System Support and Design Issues in Query Processing

•Tao Yang 293S, 2020

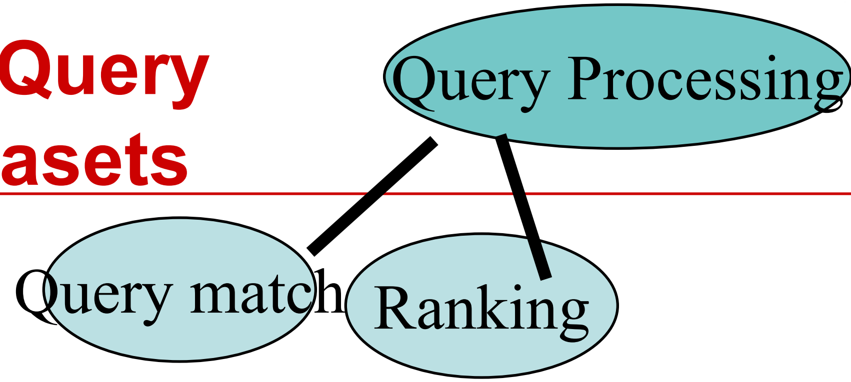
Content

- System support and design tradeoffs in online systems and query processing
  - Objective: fast response, high throughput, and high availability
- Experience with Ask.com online architecture

Online Data for Search: Inverted Index and Auxiliary Structures

- **Inverted lists usually stored together in a single file for efficiency**
  - Term statistics stored at start of inverted lists
- ***Vocabulary* or *lexicon***
  - Contains a lookup table from index terms to the byte offset of the inverted list in the inverted file
  - Either hash table in memory, key-value stores, or B-tree for larger vocabularies
- **Document-oriented information**
  - E.g. Document quality score, freshness indicator, page text content
  - In-memory hashtable, key-value stores
- **Other information**
  - Collection statistics. Web host information

Design Consideration of Query Processing for Large Datasets



- Go through all postings of queries words
- Conduct matching & ranking

- **Estimate I/O cost**
- **Memory cache for storing frequently accessed items**
  - Cache size requirement: Is there enough memory?
  - Does program exhibit cache locality?
- **Distribute data to multiple machines for parallel processing**
  - Distribute disk data to p machines evenly
  - Distributed memory data to p machines evenly

System Challenges for Online Services

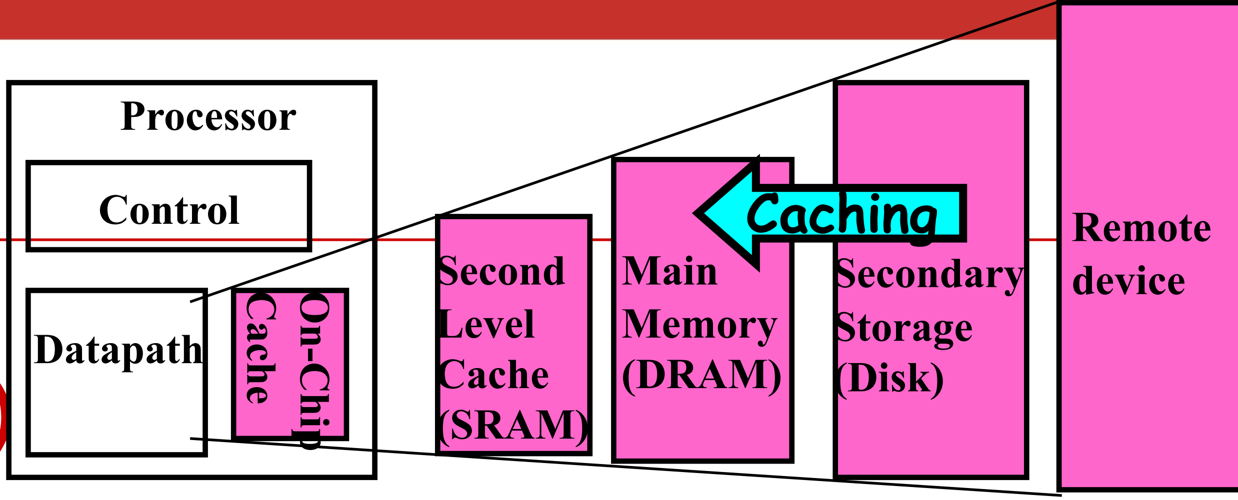
- **Challenges/requirements for online services:**
  - Each system needs tens or hundreds of subservices, running on hundreds or thousands of machines if not more.
  - Low response time, high throughputs
  - Data intensive, need to consider impact of cache, memory, disk I/O
  - Huge amount of data, requiring
    - Large-scale clusters.
    - Incremental scalability.
  - 7×24 availability with fault tolerance:
    - Operation errors, Software bugs, Hardware failures
  - Resource management, QoS for load spikes.
- Careful **design planning** in architecture and system support choices for reliable/scalable online services

