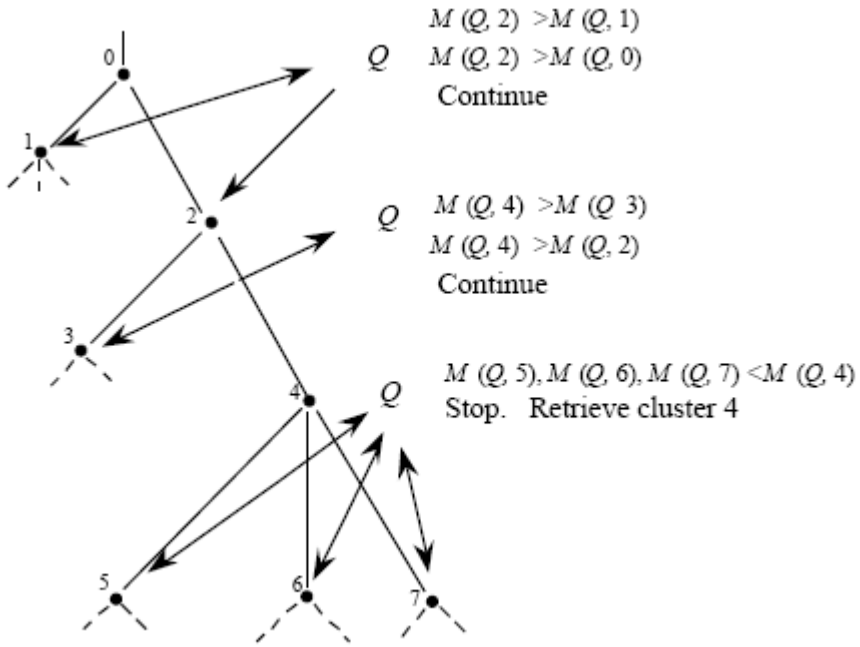
K_3 -list : D_1, D_2, D_3

K_4 -list : D_1

and $Q = (K_1 \text{ AND } K_2) \text{ OR } (K_3 \text{ AND } (\text{NOT } K_4))$

   Then to satisfy the (*K*1 AND *K*2) part we intersect the *K*1 and *K*2 lists, to satisfy the (*K*3 AND (NOT *K*4)) part we subtract the *K*4 list from the *K*3 list.

10) Cluster-based retrieval
    Ans:
    (1) We assume that effective retrieval can be achieved by finding just one cluster;
    (2) We assume that each cluster can be adequately represented by a cluster representative for the purpose of locating the cluster containing the relevant documents;
    (3) If the maximum of the matching function is not unique some special action, such as a look-ahead, will need to be taken;
    (4) The search always terminates and will retrieve at least one document.



Unit 3

1) How effectiveness of IR is measured
   Ans:
   Precision versus recall graph using ranking for query say q.
   Explanation of how graph is plotted also how precision and recalls are calculated from ranking.
2) Precision Histograms
   Ans: $RP_{A/B(i)} = RP_{A(i)} - RP_{B(i)}$
   Where $RP_{A(i)}$ and $RP_{B(i)}$ Are R-precision values of algorithm A and B for i^{th} Query.
   $RP_{A/B(i)}$ versus query number graph is precision histogram.
3) Harmonic mean
   Ans: combines recall and precision.
4) E measure
   Ans: Harmonic Mean which allows specifying interest in recall or precision.
5) User oriented measures
   Ans: coverage and novelty.
6) TREC
   Ans: Text Reference Collection used as a reference for evaluating IR system.
7) Users Interface & Visualization

Unit 4

1) OPAC
   Ans: first, second and third generation functionality.
2) Digital libraries

Ans: Architecture, different issues like multilingual documents, Multimedia documents, structured documents, distributed collection

Unit 5

1) Steps in multimedia IR
   Ans:
   a) Query specification
   b) Query processing and optimization
   c) Query answer
   d) Query iteration
2) MULTOS data model
   Ans:
   Logical, layout and conceptual structure. With example of conceptual structure.
3) Conditions on multimedia data
   Ans:
   Attribute predicate, structural predicate and semantic predicate.
4) MULTOS query language

Ans : like SQL query language.

Unit 6

1) What is page rank? How to compute page rank?
Ans:

PageRank is a link analysis algorithm, named after Larry Page,[1] used by the Google Internet search engine that assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it assigns to any given element *E* is also called the *PageRank of E* and denoted by $PR(E)$.

