

A Concept of Large Scale Disk-Based Metric Indexing Structure for Approximate Information Retrieval by Content

Stanislav Barton^{*‡}, Valerie Gouet-Brunet^{*}, Marta Rukoz^{†‡},
Philippe Rigaux[‡]

^{*}Conservatoire National des Arts et Metiers

[‡]LAMSADE - Universite Paris Dauphine

[†]Universite Paris Ouest Nanterre La Defense

Paris, France

June 30, 2010

- Motivation
- Disk-based Access Structure Outline
- Experimental Evaluation
- Going for Really Large Scale via Map-Reduce Framework
- Preliminary Results
- Conclusions

Motivation

- Large collections of multimedia data exist : Flickr, Google Images, etc.
- Search implemented usually on the textual meta-data represented by annotations or tags
- Search by content – a paradigm where automatic object description based on its content is exploited
- The content of an object is represented as a vector (multi-dimensional) – the search for similar objects then searches for **nearest** vectors in vector space
- Extensibility – object representation and distance function (no limitation to L_P metrics – Euclidean vector spaces)
- Content-based search is a challenging problem on large (web) scale collections

Motivation

- Many access structures work well on vector data, but are limited by main memory (LSH, MChord, Metric Inverted File, etc.)
 - With transition to cheaper and larger disk memory, performance degrades significantly
 - Scalability addressed by distribution of the access structure to cluster of computers (one computer with the same RAM is much more \$\$\$)
- Proposed access structure trades approximation and slower query processing times for high scalability by transition to cheap secondary memory

Extensibility via Metric Spaces

The metric space is defined as a pair (D, d) where D denotes the domain of objects and $d : D \times D \rightarrow \mathbb{R}$ is a total function:

$\forall x, y \in D, d(x, y) \geq 0$	non-negativity
$\forall x, y \in D, d(x, y) = d(y, x)$	symmetry
$\forall x, y \in D, x = y \Leftrightarrow d(x, y) = 0$	identity
$\forall x, y, z \in D, d(x, z) \leq d(x, y) + d(y, z)$	triangle inequality

Very sparse distance matrix:

- Based on matrix of mutual distances – problem with quadratic complexity (AESA [Ruiz, 1986] – Approximating and Eliminating Search Algorithm)
- Concept of reference objects already known – pivots P (Linear AESA [Micó *et al.* , 1994])
- Compute $|P| \times n$ distance computations – n is the size of indexed collection $|P| \ll n$
- Store only m closest distances for each object – $m \ll |P|$ (Metric Inverted File [Amato & Savino, 2008])
- Storage complexity – $m \times n$

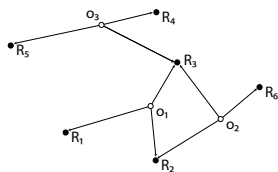
Index creation:

- Reference objects are *randomly* selected from the collection
- For each object from the collection, distance to all reference objects is computed
- Predefined number m of closest distances are maintained for each object

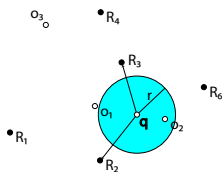
Search:

- 1 Find s nearest neighbors among reference objects to the query object
- 2 Prune the collection by the distances to the s nearest reference objects (filtering)
- 3 Verify the candidates by computing original distance to the query object

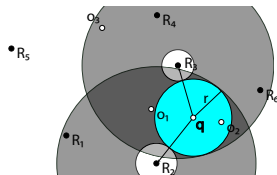
Access Structure Outline



Assignment



RO cells



Filtering with 2 ROs

- Arrows for assignment denote closest three ROs for each object
- Different colors denote different cells and their intersection ($s = 2, R_2, R_3$)
- Use also the distances of q to closest ROs to filter out more candidates
 - An object o **can** be pruned if $|d(q, RO) - d(RO, o)| > r$
 - An object o **cannot** be pruned if $d(q, RO) - r < d(RO, o) < d(q, RO) + r$
 - An object must lie in the intersection

Access Structure Implementation

- The access structure is implemented as a relational database
- One table for each reference object, storing the object ID and distance from RO
- Indices are kept for object IDs and distances
- Filtering (pruning) is inner join of s closest reference object tables with predefined distance ranges
- Verification of candidates by computing the original distance after object representation retrieval
- Tables and object representations kept on disk (separately)