Response Time vs Concurrency for Search Query Processing

- Backend response time requirement: ~200 ms per request
- Throughput requirement: number of requests second per machine
  - 100 Requests/second per machines
  - → 10 machines 1000 requests →86 million requests/day
- Rules of thumb
  - Writes are expensive. Reads are cheap (Search engine does read most of time)
  - Access HDD is expensive and a few are allowed per query. Access SSD is better. More are allowed per query
  - Minimize disk I/O by combing small I/O accesses
- Distributed processing/parallel processing is feasible
  - But watch cost of network communication/latency

Jeff Dean

Numbers Every Engineer Should Know (Approximately)



$ms=10^{-3}s$
$\mu s=10^{-6}s$
$ns=10^{-9}s$

CPU

- L1 cache reference 0.5 ns
- L2 cache reference 7 ns

Memory

- Memory Main memory reference 100 ns
- Read 1 MB sequentially from memory 0.25ms

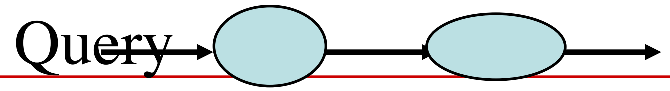
Disk

- HDD Disk seek 8ms while SSD takes 0.1ms for reading 4KB
- Read 1 MB sequentially from disk 10ms-1ms

Network

- Round trip within same datacenter 0.5ms
  - The part of transferring 1K bytes over 1 Gbps network 10μs
- Read 1 MB sequentially from network 10ms
- Send packet CA->Europe->CA 150ms

Modeling of Response Time for Query Processing



Components of time cost

Query response time ≈

+#instruction*CPUCost

+#MemoryAccess*MemoryCost

+#NetworkOPs * NetworkCost

+ #IO-OPs*IOCost

NetworkCost = Startup latency + DataSize/TransferRate

IOCost = Startup latency+ DataSize/TransferRate

What do we learn from these numbers?

Bad for each query

- 1 cache reference 0.5 ns
- L2 cache reference 7 ns
- Memory Main memory reference 100 ns
- Read 1 MB sequentially from memory 0.25ms

Poor L1/L2 locality for compute-intensive core

Scan 1000MB list

- HDD Disk seek 8ms while SSD takes 0.1ms for reading 4KB
- Read 1 MB sequentially from disk 10ms-1ms

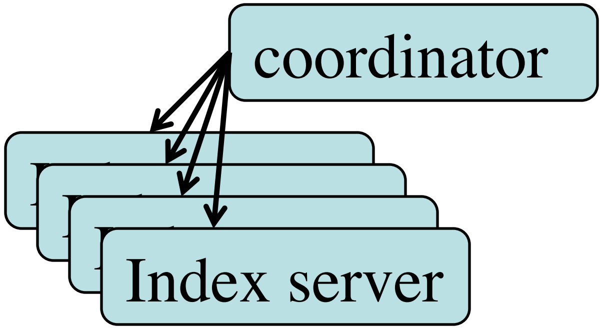
Access disk 10,000 times

- Round trip within same datacenter 0.5ms
  - Transmitting 1K bytes over 1 Gbps network 10μs
- Read 1 MB sequentially from network 10ms
- Send packet CA->Europe->CA 150ms

Remote hash table lookup for 5,000 times

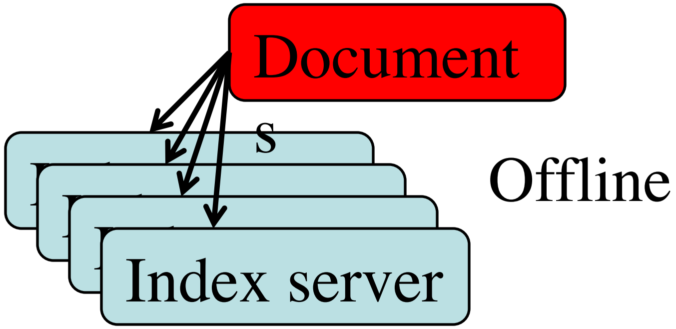
Parallelism Management in a Cluster of Machines for Search

