Lecture 8, 11, and 12

P2P with TomP2P

http://tomp2p.net/doc

Advanced Topics

Universität Zürich[UZH]

*Original slides for this lecture provided by David Hausheer (TU Darmstadt, Germany), Thomas Bocek, Burkhard Stiller (University of Zürich, Department of Informatics, Communication Systems Group CSG, Switzerland,

0. Lecture Overview

1. Bloom filters
2. Mechanisms based on Hashing in DHTs
3. Map Reduce
4. NoSQL
5. References

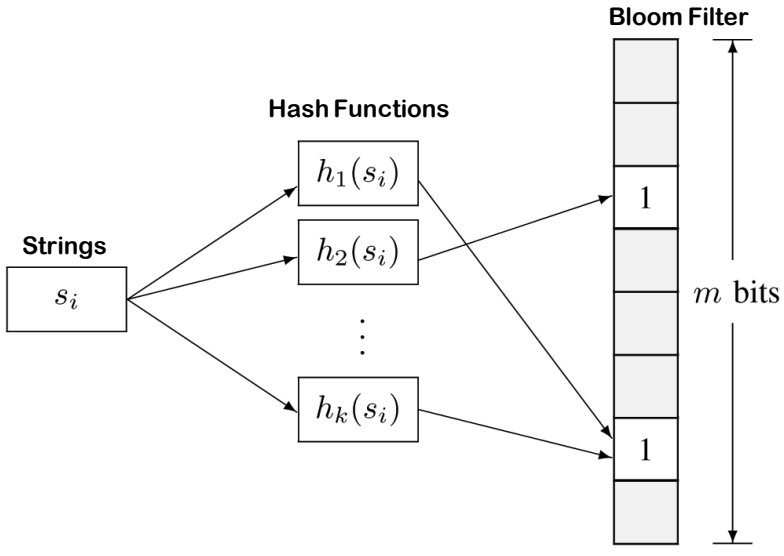
1. Bloom Filters

Traditional Bloom Filter, Attenuated Bloom Filter

Traditional Bloom Filter

- **An array of *m* bits, initially all bits set to 0**
- **A bloom filter uses *k* independent hash functions**
  - *h1, h2, …, hk* with range {1, …, m}
- **Each key is hashed with every hash function**
  - Set the corresponding bits in the vector
- **Operations**
  - Insertion
    - The bit A[hi(x)] for 1 < i < k are set to 1
  - Query
    - Yes if all of the bits A[hi(x)] are 1, no otherwise
  - Deletion
    - Removing an element from this simple Bloom filter is impossible

Insertion of an Element



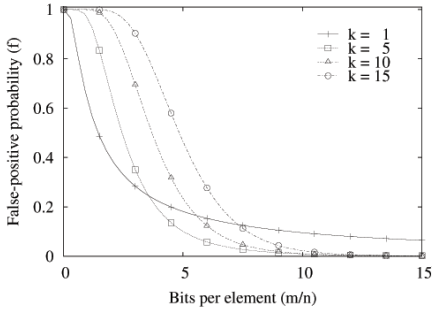
Properties

- **Space Efficiency**
  - Any Bloom filter can represent the entire universe of elements
    - In this case, all bits are 1
- **No Space Constraints**
  - Add never fails
  - But false positive rate increases steadily as elements are added
- **Simple Operations**
  - Union of Bloom filters: bitwise OR
  - Intersection of Bloom filters: bitwise AND

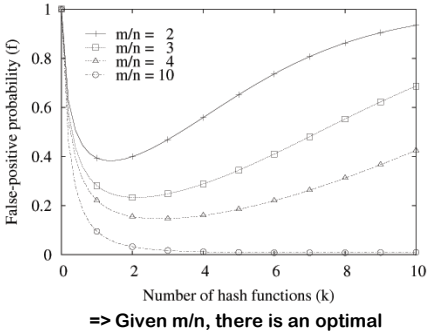
False-Positive Probability

- **No false negative, but false positive**
- **False-positive probability:**
  - *n* number of strings; *k* hash functions; *m*-bit vector

$$f = (1 - e^{-\frac{nk}{m}})^k$$



=> A longer bit vector and fewer insertions are always better



=> Given m/n, there is an optimal number of hash functions (opt. k = m/n ln 2) (when 50% of the bits are set)

