Scalability Challenges in Web Search Engines

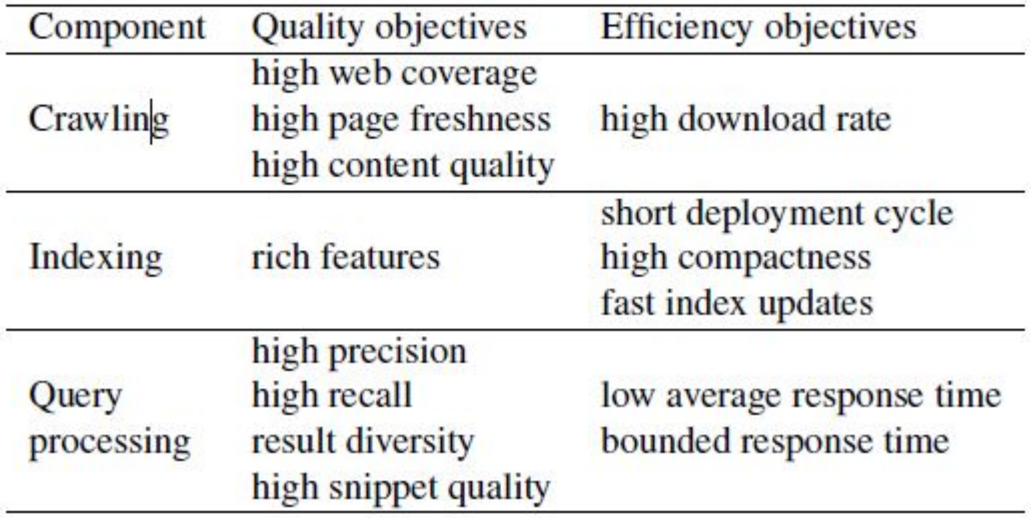
B. Cambazoglu and R. Baeza-Yates

Components of web search engine

1. **Crawler**: Locates the pages in the Web and download their content, which is stored on disk for further processing. This processing typically involves various parsing, extraction, and classification tasks. Eg. Yioop Classifiers for Spam, Homepage adding meta words
2. **Indexer**: Extracts the textual content from the stored pages and builds an inverted index for processing queries.
   - Also extracts additional features that will be used by query processor in relevance estimation. For eg. (position, score) tuple in Yioop
3. **Query Processor**: Evaluates user queries online over the constructed index. Presents small subset of results along with additional information like Title, small snippet, typically sorted by scores.

Efficiency Matrix

Table 1: The main quality and efficiency objectives in crawling, indexing, and query processing



Component	Quality objectives	Efficiency objectives
Crawling	high web coverage	high download rate
	high page freshness	
	high content quality	
Indexing	rich features	short deployment cycle
		high compactness
		fast index updates
Query processing	high precision	low average response time
	high recall	bounded response time
	result diversity	
	high snippet quality	

Efficiency parameters

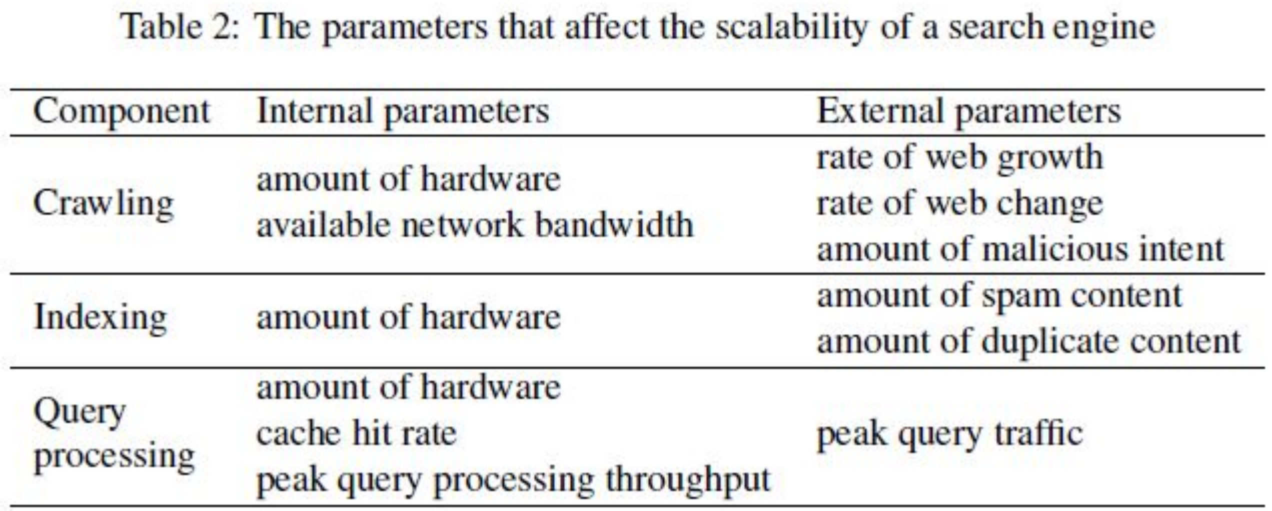


Table 2: The parameters that affect the scalability of a search engine

Component	Internal parameters	External parameters
Crawling	amount of hardware available network bandwidth	rate of web growth rate of web change amount of malicious intent
Indexing	amount of hardware	amount of spam content amount of duplicate content
Query processing	amount of hardware cache hit rate peak query processing throughput	peak query traffic