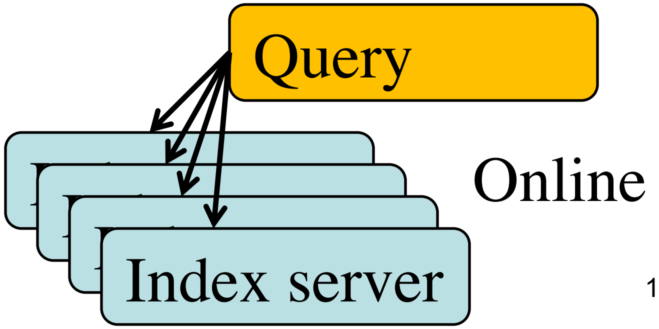
- **Basic steps for parallel processing**
  - All queries sent to a *coordination machine*
  - The coordinator then sends messages to many *index servers*
  - Each index server does some portion of the query processing
  - The coordinator organizes the results and returns them to the user
- **Two main approaches**
  - Document distribution
    - by far the most popular
  - Term distribution



Document-based distribution

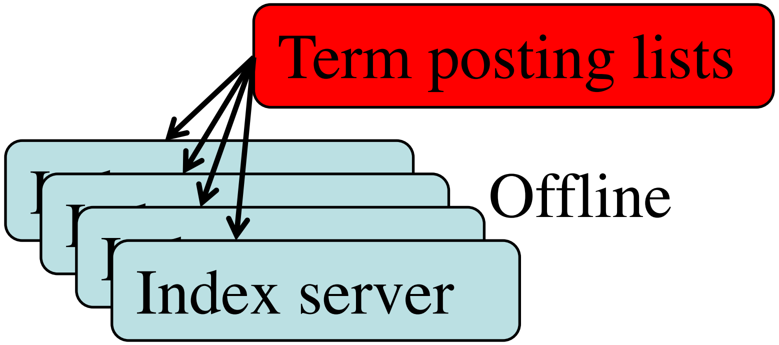
- **Document distribution**
  - Each index server acts as a search engine for a small fraction of the total collection
  - A coordinator sends a copy of the query to each of the index servers, each of which returns the top-*k* results
  - Results are merged into a single ranked list by the coordinator

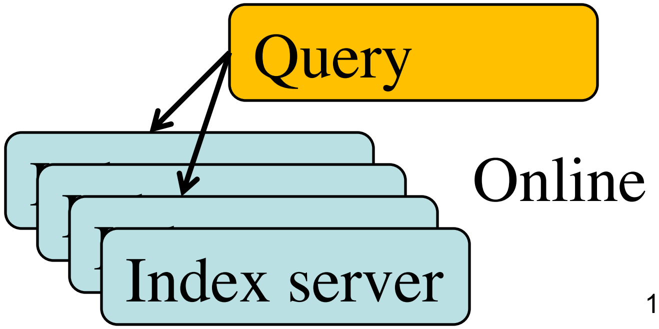




Term-based distribution

- **Single index is built for the entire cluster**
- **Each posting list of a term is assigned to one index server**
- **During query processing,**
  - One of the index servers is chosen to process the query
  - Usually the one holding the longest inverted list
  - Other index servers send information to that server
  - Final results sent to director