The name "PageRank" is a trademark of Google, and the PageRank process has been patented (U.S. Patent 6,285,999). However, the patent is assigned to Stanford University and not to Google. Google has exclusive license rights on the patent from Stanford University. The university received 1.8 million shares of Google in exchange for use of the patent; the shares were sold in 2005 for $336 million.

Google describes PageRank:

> “PageRank relies on the uniquely democratic nature of the web by using its vast link structure as an indicator of an individual page's value. In essence, Google interprets a link from page A to page B as a vote, by page A, for page ”

B. But, Google looks at more than the sheer volume of votes, or links a page receives; it also analyzes the page that casts the vote. Votes cast by pages that are themselves "important" weigh more heavily and help to make other pages "important".

In other words, a PageRank results from a "ballot" among all the other pages on the World Wide Web about how important a page is. A hyperlink to a page counts as a vote of support. The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it ("incoming links"). A page that is linked to by many pages with high PageRank receives a high rank itself. If there are no links to a web page there is no support for that page.

Google assigns a numeric weighting from 0-10 for each webpage on the Internet; this PageRank denotes a site's importance in the eyes of Google. The PageRank is derived from a theoretical probability value on a logarithmic scale like the Richter Scale. The PageRank of a particular page is roughly based upon the quantity of inbound links as well as the PageRank of the pages providing the links. It is known that other factors, e.g. relevance of search words on the page and actual visits to the page reported by the Google toolbar also influence the PageRank.[*citation needed*] In order to prevent manipulation, spoofing and Spamdexing, Google provides no specific details about how other factors influence PageRank.[*citation needed*]

Numerous academic papers concerning PageRank have been published since Page and Brin's original paper.In practice, the PageRank concept has proven to be vulnerable to manipulation, and extensive research has been devoted to identifying falsely inflated PageRank and ways to ignore links from documents with falsely inflated PageRank.

Other link-based ranking algorithms for Web pages include the HITS algorithm invented by Jon Kleinberg (used by Teoma and now Ask.com), the IBM CLEVER project, and the TrustRank algorithm.

1) What are challenges of web?
   Ans:

   1] ALGORITHMIC challenges:
   we describe six algorithmic problems that arise in web search engines and that are not or only partially solved: (1) Uniformly sampling of web pages; (2) modeling the web graph; (3) finding duplicate hosts; (4) finding top gainers and losers in data streams; (5) finding large dense bipartite graphs; and (6) understanding how eigenvectors partition the web

   2] Usability Challenges of Web-Based Applications

   3] etc search on internet

Q. Explain search engine architectures in detail.
Ans: google or any other search engine architecture

Q. What are the Metasearchs? Explain with suitable example

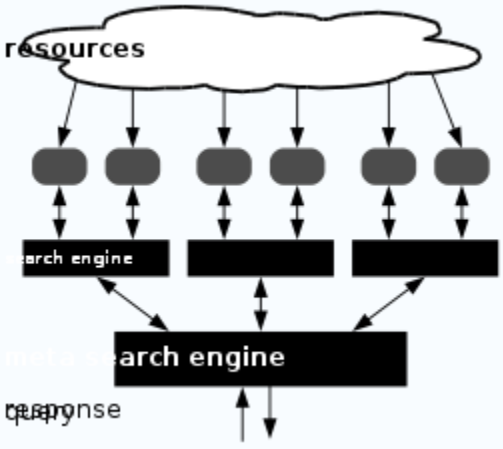
Ans: Metasearch engine

A **meta-search engine** is a search tool[1] that sends user requests to several other search engines and/or databases and aggregates the results into a single list or displays them according to their source. Metasearch engines enable users to enter search criteria once and access several search engines simultaneously. Metasearch engines operate on the premise that the Web is too large for any one search engine to index it all and that more comprehensive search results can be obtained by combining the results from several search engines. This also may save the user from having to use multiple search engines separately.

The term "metasearch" is frequently used to classify a set of commercial search engines, see the list of search engines, but is also used to describe the paradigm of searching multiple data sources in real time. The National Information Standards Organization (NISO) uses the terms Federated Search and Metasearch interchangeably to describe this web search paradigm.

- File
- File history
- File links
- Global file usage

Operation



architecture of a metasearch engine

Metasearch engines create what is known as a virtual database. They do not compile a physical database or catalogue of the web. Instead, they take a user's request, pass it to several other heterogeneous databases and then compile the results in a homogeneous manner based on a specific algorithm.