Examples

- **Example for False-positives**
  - Insertions
    - Hash („color printer“) => (1,4,6)
    - Hash („digital camera“) => (3,4,5)
    - Bloom filter (1,3,4,5,6)
  - Query
    - Hash („heat sensor“) => (3,4,6)
    - Matches since bits 3,4,6 are all set to 1
- **False-negative**
  - Query
    - Hash (“color printer”) => (1,4,6) , matches (1,3,4,5,6) → no false-negative

Bloom Filter Variants (1)

- **Compressed Bloom Filters**
  - When the filter is intended to be passed as a message
  - False-positive rate is optimized for the compressed bloom filter (uncompressed bit vector m will be larger but sparser)
  - However, compression/decompression, more memory
- **Generalized Bloom Filter**
  - Two type of hash functions gi (reset bits to 0) and hj (set bits to 1)
  - Start with an arbitrary vector (bits can be either 0 or 1)
  - In case of collisions between gi and hj, bit is reset to 0
  - Produces either false positives or false negatives

Bloom Filter Variants (2)

- **Counting Bloom Filters**
  - Entry in the filter not be a single bit but a counter
  - Delete operation possible (decrementing counter)
- **Scalable Bloom Filter**
  - Adapt dynamically to number of elements, consist of regular Bloom filters
- **Attenuated Bloom Filter**
  - Use arrays of Bloom filters to store shortest path distance information
  - Each neighbor link is associated with an attenuated Bloom filter
  - An attenuated Bloom filter consists of *d* normal Bloom filters

    → *d*th filter keeps track of resources reachable via *d* hops.
  - Finds fast only resources within *d* hops, false positive

Sean C. Rhea, John Kubiatowicz: Probabilistic Location and Routing; Infocom 2002.

Example and Applications

- **Demo**
  - Setup: Bloom Filter of size 128 bits, 20 Number160 objects
- **Applications: Distributed Caching, Spell checking, Routing, (distributed) Databases**
- **B-Tracker uses Bloom Filters**
  - M06, slide 22 “To avoid duplicates send compressed list of known peers”
  - Idea: store peers in Bloom Filter and send it. Other peers only send us peer not in the Bloom Filter
    - Less traffic (request is larger, reply may be smaller)
    - False positive are possible
- **Demo: Bloom Filter for get() in TomP2P**

2. Mechanisms based on Hashing in DHTs

And / or searching
Similarity Search
Range queries

Mechanisms based on Hashing in DHTs

- **Search in DHT**
  - `DHT.get(h(„Communication Systems Group“))`
  - In order to find it: `DHT.put(h(„Communication Systems Group“), value)`
- **Keywords**
  - `DHT.get(h(„Communication“))`
  - Find it: `DHT.put(h(„Communication“), value)`, `DHT.put(h(„Systems“), value)`, `DHT.put(h(„Group“), value)`
  - value points to `h(„Communication Systems Group“)`
- **Keywords drawbacks**
  - Find good keywords → “the”, “a” are not good keywords
  - Exact matches only