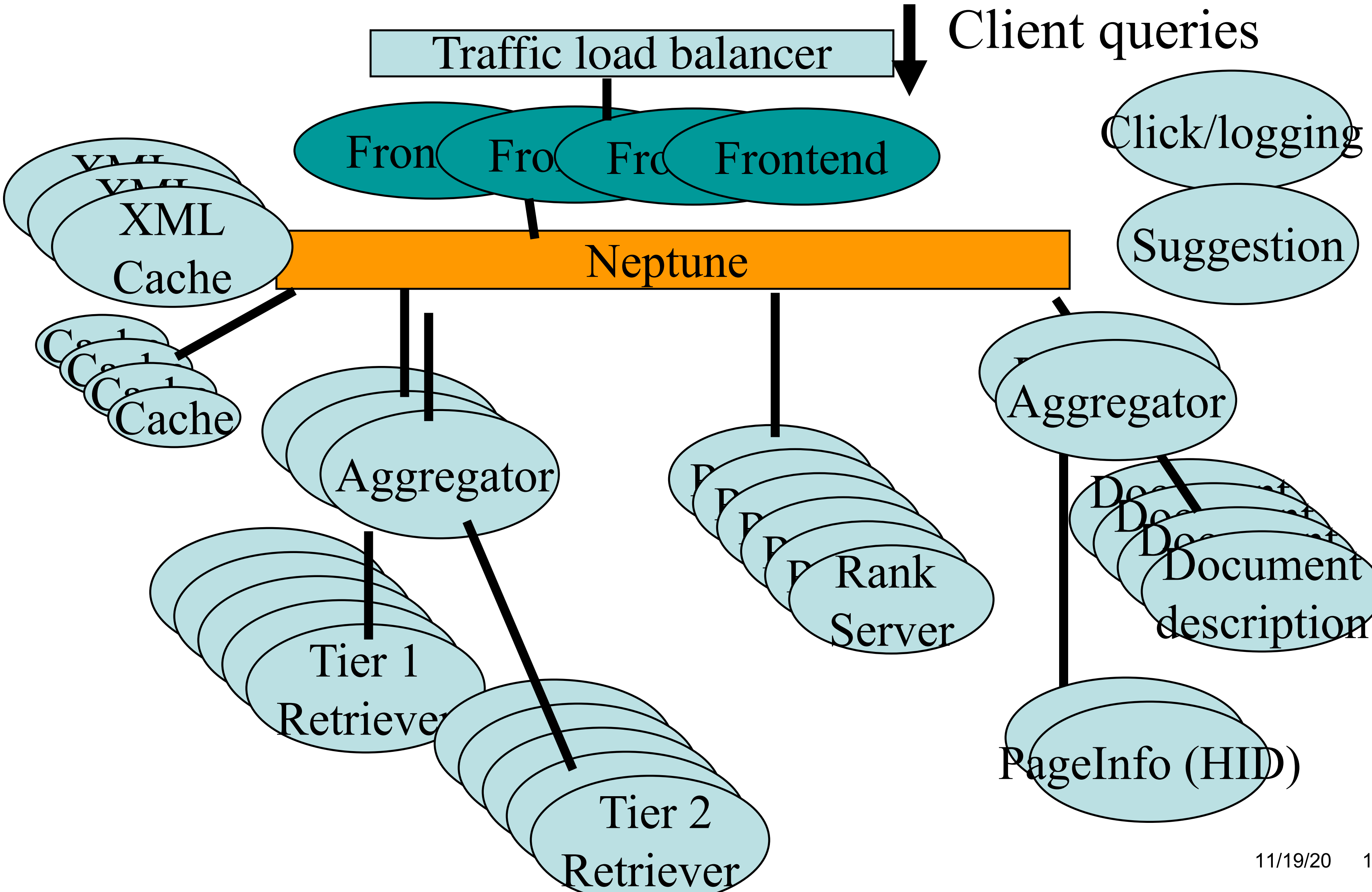
Layout of inverted index impacted by online algorithms

- **Early termination of faster query processing**
  - Ignore lower priority documents at end of lists
  - Fast (but unsafe) optimization
- **Ordering of inverted posting lists**
  - Impact sorted index: high score documents first
  - Document sorted index: increasing order of doc IDs
  - How to combine the advantages?

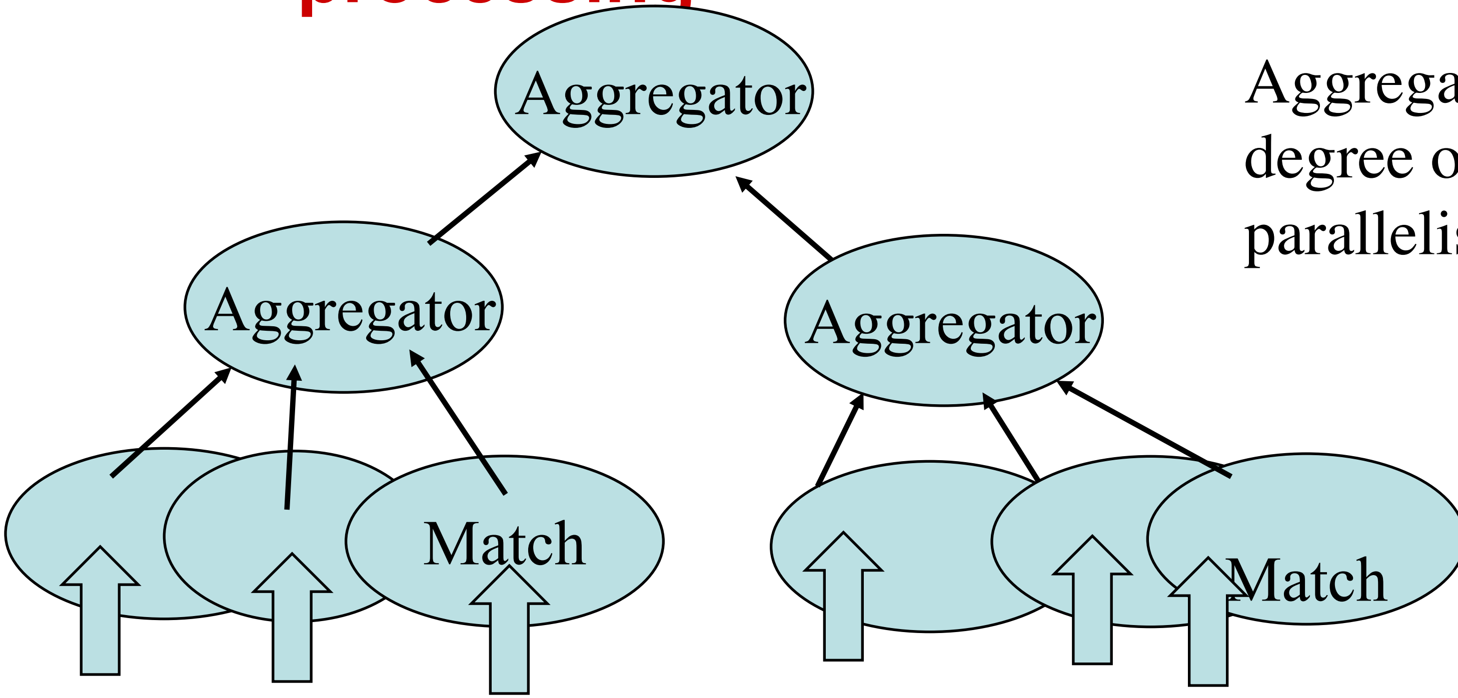
Sort layers by impact, and then sort documents by IDs within each group