Parameter Setting

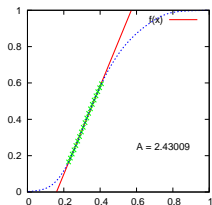
- On various data, one access structure performs differently
- One access structure on one data performs differently under various parameter settings
- Tight correlation between indexability of data and its intrinsic dimension
- How to select the correct number of reference object?

$$|P| = \frac{m \times n}{O}$$

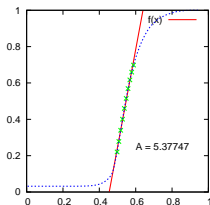
O estimated average number of objects per table

m data's intrinsic dimensionality

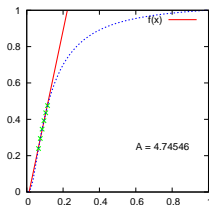
Cumulative Distribution Slope [Barton et al. , 2010]



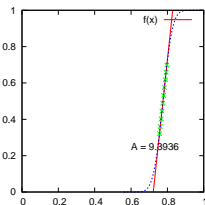
1 RGB



9 SIFT



11 Audio



101 Uniform

- Normalize cumulative pairwise distance distribution histogram
 - Estimate slope of the flat part
 - Use estimations on uniform data sets (embedding \simeq intrinsic dimensionality) to calculate intrinsic dimensionality value
- Extensible – work with any distance function, metric space friendly

ID	Descriptor	Dim.	Size	Dist.	CDS
<i>Global Visual Descriptors Data Sets</i>					
1	RGB histogram	125	500,000	L_2	5
2	RGB histogram		1,000,000	L_2	
<i>Local Visual Descriptor Data Sets</i>					
11	SIFT	128	500,000	L_2	8
12	SIFT		1,000,000	L_2	
13	SIFT		30,000,000	L_2	
<i>Audio Descriptors Data Sets</i>					
21	MFCC	351	500,000	SKLD	7 (20)
22	MFCC		1,000,000	SKLD	

- European Web Archive – publicly available video and audio clips processed – thousands of hours
- Audio – Symetrized Kullback-Leibler Divergence (SKLD) is a semi-metric !
- ID 13 represents about 40,000 CD cover pictures

Hardware and Software settings

- 1 linux machine, 2 Intel XEON 2.0 GHz processors, 2 cores each
- 750GB disk
- 64 bit JAVA 1.6, PostgreSQL 8.3

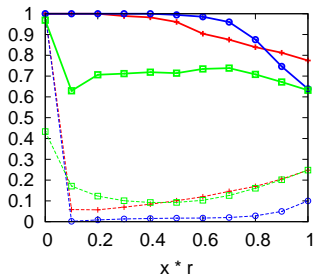
Access Structure Creation Time

- Parameter setting summary for considered data and measured time:

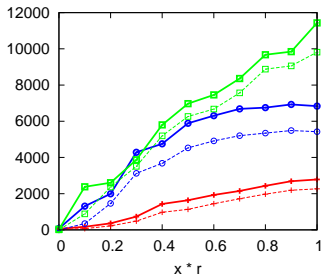
ID	RGB 500k	SIFT 500k	Audio 500k	RGB 1M	SIFT 1M	Audio 1M
m	5	8	20	5	8	20
$ P $	32	50	125	65	100	250
Time (mm:ss)	5:02	7:24	22:03	10:30	15:34	49:24

- Sequential scan used to find nearest ROs to indexed object
- One audio vector 351 dimensions
- No parallelization

Query Processing Speed – Varying Range on 1M Data Sets



R 2 —+— P 12 -o--
 P 2 -+-- R 22 -x--
 R 12 -o-- P 22 -x--



time 2 —+— read 12 -o--
 read 2 -+-- time 22 -x--
 time 12 -o-- read 22 -x--

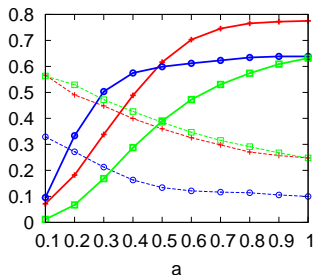
- Parameter $s = 3$, Data sets: ID 2 - RGB, ID 12 - SIFT, ID 22 -Audio
- Set of queries that on average returned 0.5% objects from the 500k data set
- *Time* = average total time to process query
- *Read* = average time spent of reading original object representations from disk
- *Time – Read* = time spent on processing SQL query