Objectives: Crawler

- High web coverage
- High page freshness
- High content quality
- Selectively refetches data to keep up-to-date version. Yioop supports this via ‘Allow Page Recrawl After’ config param.
- More important pages are downloaded more often to keep information up-to-date for important pages

Objective: Indexer

- Extract rich set of features from downloaded content. For eg. Yioop supports custom IndexingPlugins to extract additional features
- Common features are corpus statistics (term frequency, doc length) analysis matrices (click features, query log analysis, session analysis)
- Reduce index development cycles. (This is important because search engines searches using old indexes until new index is calculated)
- Speeding up index update
- Create compact index

Objective: Query processor

- Precision: fraction of relevant results in generated search result.
- Recall: fraction of relevant results returned in all relevant results that are available in index
- Achieve high precision and high recall
- High result diversity minimizes overlap of information from search results
- Providing descriptive summaries of document
- Most important efficiency metric is speed.
- Response time is query processing time plus latency.
- Response time should be bounded.

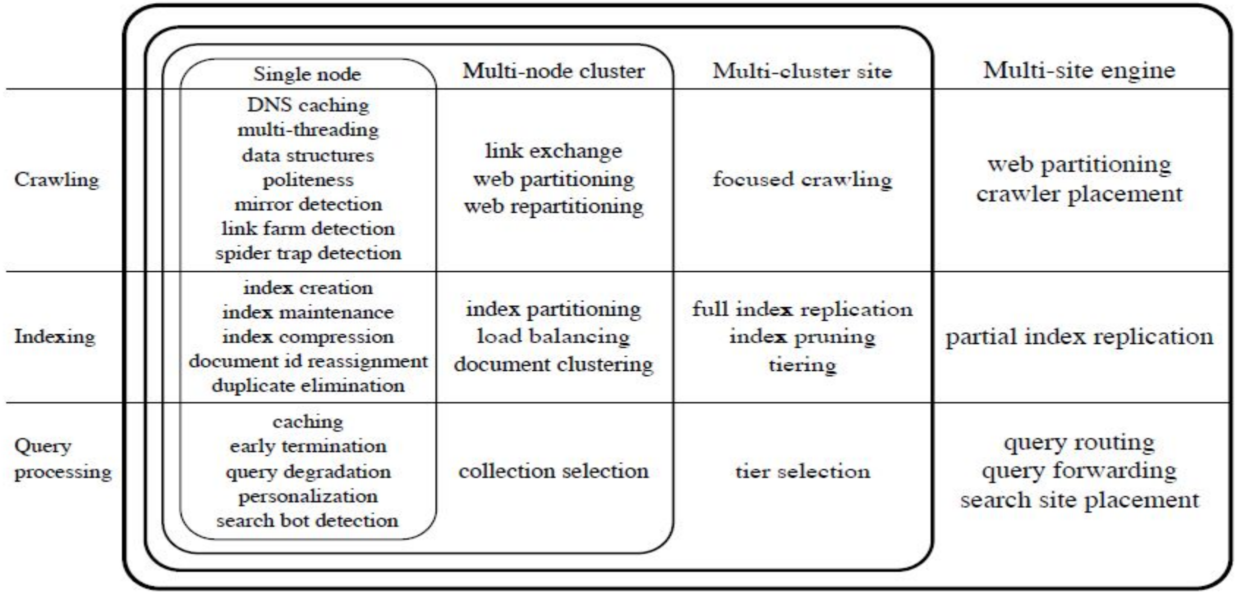
Parameters

- Internal parameters: Search engine has control over
- External parameters: Parameters that can not be controlled by search engine
- Internal parameters - hardware is an important parameter. Storage capacity greatly affects crawling, indexing and query processing.
- Network bandwidth affects speed of page download
- Cache is an important param which affects performance. Higher cache hits reduces query traffic reaching query processors.
- Peak query processing throughput is determined mostly by underlaying hardware

Parameters

- External parameters include - Web growth, Web change, and Malicious intent. For eg. Spider traps, link farms

Scalability Issues



	Single node	Multi-node cluster	Multi-cluster site	Multi-site engine
Crawling	DNS caching multi-threading data structures politeness mirror detection link farm detection spider trap detection	link exchange web partitioning web repartitioning	focused crawling	web partitioning crawler placement
Indexing	index creation index maintenance index compression document id reassignment duplicate elimination	index partitioning load balancing document clustering	full index replication index pruning tiering	partial index replication
Query processing	caching early termination query degradation personalization search bot detection	collection selection	tier selection	query routing query forwarding search site placement

Figure 2: Issues that have an impact on scalability.