Mechanisms based on Hashing in DHTs

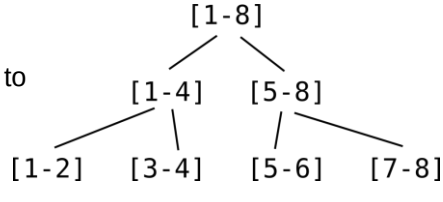
- **Find Communication OR Systems**
  - `DHT.get(h(„Communication“))` and `DHT.get(h(„Systems“))`, combine results
- **Find Communication AND Systems**
  - 1. `DHT.get(h(„Communication“))` and `DHT.get(h(„Systems“))`, intersect results
    - Overhead – use Bloom Filters
  - 2. `DHT.get(h(„Communication“) xor h(„Systems“))`
    - In order to find it: `DHT.put(h(„Communication“) xor h(„System“), value)`, `DHT.put(h(„Communication“) xor h(„Group“), value)`, `DHT.put(h(„Group“) xor h(„System“), value)`
    - Combination needs to be known in advance

Mechanisms based on Hashing in DHTs

- **Range Queries**
  - Problem: random insert vs. sequence insert
  - Max. nr of items (n), nr of items per peer (m)
  - Sequence → [0..n] [n..2n] [2n..3n] […] → peer responsible for range, hash it, store it, done.
    - But random: worst case: 1 peers has 1 data item, range query for range [0..x] contacts x/n peers.
- **Over-DHT**
  - **PHT**: trie (prefix tree); **DST**: segment → tree on top of DHT
  - Main idea: hash of tree-node (resp. for range) → DHT
  - PHT: Peer stores n data items, if n reached, splits data (moves data across peers)
  - DST: stores data on each level (redundancy) up to a threshold
    - No data splitting

Mechanisms based on Hashing in DHTs

- **Example:**
  - Set n = 2, m=8
  - `1, "test"; 2, "hallo"; 3, "world"; 5, "sys"`
- **Tree: store value**
  - Translate `putDST(1, "test")` to
    - put(hash([1-8]),"test") → may be stored (only if threshold not reached)
    - put(hash([1-4]),"test") → may be stored
    - put(hash([1-2]),"test") → will be stored
    - Store `put(3, "world"),put(2, "hallo")` and `put(5, "sys")`



Mechanisms based on Hashing in DHTs

- **Query `getDST(1..5)` translates to**
  - `get(hash[5-6])` → returns “sys”
  - `get(hash[1-4])` → returns “test”, “world” and tells us that threshold has been reached
  - `get(hash[1-2])` → returns “hallo”, “test”
  - `get(hash[3-4])` → returns “world”
- **Range query as series of `put()` and `get()`**
- **Demo**
  - Storage modification

Mechanisms based on Hashing in DHTs

- **Similarity Search in DHT**
  - http://fastss.csg.uzh.ch

- **Project that brings similarity search to HT / DHT**
  - Problem: Search for “netwrk” fails for DHTs
- **Similarity: Edit distance / Levenshtein distance**
  - Min operations to transform one string into another, operations: insert, delete, replace
  - Calculated in matrix size O(m x n)

$$d[i,0] = i,\ d[0,j] = j,$$
$$d[i,j] = min\ (d[i-1,j]+1,\ d[i,j-1]+1,$$
$$d[i-1,j-1] + (if\ s1[i] = s2[j]\ then\ 0\ else\ 1))$$

Mechanisms based on Hashing in DHTs

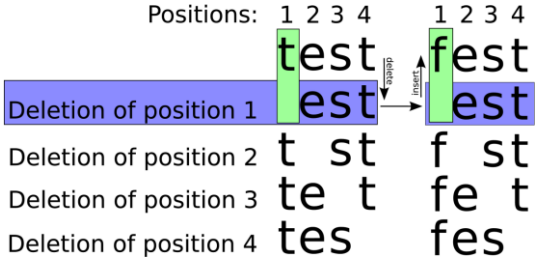
- **Example d(test,east) = 2 (remove a, insert t)**

		T	E	S	T
	0	1	2	3	4
E	1	1	1	2	3
A	2	2	2	2	3
S	3	3	3	2	3
T	4	3	4	3	2

- **Expensive operation if all words need testing**
- **Main idea: pre-calculate errors**
  - All possible errors? Neighbors for test with ed 2: test, testa, testaa, testab, ... , tea, teb, tec, ..., teaa, teab, ... → 23883 more of those!