No two metasearch engines are alike. Some search only the most popular search engines while others also search lesser-known engines, newsgroups, and other databases. They also differ in how the results are presented and the quantity of engines that are used. Some will list results according to search engine or database. Others return results according to relevance, often concealing which search engine returned which results. This benefits the user by eliminating duplicate hits and grouping the most relevant ones at the top of the list.

Search engines frequently have different ways they expect requests submitted. For example, some search engines allow the usage of the word "AND" while others require "+" and others require only a space to combine words. The better metasearch engines try to synthesize requests appropriately when submitting them[*citation needed*].

Quality of results

Results can vary between metasearch engines based on a large number of variables. Still, even the most basic metasearch engine will allow more of the web to be searched at once than any one stand-alone search engine. On the other hand, the results are said[*who?*] to be less relevant, since a metasearch engine can't know the internal "alchemy" a search engine does on its result (a metasearch engine does not have any direct access to the search engines' database).

Metasearch engines are sometimes used in vertical search portals, and to search the deep web[

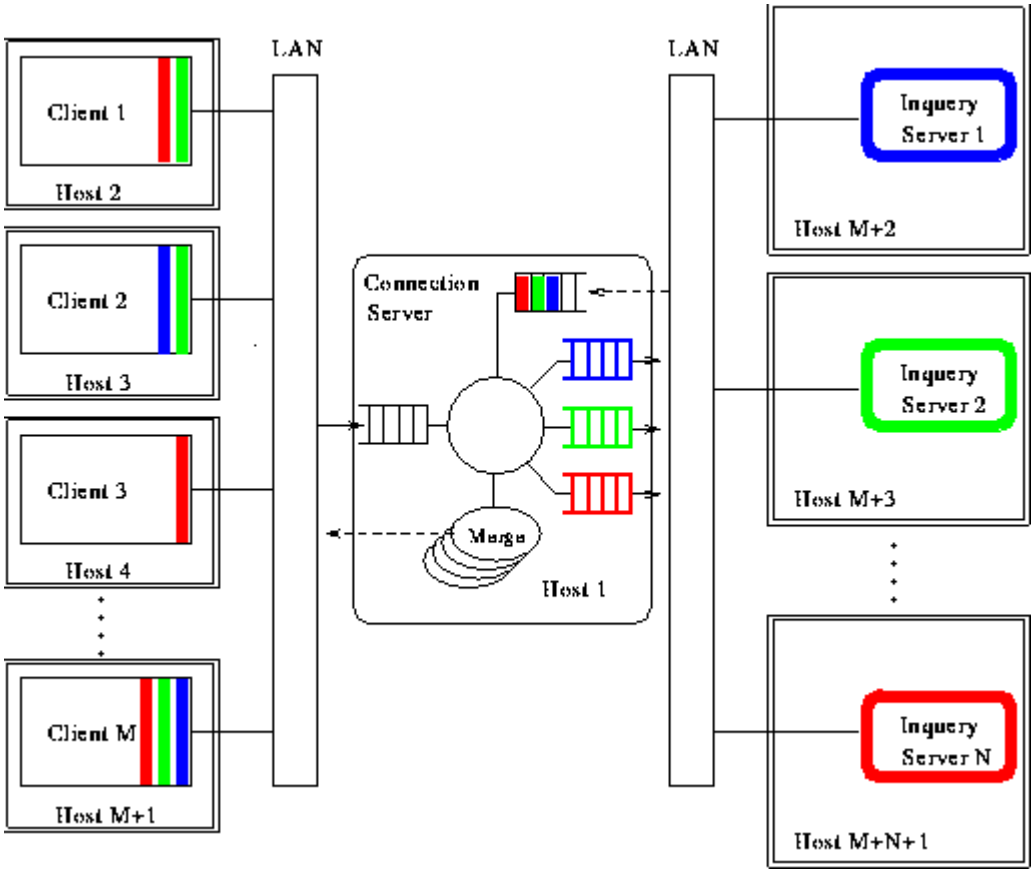
Q. Distributed & parallel IR?

