Scalability Challenges in Web Search Engines

Major components of web search engines

- Crawler: locate and download webpages for further processing
- Indexer: extracts text and features for relevance estimation from pages and builds inverted index
- Query processor: receives online queries, evaluates them over index, presents top results

Component Objectives

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

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

Parameters affecting Scalability

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

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 System
  - single computer is dedicated to each component
  - web is crawled by a single computer, usually through multithreading
  - common doc repository
  - Crawling:
    - How it works:
      - seed URLs fetched by establishing HTTP connections with web page servers
      - downloaded pages stored in repository
      - downloaded pages are (simultaneously on storing) parsed and extracted links are added to frontier (queue) to be downloaded
      - hash values of discovered URLs are stored in hash table to check for duplicate links
      - encountered robots.txt and cache for DNS host entries (for DNS resolution) are maintained
    - Takeaways:
      - threads handle each download request separately
      - number of threads can be increased until bandwidth limit
      - issues:
        - main bottlenecks: disk access / memory (need efficient data structures)
        - download order of URLs (by prioritization algorithms)
        - malicious intent of website owners: such as delay attacks (cause unnecessary delays in responses), spider traps can degrade system resource utilization, link farms can harm crawlers since a crawler may allocate significant portion of its resources to download these spam pages

- Indexing:
  - How it works:
    - pass each doc in crawled collection through preprocessing pipeline: HTML parsing, link/text extraction, filtering spam, feature extraction, etc
    - each doc is then represented by specific terms, which indexer converts into inverted index
    - try to keep index compressed in main memory for efficiency
    - while processing queries, relevant portions are decompressed and used
  - Takeaways:
    - aims for most compact index with least compression overhead
    - eliminate exact (accurate, easy) / near (inaccurate, tough and costly) duplicate documents: reduces index size and saves computing power

- Query Processing:
  - How it works:
    - user's queries are preprocessed: stemming and stopword elimination, spell correction, localization, etc
    - transformed into internal representation
    - looked up in result cache: for hit, cache immediately serves results; for miss, query processed over index
  - Takeaways:
    - main issue with cache results: freshness
      - invalidate cache entries if index is updated for modifications in doc collection
      - try to keep most frequently/recently accessed posting lists in main memory to speed up query processing
    - query degradation:
      - if query processing time exceeds allowed budget, computation is terminated and search results computed until then are returned
      - in heavy traffic (query count exceeds processor capacity), queries are handled in degradation mode (partially computed search results

- Multi-Node Cluster
  - parallelized execution of components
  - increased page download rates due to concurrent crawling
  - faster index creation and deployment due to many available computers
  - query response times reduced due to query processing being parallelized over multiple computers
  - user query is issued to broker node, which dispatches it to search nodes
  - broker node also merges search nodes' results and returns them
  - Crawling:
    - partition web across multiple parallel crawlers based on their URL hashes
    - coordination issues: each crawler has local data structures so cannot check if a link has already been fetched, crawling order cannot be optimized globally
      - naive solution: simply ignore newly discovered link if current node is not responsible for fetching it: reduces coverage
      - delay download of non-local links until all local links are exhausted: full coverage, but redundant crawling possible
      - communicate discovered non-local links to nodes that are responsible for fetching them: most efficient, but significant communication overhead
        - more efficient: communicate in batches of links

- Indexing:
  - created index will be stored in multiple disjoint subindexes
    - each subindex acts as a separate entity that can be individually queried
    - document-based partitioning: each subindex contains postings of a disjoint set of documents
    - term-based partitioning: each part contains a set of posting lists associated with a disjoint set of terms
    - docs can also be clustered by topic: each cluster is assigned to a separate node

- Query Processing:
  - document-based partitioning:
    - broker issues query to search nodes
    - top k results are computed parallely and returned to broker
    - merging is a trivial operation since global collection stats are available to all nodes and so relevance scores are compatible
    - better load balancing, lower processing times, better fault tolerance
  - term-based partitioning:
    - query issued only to nodes containing posting lists for those terms
    - partial doc scores are computed on nodes and then merged into global resultset by broker
    - lower performance, higher query throughput
  - topical partitioning:
    - collection selection techniques determine which nodes will process query
    - collection selection: broker maintains summaries of collections in each search node and issues query selectively, according to relevance of collections to query

- Multi-Cluster Site
  - multiple clusters are built within a search site to run many instances of the three components
  - each crawling cluster is specialized in fetching a certain type of content (e.g., news pages)
  - for every crawled collection, an indexing cluster builds a corresponding index served by separate search clusters (each has its own broker)
  - federator blends results from multiple brokers and returns final result
  - Crawling:
    - usually, one large cluster crawling entire web and small-scale clusters that run focused web crawlers to harvest web selectively according to a selected theme (e.g., news pages, blogs, images, academic papers)
  - Indexing:
    - creates replicas of index on multiple clusters
    - index pruning:
      - objective is to create a small web index containing postings of docs that are more likely to appear in future search results and process queries only over this index
    - tiering:
      - index is disjointly partitioned into multiple tiers (subindexes) based on quality of documents on separate cluster
      - tiers are ordered in increasing order of average document quality and hit by queries in this order
  - Query Processing:
    - constructing multiple copies of same query processing cluster and replicating same index on these clusters increases query processing throughput
    - for pruned index: queries first processed on pruned index then full index
    - for tiered index: query is sequentially processed over tiers, in increasing quality order of tiers

- Multi-Site Engine
  - most sophisticated architecture: distributes all three components over multiple, geographically distant sites
  - Crawling: sites crawl web pages in geographical neighborhood
  - Indexing: disjoint indexes are built from local doc collections by language-based or region-based partitioning
  - Query Processing: queries are first processed on local sites, then forwarded to non-local sites for further evaluation, and all results are merged by initial local site

Open Issues

- Crawling
  - effective web partitioning: come up with techniques to accurately identify locations of web sites and map them to closest crawling sites
- Indexing
  - tiering: more clarity needed on how to optimally select number and sizes of tiers, and how to place docs in tiers
- Query Processing
  - freshness: identify and refresh stale cache entries before query requests without major computational overhead

Reference

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.