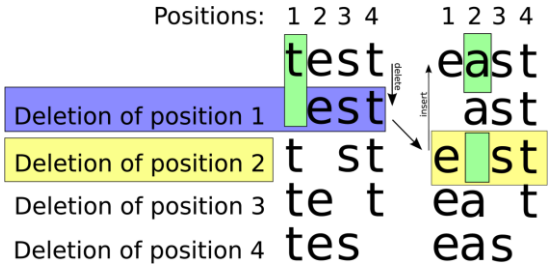
Mechanisms based on Hashing in DHTs

- **FastSS pre-calculates with deletions only**
  - Neighbors for *test* with ed 2: test, est, st, et, es, tst, tt, ts, tet, te, tes
  - Pre-calculation on query **and** index
  - 11 neighbors → 11 more queries, indexed enlarged by 11 entries
- **Example d(test,fest)=1** (query) (index)



Mechanisms based on Hashing in DHTs

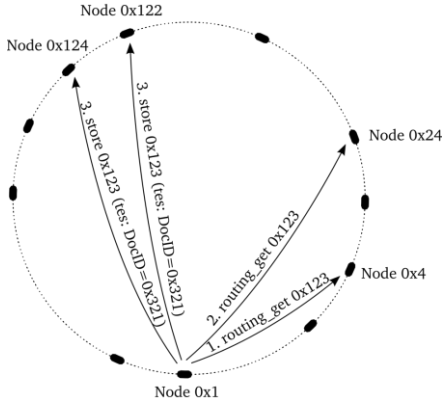
- **Example d(test,east)=2 (query) (index)**



- **P2PFastSS implemented on top of TomP2P (early version) – tests with indexing Wikipedia abstracts**

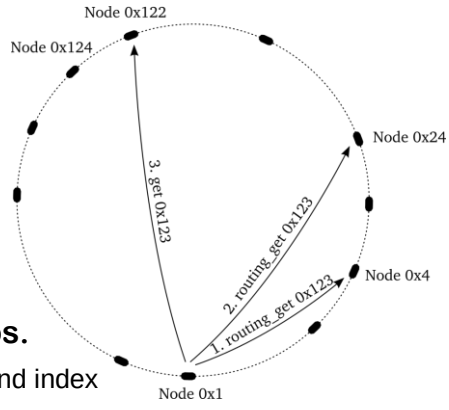
Mechanisms based on Hashing in DHTs

- **Index documents using** `put(hash(document), document)`
  - Document (0x321) contains word test
- **Index all neighbors (test, tes, tst, tet, est) using** `put(hash(neighbor), point to document)`
  - `hash("tes") = 0x123`



Mechanisms based on Hashing in DHTs

- **User searches for “tesx”**
- **Neighbors are generated (tesx, esx, tsx, tex, tes)**
  - get(hash(neighbor)) → 0x123
  - Find pointer to document (0x321)
  - document = get(0x321)
- **Tests with edit distance 1, partially 2, ignoring delete pos.**
  - Overhead (*n choose k*) for query and index
- **Similarity search as series of `put()` and `get()`**
- **Demo**



3. MapReduce

Introduction, Idea, Examples, Demo

Introduction

- **Inspired by functional programming (Lisp, Haskell, …)**
- **Main Idea**
  - Map operates on a list of values to produce a new list of values, applying the same computation
  - Reduce operates on a list of values to combine those values into a single / few values, applying the same computation
- **Used to deal with large data-sets in-parallel on large clusters in a reliable, fault-tolerant manner**
- **Map(k1, v1) → k2, list(v2)**
  Reduce(k2, list (v2)) → k3, list(v3)
- **Apache Hadoop, mongoDB, Google MapReduce, riak, TomP2P, …**