Ans:

Distributed and Parallel Information Retrieval

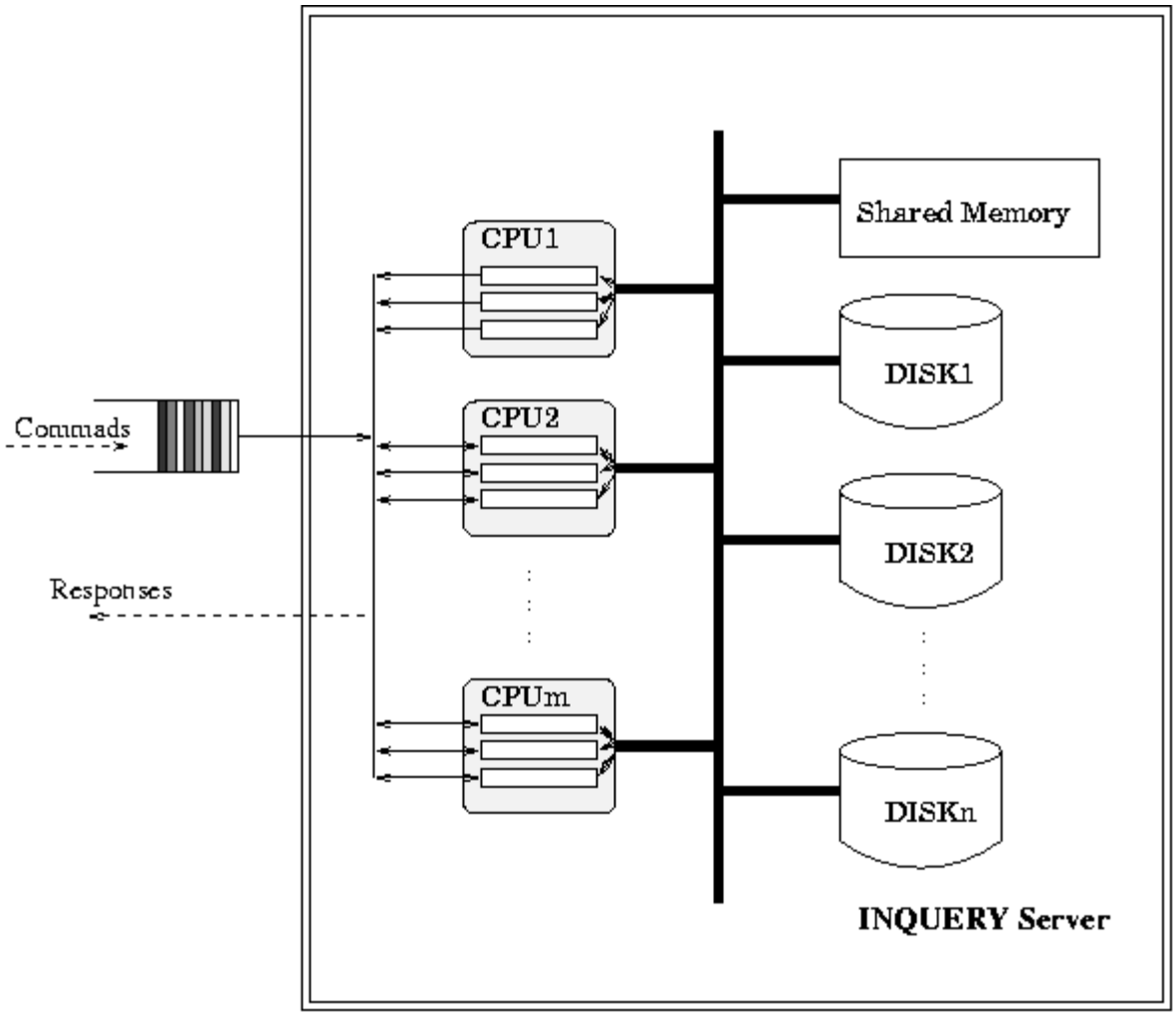
Providing timely access to text collections both locally and across the Internet is instrumental in making information retrieval (IR) systems truly useful. In order to users to effectively access these collections, IR systems must provide coordinated, concurrent, and distributed access. We investigate different architectures for distributed and parallel information retrieval in order to provide efficient and scalable IR services. In particular, we use partial collection replication.

Performance of Distributed Information Retrieval Architectures



We developed a prototype distributed information retrieval system which allows multiple users to search multiple document collections concurrently. The Distributed Information Retrieval Architecture consists of a set of clients (users) that connect to a set of IR systems via a connection server. The connection server is responsible for all messages sent between the clients and the IR systems. Since it is difficult to test the performance of large distributed systems, we created a simulation model of the distributed system in order to analyze the performance of the system under a variety of conditions. Furthermore, we are able to easily change the architecture and evaluate the effectiveness of the change. Our results show that our architecture is able to handle a large number of clients and text collections by using a surprisingly small number of connection servers.