Single Node Crawling

- Fetched all Seed URLs
- Downloads and stores pages in repository
- Parses content, extracts new links and add to the frontier (set of URLs yet to be downloaded). Typically a queue.
- URLs are hashed and stored in hash tables to check existence.
- Crawler maintains robot.txt and cache of DNS entries of hosts
- DNS caching reduces dns lookup overhead
- Utilizes multi-threading for downloading as network is usually not bottleneck with this architecture.
- Disk access and memory access may form bottleneck hence efficient data structures are used.
- Download priority in frontier is important
  - Earlier approaches used quality matrics (eg. linkage between pages) and recrawl based in likelihood of modification
  - Recent approaches uses impact based crawling which prioritize pages which are expected to occur more frequently in search results. Websites which contributes more pages are prioritized for early crawling and recrawling.

Single Node Indexing

- Sequentially creates index from each document.
- Parses html, parses links, soft 404 detection, text classification, entity recognition, and feature extraction.
- Document is represented by terms appearing in it and also by additional terms.
- Cleans document and creates inverted index
- Inverted index stores terms in vocabulary and postings. Posting stores document id and the frequency of the term in the document
- Position information is needed to computing relevance based on proximity of query terms in documents.
- Also position where it occurs like in title, header, body or footer.

Single Node Indexing

- To keep index fresh, modification in the document should be reflected in the index.
- Simplest option is to rebuild complete index
- Another option is to incrementally update index as new documents arrive. Updates can be accumulated in memory and merged periodically.
- For efficiency reasons, inverted index is tried to keep in memory in compressed form such that decompression overhead is minimal.
- Indexers also tries to remove exact and near duplicates documents. Hashing can be used to remove exact duplicates. Near duplicate deletion is more costly and inaccurate.

Single Node Query Processing

- Query cleaning like case-folding, stemming, stopword elimination
- Complex operation like spelling correction, phrase recognition, entity recognition, localization, query intent analysis.
- internal representation of query is looked up in cache to determine if related search results are already computed for previous submission of the query.
- Higher cache hits results better performance.
- Static-dynamic combined caches are generally used.
- Recent works are mostly focused on admission, eviction, and prefetching policies to increase hit rates.

Single Node query processing

- Some works predicted processing cost of query while making decision on admission and eviction.
- Cache freshness has been argued as most important issue with result caching.
- To achieve freshness, caches are invalidated when index is updated because of change in document
- Result cache are kept up to date using time-to-live mechanism and proactively refreshing cache when backend system is not busy.

Single Node query processing

- Queries whose results are not found in cache, speed up applied using various kind of optimizations.
- Posting list caching is done to keep most frequently or recently accessed posting list is tried to keep in main memory
- Another important optimization is early termination
- Query degradation happens when traffic is high and queries are terminated early. This results in low quality search results.

Multi node crawling

- In practise, single node crawlers can be used in embarrassingly parallel mode and crawl web independently.
- May result in redundancy because of duplicate crawls.
- Simple fix is to partition web into different crawlers (For eg. use hashing to check if page belongs to current crawler).
- Naive way is to ignore pages which does not belong to current crawler which decreases the coverage.
- Other way is to delay download until all local downloads are completed.
- Best way is to communicate the responsible crawler about the page

Multi node crawling

- Best way to partition web is to assign complete website to a single crawler than individual page.
- This increases politeness as number connections on websites is reduced.

Multi node indexing

- Indexes are partitioned into number of disjoint subindexes.
- Inverted indexes can be partitioned based on documents or terms.
- In document based partitioning subindexes contain postings of disjoint sets of documents.
- Also known as local index partitioning as subindex can be built locally by separate indexing nodes.
- Assignment is done creating map between hashes of DocId and Indexing Node id.
- It requires collecting and replicating from all nodes the global collection statistics.

Multi node indexing

- Term based is known as global index partitioning
- Each part contains postings of disjoints sets of terms
- Due to large variation in posting lists sizes and their accessing frequencies main goal is to achieve reasonable load balance during parallel query processing.
- Both term- and document- indexes can be constructed by using parallel indexing system.
- These methods are non-topical partitioning approaches.
- Topical partitioning can also be done by clustering similar types of documents. For eg K-means can be used for clustering

Multi node query processing

- Depends on the way index is partitioned
- In document based partition, broker issues queries to all nodes. Results are concurrently processed and returned to broker.
- Global collection statistics is available to all nodes. Hence, relevance score assigned are compatible
- Merging of result is trivial operation at broker.
- In case of term frequency, query is issued to only search node that contains posting lists associated to query terms.
- Contacted node computes result sets, where document scores are partial, using posting lists related to query.
- Result set is transferred to broker which merges it with global resultset.
- In topical document clustering, collection selection techniques are used to determine nodes to process query.

Reference

[1] B. Cambazoglu and R. Baeza-Yates, “Scalability Challenges in Web Search Engines,” in *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 7, 2011, pp. 27–50. doi: 10.1007/978-3-642-20946-8_2.