Example

- **Hello World of Map Reduce: WordCount**
  - Input text

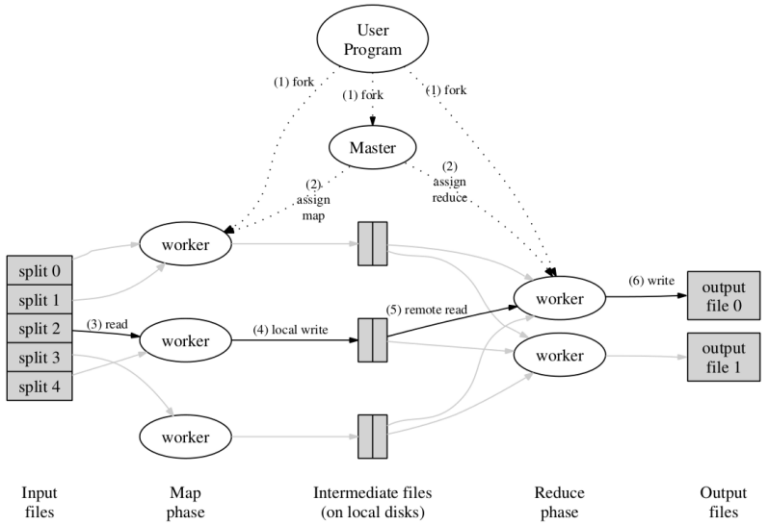
```
map (String value) {
  for each word w in values {
    store(w, 1);
}}

reduce (List intermediate_values) {
  int result = 0;
  for each v in intermediate_values {
    result += v;
    store(result);
}}
```

Example

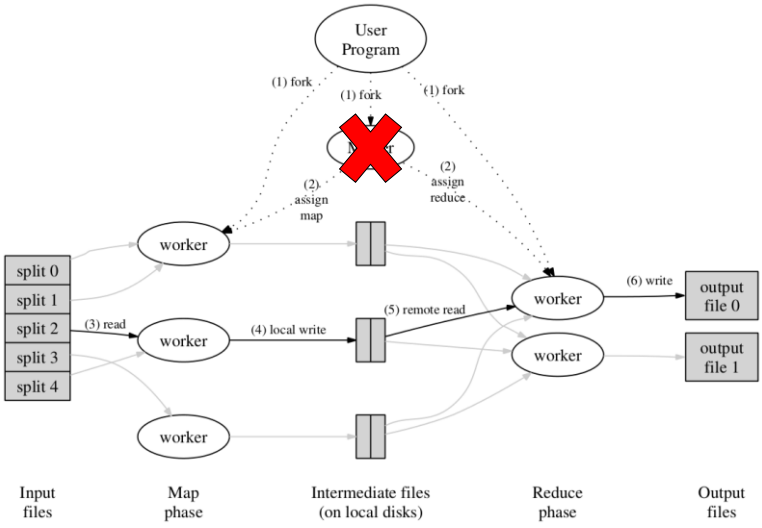
- **MapReduce example**
  - Text1: “to be or not to be”
  - Text2: “to do is to be”
  - Text3: “to be is to do”
- **Map phase**
  - store (to, 1), store (be, 1), store (or, 1), store (not, 1), store (to, 1), store (be, 1)
  - store (to, 1), store (do, 1), store (is, 1), store (to, 1), store (be, 1)
  - store (to, 1), store (be, 1), store (is, 1), store (to, 1), store (do, 1)
- **Reduce phase**
  - On node that stores “to” → add it → 6 → store it on node X
  - On node that stores “be” → add it → 4 → store it on node X
  - On node that stores “or” → add it → 1 → store it on node X
  - On node that stores “not” → add it → 1 → store it on node X
  - On node that stores “do” → add it → 2 → store it on node X
  - On node that stores “is” → add it → 2 → store it on node X
- **Result phase**
  - Get data from node X

Mechanism



Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113.

Mechanism



Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113.