Parallel Information Retrieval Servers

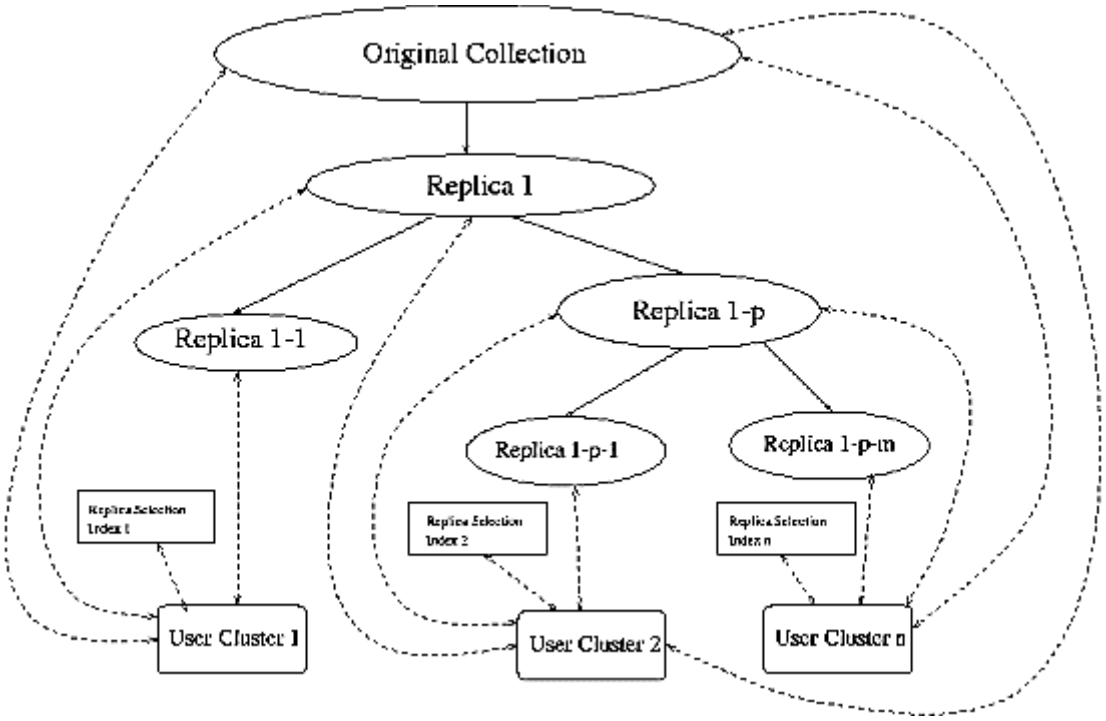


We have also developed a Parallel IR Server to investigate how best to exploit a symmetric multiprocessors when building high performance IR servers. Although it is possible to create systems with lots of CPU and disk resources, the important questions are *how much* of *which* hardware and *what* software structure is needed to effectively exploit hardware resources. In some cases, adding hardware *degrades* performance rather than improves it. We also compare the performance of a multi-threaded IR server to a multitasking IR server. As expected, the multi-threaded server is faster, but the multitasking server is competitive. Thus, for a large legacy system, it may not be worth the effort to convert a single threaded system into a multi-threaded system.

Replicating Information Retrieval Collections

Distributing excessive workloads and searching as little data as possible while maintaining acceptable retrieval accuracy are two ways to improve performance and scalability of a distributed IR system. Replication and partial replication of a collection serves these two purposes. Replicating collections on multiple servers can distribute excessive workloads posed on a single server; replicating collections on servers that are closer to their users can improve performance by reducing network traffic and minimizing network latency. We investigate how to build and rank partial replicas in a

information retrieval system. We build a hierarchy of replicas based on query frequency and available resources (the number of servers and their size).



Here we give an example to show how this architecture works. When a user in the user cluster 1 issues a query, the query goes to replica selection index 1 to determine relevance of `Replica 1-1`, `Replica 1`, and the original collection. If `Replica 1-1` is ranked as the top one, query may go to any of `Replica 1-1`, `Replica 1`, and the original collection for processing, based on server load and network traffic. If `Replica 1` is ranked as the top one, query may go to either `Replica 1` or the original collection. Otherwise the query only goes to the original collection.