Term	Sort by IDs	Sort by IDs	Sort by IDs
	Impact layer 1	Impact layer 2	Impact layer 3

Ask.com Search Engine

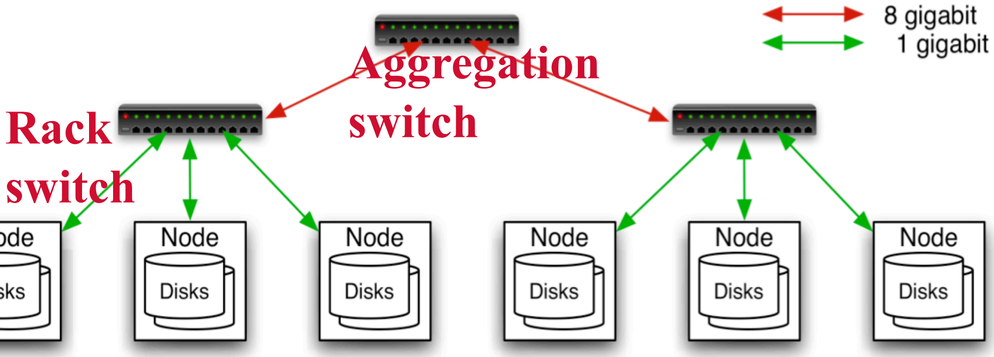


Multi-tier aggregation for query stream processing