Demo

- **For TomP2P we need to map this to the key-value concept** → submitTask()
  - OverDHT (same category as FastSS, DST)
- **Redundancy**
  - Always record from which peer the result came
- **Demo with TomP2P**
  - Redundancy not supported yet
  - Load balancing not supported yet

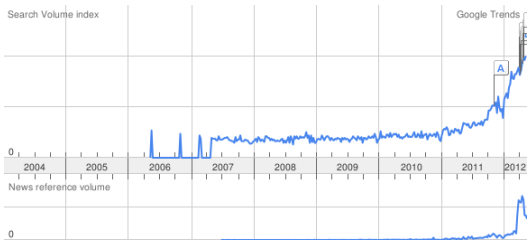
```
ReduceTask reduce = new Task() {
 exec(key, data) {
  int r[] = get(key)
  a=sumAll(r);
  DHT.addList(result, a);
}}
MapTask map = new Task(){
 exec(key, data) {
  String text = data.getText();
  for each word w in text {
    DHT.addList(w, 1)
}}}
DHT.submit(key1, map, text1);
DHT.submit(key2, map, text2);
DHT.submit(key3, map, text3);
//start reducer once all are done
for each word w from futureTasks {
  DHT.submit(w, reduce);
}
DHT.get(result);
```

4. NoSQL

Introduction, Idea, Examples
Concept

Introduction

- **NoSQL: Not only SQL (http://nosql-database.org/)**
  - Non-relational, distributed, scalable, fault-tolerant
  - Types
    - Key/Value Storage
    - Document Storage
    - Column Storage
    - Graph Storage
    - Others: Geospatial, Object (not much difference to document storage)



- **Google trend for "Big Data"**
- **UnQL (Unstructured Query Language)**
  - Demo TomP2P (small subset of UnQL)
  - … order by …

NoSQL

- **Eventually consistent**
  - BASE (Basically Available, Soft state, Eventual consistency) ↔ ACID
  - Waiting long enough → get consistent data
  - May return inconsistent data in the meantime
  - Conflict resolution: Lamport timestamps and vector clocks

- **CAP Theorem (Consistency, Availability, Tolerance to network partitions)**
  - Distributed computing
  - At most two of them, focus on AP

Key / Value

- **Types of consistency (Cassandra) write / read**
  - One – if one node saves the data, it is reported to the client
  - Quorum – if a majority reports to save the data, report it
  - All, if a nodes save the data, report it
  - TomP2P: Quorum, everything else → its complicated ☺

- **Key / Value: collection of Key / Values**
  - Find fast content with given key / Big Data
    - TomP2P is an example for Key / Value storage
    - Memcache (memory only), Redis, SimpleDB, …

- **Demo TomP2P: inconsistency join / leave**
  - DHT attack – countermeasures – voting schemes

Document Storage

- **Similar to Key / Value, Value is document**
  - CouchDB, MongoDB
  - Access mostly through HTTP
  - Store data as JSON / BSON
- **Schema-less**
  - Types determined by the application

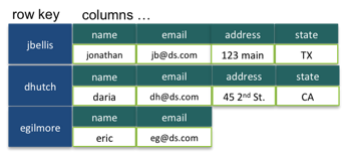
```
"name" : "Jim",
"number" : 052435345


"name" : "Bob",
"number" : 091454353,
"lectures" : "612 - Peer-to-Peer Systems and Applications"
```

- **Demo TomP2P: WebService / JSON**

Column Storage

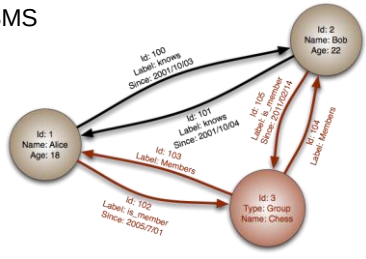
- **Key associated with multiple attributes**
  - TomP2P similar (locationKey, domainKey, contentKey, value)
  - Can “emulate” Key / Value
  - HBase, Cassandra, Google BigTable
  - No constraints / foreign key
  - Denormalization (avoid in RDBMS)
- **How to make TomP2P a column storage**
  - Split locationKey or domainKey or contentKey (64bit / 96bit)
  - Keyspace, column family objects, row key, columns (name, value)
- **Demo TomP2P: multiple columns**