Speeding Up the Search

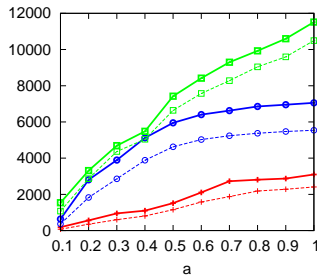
- In comparison with the filtering phase, the verification phase of the search algorithm is very expensive
 - Filtering done in almost constant time
- Reducing the number of candidates – *aggressive pruning*
- An object o **cannot** be pruned if
$$d(q, RO) - r \times a < d(RO, o) < d(q, RO) + r \times a, a \in [0, 1]$$
- Omitting whole verification phase – order candidates w.r.t their *rank* and report k -best

- $rank(c) = \sum_{i=1}^s (d(q, RO_i) - d(RO_i, c))^2$

Speeding Up – Aggressive Pruning



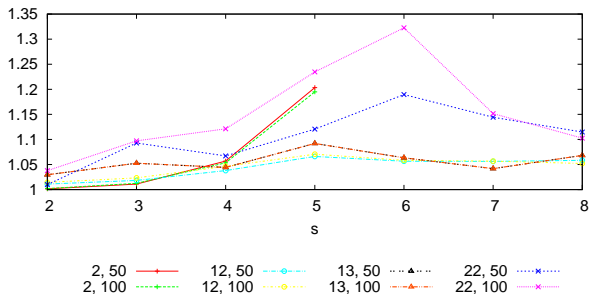
R 2 —+— P 12 —o—
P 2 - -+ - - R 22 —x—
R 12 —o— P 22 - -x - -



time 2 —+— read 12 —o—
read 2 - -+ - - time 22 —x—
time 12 —o— read 22 - -x - -

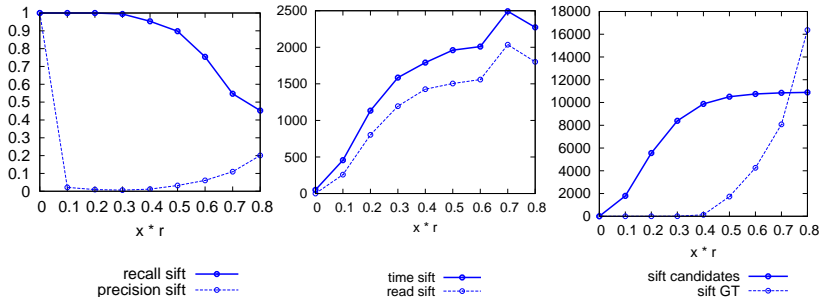
- Varying the pruning constant a – the size of the range annulus
- Parameter $s = 3$, Data sets: 2 - RGB, 12 - SIFT, 22 - Audio

Speeding Up – Omitting Expensive Verification



- Ranking function evaluation on data sets ID 2 (RGB), 12, 13 (SIFT) and 22 (Audio)
- Order candidates by their ranking
- Compute average distance for first k best candidates and k ground truth objects
- Ratios for first 50 and 100 objects is reported

Going for the Large Scale – 30M SIFTs



- $s = 3, a = 1, |P| = 3,500 \sim \frac{30,000,000 \times 8}{80,000}$
- 20 hours to create the index (two parallel processes)
- SQL query processing time still constant and comparable

- Primary motivation was an access structure for:
 - Any multimedia data – not necessarily vectors, not necessarily metric distance function
 - Large scale – scalable to hundreds of millions of objects
 - Common HW – machine with usual main memory and disk
- We have:
 - Experimental evaluation with almost no parallelization optimizations
 - Approximated k NN queries processed within ~2s (30M dataset)
 - Ran on one machine (no super server, 2 years old *better* desktop)
 - Access structure suitable for real time applications

Going for the Really Large Scale

- The concepts exploited by the proposed access structure allow massive parallelization
 - Both the indexation and search can be parallelized
 - The storage itself can be parallelized
- ⇒ Framework for large scale data operations – **MapReduce** and **Google FS** [Dean & Ghemawat, 2004]
- ⇒ Framework for large scale data storage – **Bigtable** [Chang *et al.* , 2006]