Query Locality

In our work, we examine queries from real system logs and show that there is sufficient query locality in real systems to justify partial collection replication. We use 40 days of queries from THOMAS, the Library of Congress, and one day of Excite logs to study locality patterns in real systems. We find there is sufficient query locality that remains high over long periods of time which will enable partial replication to maintain effectiveness and significantly improve performance. For THOMAS, updating replicas hourly or even daily is unnecessary. However, we need to some mechanism to deal with bursty events. We propose two simple updating strategies that trigger updates based on events and performance, instead of regular updating. In our traces, query exact match misses many overlaps between queries with different terms that in fact return the same top documents, whereas partial replication with an effective replica selection function will find the similarities. We believe this trend will hold for other query sets against text collections and for web queries.

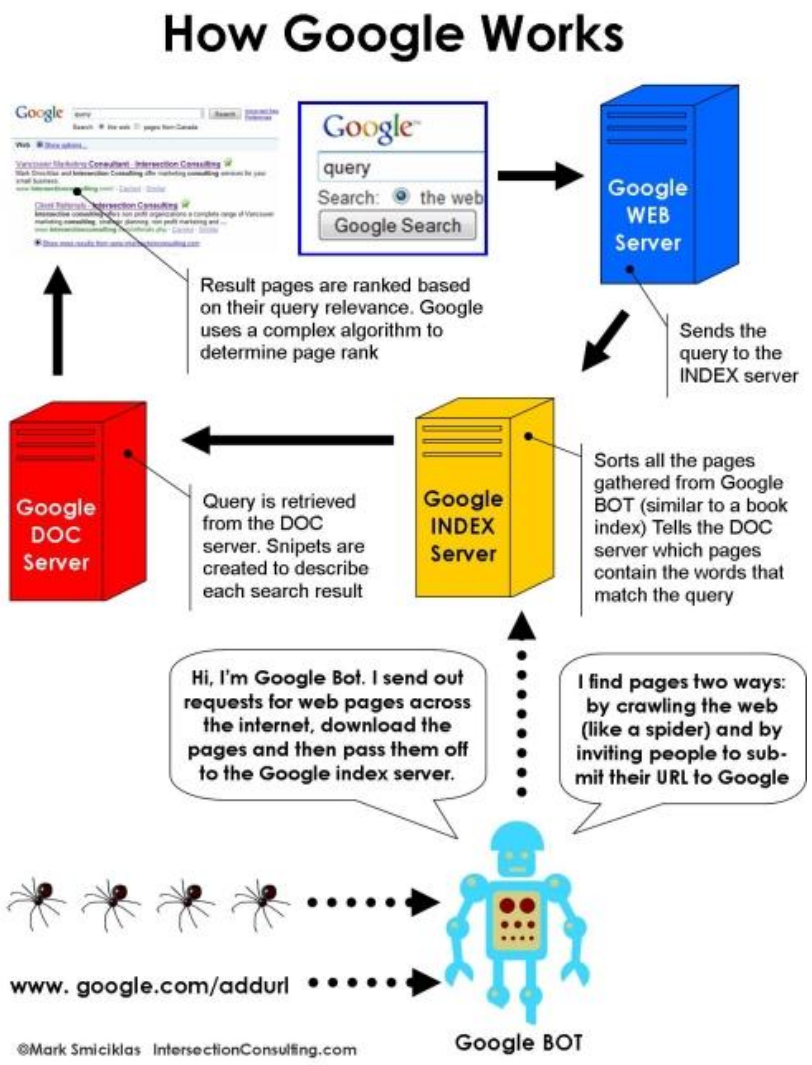
Finding the Appropriate Replica

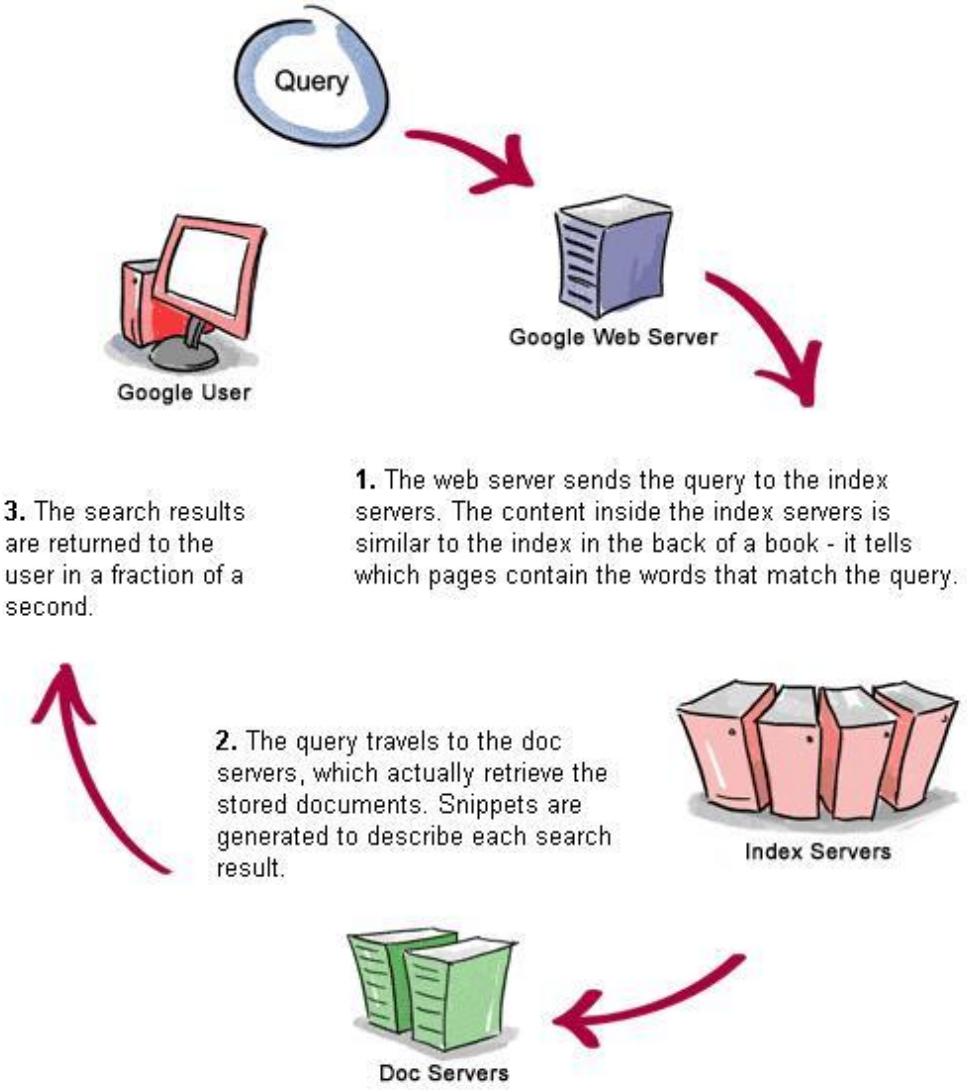
We extend the inference network model to rank and select partial replicas. We compare our new replica selection function to previous work on collection selection over a range of tuning parameters. For a given query, our replica selection algorithm correctly determines the most relevant of the replicas or original collection, and thus maintains the highest retrieval effectiveness while searching the least data as compared with the other ranking functions. Simulation results show that with load balancing, partial replication consistently improves performance over collection partitioning on multiple disks of a shared-memory multiprocessor and it requires only modest query locality.