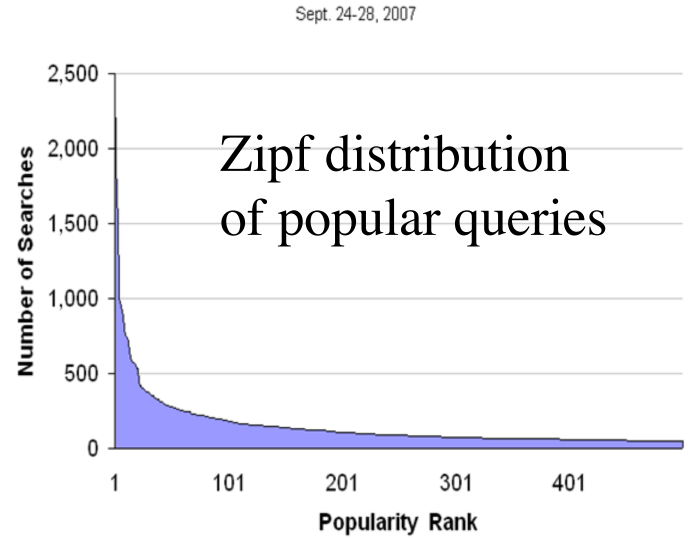


Aggregator controls degree of fan-out parallelism

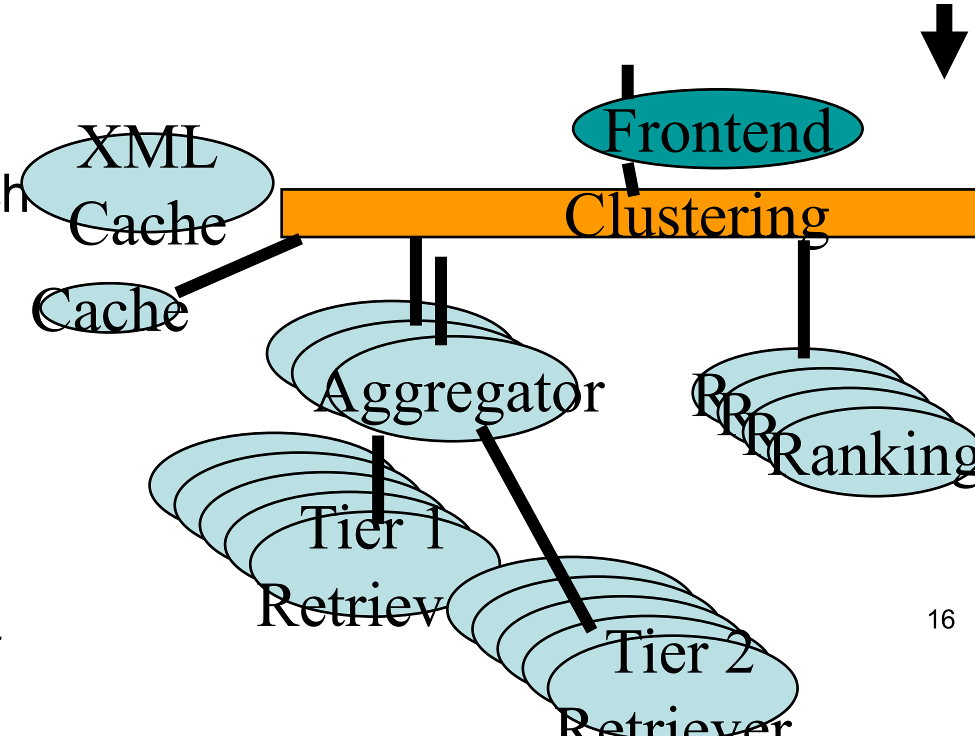
Cluster architecture



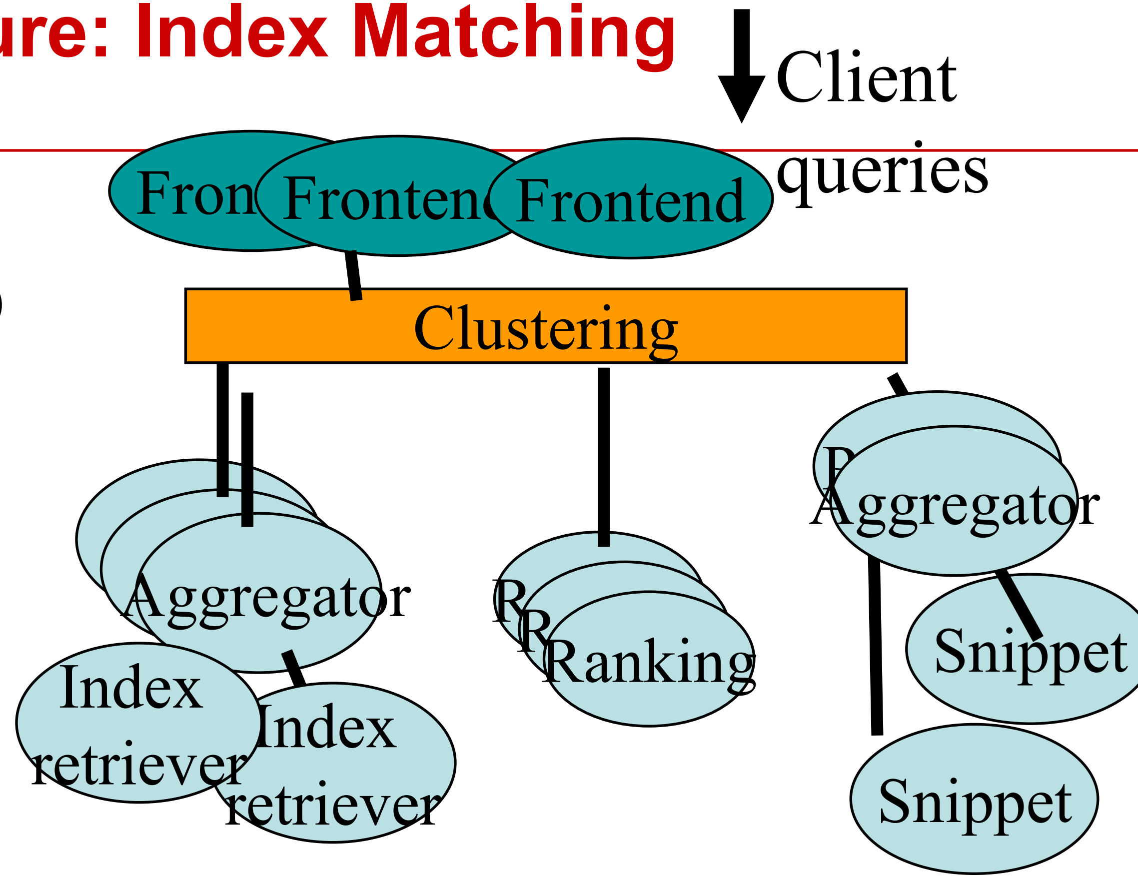
Online Architecture: Frontends and Cache