http://www.datastax.com/docs/1.0/ddl/column_family

Graph Databases

- **Focus on structure of data**
  - Nodes (similar to objects in OO languages), edges, and properties
  - Neo4j, InfiniteGraph
  - From http://neo4j.org: "For many applications, Neo4j offers performance improvements on the order of 1000x or more compared to relational DBs."
    - Can be more efficient than joins in RDBMS
- **Graph database with TomP2P?**
  - At least one CT group does recommendation system with graphs
  - Demo next week!



http://en.wikipedia.org/wiki/Graph_database

Other Types / Conclusions

- **Geospatial databases**
  - Support: MongoDB, BigTable, Cassandra, CouchDB
  - MongoDB: supports two-dimensional geospatial indexes
    - Find closest N items (e.g. airports) to location X
  - Map this to a DHT – SpatialP2P

    Storing and Indexing Spatial Data in P2P Systems; V. Kantere, S. Skiadopoulos, T. Sellis, IEEE Transactions on Knowledge and Data Engineering
- **Conclusions: NoSQL vs. SQL**
  - Choose the right tool for the job!
  - high-volume, high-availability use case
  - There is no free lunch, SQL is very powerful
  - SQL on top of NoSQL → is it possible?

4. References

- Burton H. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors; Communications of the ACM, Vol. 13(7), pp. 422-426, 1970.
- S. C. Rhea and J. Kubiatowicz: Probabilistic Location and Routing; IEEE INFOCOM 2002, New York, NY, USA, pp. 1248-1257, June 2002.
- B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal: The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables; SODA 04.
- S. Cohen, Y. Matias: Spectral Bloom Filters; SIGMOD 2003.
- L. Fan, P. Cao, J. Almeida, A. Broder: Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol; IEEE/ACM Transactions on Networking, Vol. 8(3), pp. 281-293, June 2000.
- M. Mitzenmacher: Compressed Bloom Filters; IEEE Transactions on Networking, Vol. 10 (5), pp. 604-612, October 2002.
- A. Broder and M. Mitzenmacher: Network Applications of Bloom Filters: A Survey; 40th Annual Allerton Conference on Communication, Control, and Computing, pp. 636-646, 2002.
- T. Kocak, I. Kaya: Low-Power Bloom Filter Architecture for Deep Packet Inspection; IEEE Com. Letters, Vol. 10(3), pp. 210-212, March 2006.
- S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood: Deep Packet Inspection using Parallel Bloom Filters; IEEE Micro, Vol. 24 (1), pp. 52-61, January 2004.
- R. Laufer, P. Velloso, O. Duarte: Generalized Bloom Filters; Technical Report GTA-05-43, COPPE/UFRJ, September 2005.
- Changxi Zheng , Guobin Shen , Shipeng Li , Scott Shenker, „Distributed Segment Tree: Support of Range Query and Cover Query over DHT“, The 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)
- MapReduce
  - http://research.google.com/archive/mapreduce.html
  - http://www.cs.cornell.edu/courses/cs3110/2009sp/lectures/lec05.html
  - Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113.
- NoSQL
  - http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
  - http://www.mongodb.org/display/DOCS/Geospatial+Indexing
  - http://www.directionsmag.com/articles/nosql-databases-what-geospatial-users-need-to-know/164635
  - http://www.cloudtweaks.com/2011/02/a-history-of-cloud-computing/
  - http://nosql-database.org/
  - http://newtech.about.com/od/databasemanagement/a/NosqlHtm
  - http://ayende.com/blog/4500/that-no-sql-thing-column-family-databases
  - http://www.datastax.com/docs/1.0/ddl/about-data-model