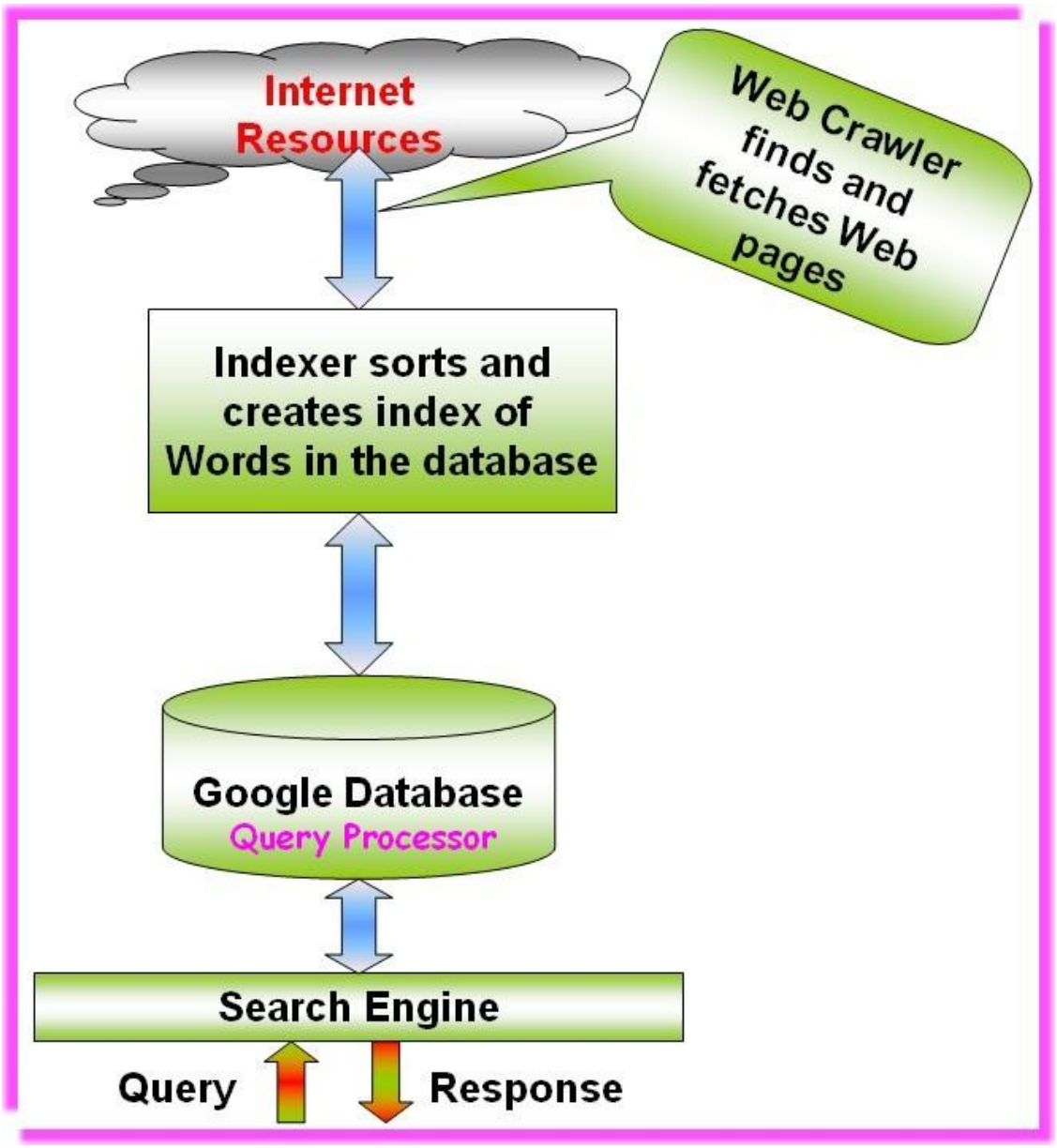
Searching a Terabyte of Text

We use our architecture to investigate how to search a terabyte of text using partial replication. We build a hierarchy of replicas based on query frequency and available resources, and use the InQuery retrieval system for the replicas and the original text database. We demonstrate the performance of our system searching a terabyte of text using a validated simulator. We compare the performance of partial replication with partitioning over additional servers. Our results show that partial replication is more effective at reducing execution times than partitioning on significantly fewer resources. For example, using 1 or 2 additional servers for replica(s) achieves similar or better performance than partitioning over 32 additional servers, even when the largest replica satisfies only 20% of commands. Higher query locality further widens the performance differences. Our work porting and validating InQuery and the simulator from a slower processor to the Alpha, as well as experiments with faster querying times which are reported elsewhere, lead us to believe the performance trends will hold for faster systems using fewer resources.

Q. How google works?


How Google Works
Google
query
Search: the web
Google Search
Google WEB Server
Result pages are ranked based on their query relevance. Google uses a complex algorithm to determine page rank
Sends the query to the INDEX server
Google DOC Server
Query is retrieved from the DOC server. Snipets are created to describe each search result
Google INDEX Server
Sorts all the pages gathered from Google BOT (similar to a book index) Tells the DOC server which pages contain the words that match the query
Hi, I'm Google Bot. I send out requests for web pages across the internet, download the pages and then pass them off to the Google index server.
I find pages two ways: by crawling the web (like a spider) and by inviting people to submit their URL to Google
www. google.com/addurl
Google BOT
©Mark Smiciklas IntersectionConsulting.com


Query
Google Web Server
Google User
1. The web server sends the query to the index servers. The content inside the index servers is similar to the index in the back of a book - it tells which pages contain the words that match the query.
3. The search results are returned to the user in a fraction of a second.
2. The query travels to the doc servers, which actually retrieve the stored documents. Snippets are generated to describe each search result.
Index Servers
Doc Servers


Internet Resources
Web Crawler finds and fetches Web pages
Indexer sorts and creates index of Words in the database
Google Database
Query Processor
Search Engine
Query
Response