- **Front-ends**
  - Receive web queries
  - Spawn a thread to handle a request
    - Use cache if possible
    - Otherwise call index matching/ranking
  - Then present results to clients (XML).
- **XML cache :**
  - Save previously-answered search results (dynamic Web content).
  - Use these results to answer new queries.
- **Result cache**
  - Contain all matched URLs for a query.
    - It does not contain the description of these URLs



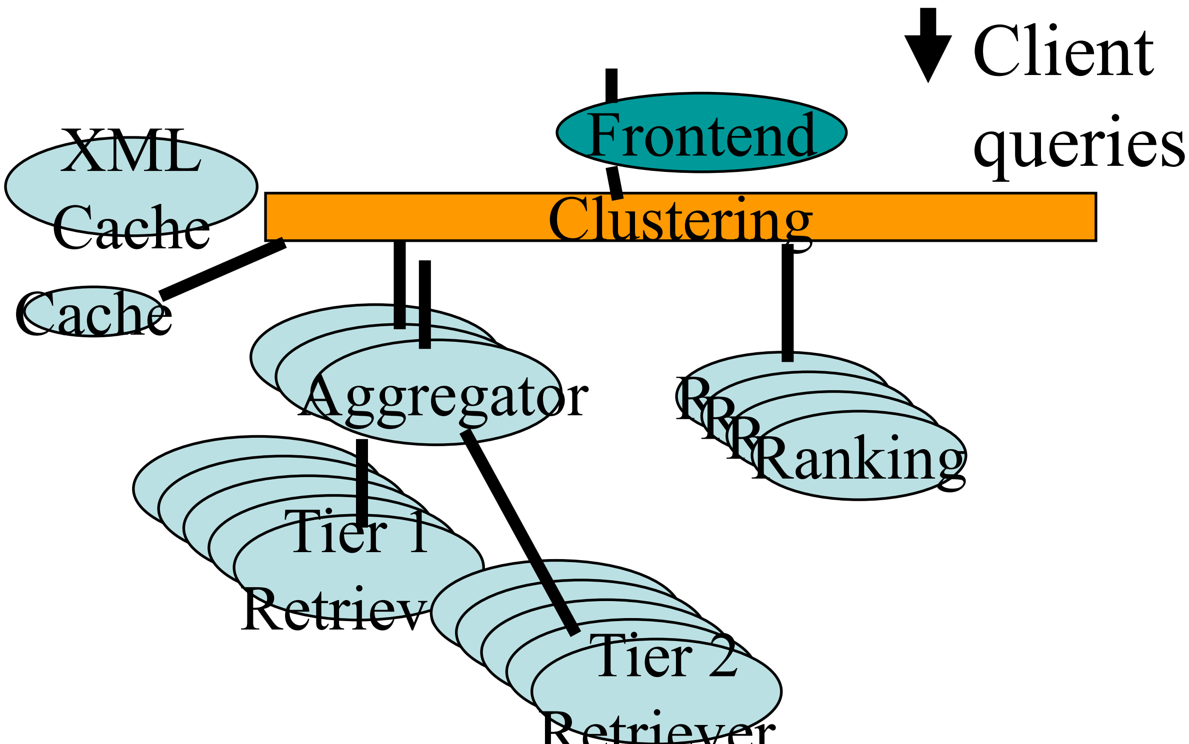
Online Architecture: Index Matching and Ranking



- **Retriever aggregators** (Index match coordinator)
  - Gather results from online database partitions.
- **Index retrievers**
  - Match pages relevant to query keywords
- **Ranking server**
  - Classify pages into topics & Rank pages
- **Snippet aggregators**
  - Combine descriptions of URLs
- **Dynamic snippet servers**
  - Extract proper description for a given URL.

