# Yioop Ranking Mechanisms

# Crawling Intro

- Meta-keywords are added to queries for more info eg. Language, safe search, etc.
- Document scoring in Yioop:
  - Doc Rank:  doc importance as a whole
  - Relevance: importance of search words to doc
  - Proximity (2+ non-meta keywords): how frequently search words appear close to each other
- Assumption:  docs indexed by importance (by DR)
- Crawl processes:
  - Name server: coordinator, starts/stops crawls
  - Queue servers: each holds a priority queue of what to download next
  - Fetchers: page (from queue server) download and processing
- urls are assigned to QS based on their hostname hashes
- Fetchers pick up a schedule of DOWNLOAD_SIZE_INTERVAL urls from QS to download at a time

# Fetchers (impact on Search ranking)

- Batches of 100 urls
- Request PAGE_RANGE_REQUEST num bytes from each url
- Process (per batch):
  - Find mimetype and choose page processor (+ scraper for HTML pages)
  - Page processor extracts doc summary
  - Indexing plugins for page processor to generate aux summary/modify extracted summary
  - Run classifiers on the summary and add any class labels and rank scores
  - Calculate hash from downloaded page minus tags/non-word characters for deduplication
  - Prune no. of links extracted from the document down to MAX_LINKS_PER_PAGE (def: 50)
  - Apply any user-defined page rules to the summary extracted
  - Retain summaries (full caches of pages if configured) in fetcher mem until either schedule is fully downloaded or hit SEEN_URLS_BEFORE_UPDATE_SCHEDULER: in that case, ship info off to appropriate queue server

- HTTP headers are used to determine mimetype, which determines page processor
- Page processor extracts:
  - Language
    - Stemmer applied based on lang
  - Title
  - Description
    - Split text into sentences, assign score, concatenate top sentences in the original order
    - Scores used later while computing importance of term to doc
  - Links
    - Used to find new pages for download, "mini-docs"
    - <= 300 links per doc, link text used as description
  - Robot Metas
- After processing, pruneLinks used to return the top 50 links
- Links are treated as separate docs
- Hostname might not match that of the queue server of current schedule
- Fetcher partitions link docs based on which queue server handles that hostname and returns info to appropriate server when it is req its schedule
- If memory is low, info is returned to appropriate queue server earlier

# Queue servers (impact on Search ranking)

- Fetcher writes data to QS web-app, web-app writes:
  - Urls to crawl in ScheduleData
  - robot.txt data in RobotData
  - Mini inverted-index/summary data in IndexData
- QS processes:
  - Indexer: adds IndexData to active partition, periodically runs DictionaryUpdater
  - DictionaryUpdater: builds inverted-index out of full partition and adds to overall index
  - Scheduler: priority queue holds urls to be downloaded next, reads ScheduleData
- Indexer saves IndexData to active partition of IndexDocBundle
- IndexDocBundle:
  - Documents:
    - PartitionDocBundle folder
    - Contains partitions (file pair):
      - .txt.gz: compressed doc_summary, doc_objects
      - .ix: record format (doc_id, offset in .txt.gz to summary, offset in .txt.gz to doc, len(doc _object))

- Pos_doc_map
  - One folder per partition:
    - Doc_map (doc_id -> {pos, score})
      - Urls with same hash value are grouped and one representative doc_id (webpage) is used
      - First pair: doc offset in .txt.gz, overall score
      - Rest: term pos in doc, score for terms between prev pos and current pos
      - Last: scores for doc wrt classifiers
    - Positions: for each partition's worth of docs, store locations of term in every doc it appears in
    - Postings:
      - One for partition new doc will be added to, one for others
      - Inverted_index for partition (term_id -> posting_list_term)
        - Posting_list_term: doc_index in doc_map, term_freq, offset of terms position list in positions file, len(positions file entry)
    - Last_entries:
      - Record keeping for each term to output postings correctly
      - (term_id, last_doc_index, last_offset, num_occ)
      - Postings stores doc_index and term offsets in posting list as difference from previous value (delta list format), last_entries keeps track of the original/non-delta values for easy computation of approximate value for next posting list to be added
- Dictionary: B+ tree where each node: (term_id -> posting_list)
- Next_partition
- Archive_info

# Constructing doc_map

- Doc importance measured by (partition, doc_map_index)
- Two types of grouping to create doc_map file:
  - By url hash (selected doc assigned sum of scores)
  - By text hash (doc with max score chosen as representative: selected doc assigned sum of scores)

# Doc Rank Score

- Find num of docs after doc A: sum over the number of documents in partition after A's partition + A' number of document after its doc_map_index in its partition.

$$\text{DOC\_RANK(A)} = \log_{10} \text{number\_of\_document\_after(A)}$$

- (Considering max 1 billion docs) DR <= 10

# Crawling

- Previously: best first search, OPIC
- Yioop uses "Host Budgeting", inspired by IRLBot
- Fetcher writes urls to /ScheduleData/ -> Scheduler picks up oldest timestamp and sorts into cache/QueueBundle subfolders:
  - UrlQueue: robot.txt downloaded
  - WaitRobotUrl: still waiting for robot.txt to be downloaded
  - CrawlDelayedHosts
- BloomFilter ensures urls aren't added to QueueBundle multiple times
- Find UrlQueue tier:
  - CLDData linear hash table:
    - SEEN_URLS: raw count of urls for a CLD
    - WEIGHTED_SEEN_URLS: less important urls add more weight
    - WEIGHTED_INCOMING_URLS: adjusted for incoming links from different CLD

- Takeaways:
  - CLD with more incoming good links has more pages in lower tiers
  - Higher the tier, more the urls waiting to be scheduled
- Scheduler picks urls by round robin, order in which urls entered tier
- Wrinkles to Crawling Process:
  - Crawl delay:
    - Robot.txt can indicate crawl delay/Yioop decides to induce delay for overloaded sites
    - Url needs to be spaced in schedule at least one batch (100 urls) from url of same host
    - If current schedule is full, url moved to CrawlDelayedHosts before requeued to UrlQueue
  - Yioop allows recrawl based on Etag and Expires url header to accommodate changes in page

- Takeaways:
  - Two fetchers can get consecutive schedules from same scheduler and return data to Indexer out of order
  - Rely on query time manipulation to try and improve accuracy

# Search Time Ranking Factors

- Incoming query modifications:
  - Control words calculated
  - Guess semantics
  - Stemming/char n-gramming, rewrite abbreviations/acronyms
- Iterator built from resultant terms to fetch summaries/links with all terms
- Single QS:
  - One iterator per term
  - Intersect iterator returns common docs with all terms, timeout added
  - Grouping iterator groups links/summaries/docs with same hashes from diff partitions
  - Docs scored, sorted, top 10 returned
- Multiple QS:
  - Network iterator
  - Multiply n (expected num of results) by alpha, divide by num of QS, get resultant from all intersect iterators
  - Group on name server

# Scoring Docs

- Bonus scores for meeting certain criteria
- Rel: divergence from randomness
- Prox:  tags
- Final doc score = DR + bonus + rel + prox