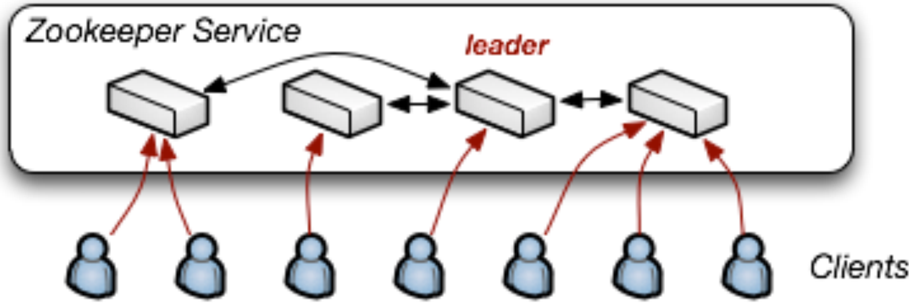
Distributed Coordination on Service Availability

- **Making a remote service call is possible, but how to coordinate information sharing?**
  - How does a machine know there are multiple copies of the same remote service available?
  - How does a machine know a remote service is down?

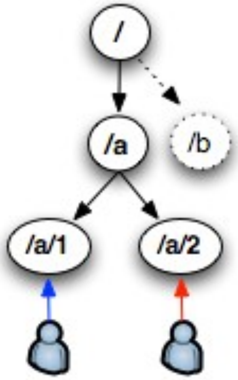


ZooKeeper: Open source coordination service for distributed applications

- **Coordinating distributed systems as “zoo” management:** http://zookeeper.apache.org
- **Start with support for a file API:**
  - A tree-like directory structure (znodes)
  - The znode will be deleted when the creating client's session times out or it is explicitly deleted
  - Partial writes/reads/rename by clients
- **Ordered updates and strong persistence guarantees**
  - Watches for data changes and ephemeral nodes
- **Distributed applications**
  - Configuration management
  - Synchronization
  - Group services



Zookeeper Operations



Operation	Description
create	Creates a znode (the parent znode must already exist)
delete	Deletes a znode (the znode must not have any children)
exists	Tests whether a znode exists and retrieves its metadata
getACL, setACL	Gets/sets the ACL (access control list) for a znode
getChildren	Gets a list of the children of a znode
getData, setData	Gets/sets the data associated with a znode
sync	Synchronizes a client’s view of a znode with ZooKeeper

Exercise: Design options for fast query processing

Assume 3 word queries	#I/O Operations	Time cost	Design options/strategies
Query word intersection of postings	3 random I/O operations to read 3 posting lists List length upto 100MB	1 or few seconds	
Rank top 1000 results	1000 random I/O operations to access features 1000bytes/doc	1000*HHD access=10 seconds 1000*SSD =100ms	
Generate 10 snippets	10 random I/O operations to read docs Each doc -2KB	10 *HHD =100ms 10*SSD=1ms	

Exercise: Strategies for fast query processing

Assume 3 word queries	#I/O Operations	Time cost	Design options/strategies
Query word intersection of postings	3 random I/O operations to read 3 posting lists. Posting list length upto 100MB	1 or few seconds	• Cache postings • Place the entire index in memory
Rank top 1000 results	1000 random I/O operations to access features 1000bytes/doc	1000*HHD access=10 seconds 1000*SSD =100ms	• Cache features, limited locality • Place all in memory • Use SSD
Generate 10 snippets	10 random I/O operations to read docs Each doc -2KB	10 *HHD =100ms 10*SSD=1ms	Use SSD

Exercise: Data distribution for parallel computing

Assume p machines for each service	**Key datasets and sizes**	**Method/design options** **How to assign data to p machines?**
Query match n documents m terms	Posting lists of terms Space cost O(n ln m) 2KB/document 100M docs → 200GB	
Rank top K results	Features of documents 100B/document 100M docs→10GB	
Generate 10 snippets	Document text 4KB/document 100M→400GB	

Exercise: Data distribution for parallel computing

Assume p machines for each service	Key datasets and sizes	Design options
Query match	Posting lists of terms 2KB/document 100M docs → 200GB	Document-oriented: Divide/map documents into p machines Term oriented: Divide terms into p machines
Rank top K results	Features of documents 100B/document 100M docs→10GB	Distribute feature vectors by documents to p machines? Or maybe just use one machine
Generate 10 snippets	Document text 4KB/document 100M→400GB	Distribute documents to p machines

Takeaways for Online Query Processing

- **A complex online system can use tens or hundreds of services running on thousands of machines**
- **Consider impact of system and architecture performance numbers in all designs:**
  - Think about scalability
    - Data: what happens data size increases by 10x or 1000x
    - Also software/machine/human aspects
  - Consider the interaction of response time and throughput
- **Strategies for faster performance**
  - Caching in memory hierarchy
  - Parallel processing of query matching and ranking
    - Document-based vs term based distribution
- **Open source packages are available for online service programming**
  - Zookeeper, Solr
  - Many companies have their internal software