Google's MapReduce

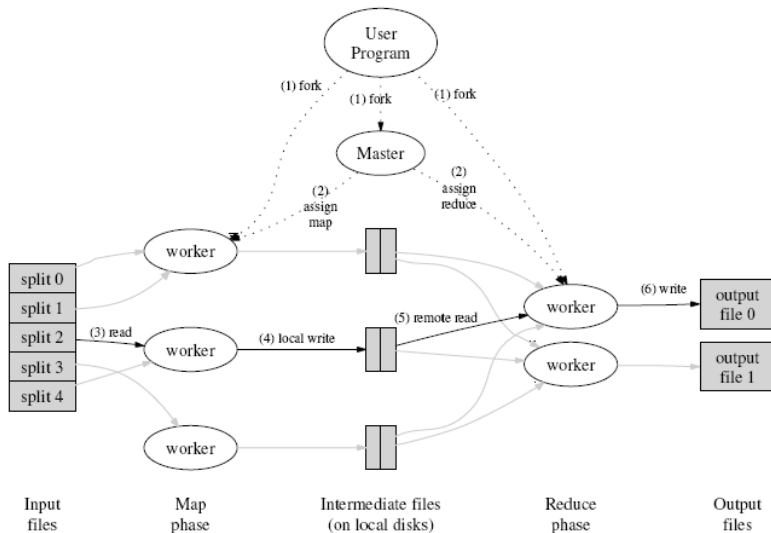
- Simplified framework for data processing on large clusters (of commodity HW)
- In three phases, transforms the data from one form to another, data representation: Key-Value pairs $\langle K, V \rangle$
- Fail-safety via data/process replication
- Speed via data-local computations
- Primary data storage in Google File System - replicated and distributed blocks of data (splits, e.g. 64MB)

Map takes $\langle K_1, V_1 \rangle$ and transforms into $\langle K_2, V_2 \rangle$ pairs

Shuffle/Sort takes $\langle K_2, V_2 \rangle$ and transforms to $\langle K_2, list(V_2) \rangle$

Reduce takes $\langle K_2, list(V_2) \rangle$ and $\langle K_3, V_3 \rangle$

MapReduce Job Execution Diagram



MapReduce Example

Word count on a large file:

- Input file stored in GFS (blocks)
 - Output contains one file of words and their frequencies
-
- 1 Map phase – Tokenize input block, for each word (K) emit $\langle K, 1 \rangle$
 - 2 Shuffle/Sort phase – Merge sort on Map outputs, produce $\langle K, list(1) \rangle$
 - 3 Reduce phase – one Reducer, input $\langle K, list(1) \rangle$, emits $\langle K, frequency \rangle$
-
- The framework automatically *pushes* the large data inputs in parallel through the cluster
 - Prevents the insufficiency of main or secondary memories on the fly – thousands of terms may cause memory leaks
 - Moves computation to data
 - Easy scalability – framework logic is far away from distributed code

Enhanced Distributed Storage – Bigtable

"A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes." [Chang et al. , 2006]

sparse = variable number of columns per row, any value can be *null*

distributed = stored on a cluster of computers

persistent = materialized

sorted map = sorted by unique row ID, no secondary indices available

- By no means a replacement for relational databases
- Stored as files in GFS, so high availability via replication
- Extreme scalability with assured performance – peta bytes of data on thousands on computers
- "row id" → list("column id" → "cell value")

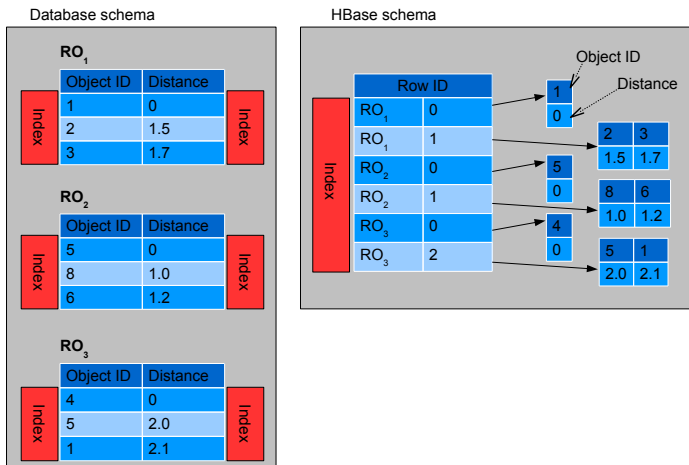
Hadoop and HBase

- Both Google frameworks have open source implementations
- MapReduce is called **Hadoop** (currently in version 0.20.2)
- Google FS is Hadoop's Distributed FS (**HDFS**)
- Bigtable is **HBase** (currently in version 0.20.4)
- All implemented in JAVA
- HDFS works as an overlay on already installed FS, no need for specialized OS installations

Proposed Disk-based Access Structure in Hadoop/HBase

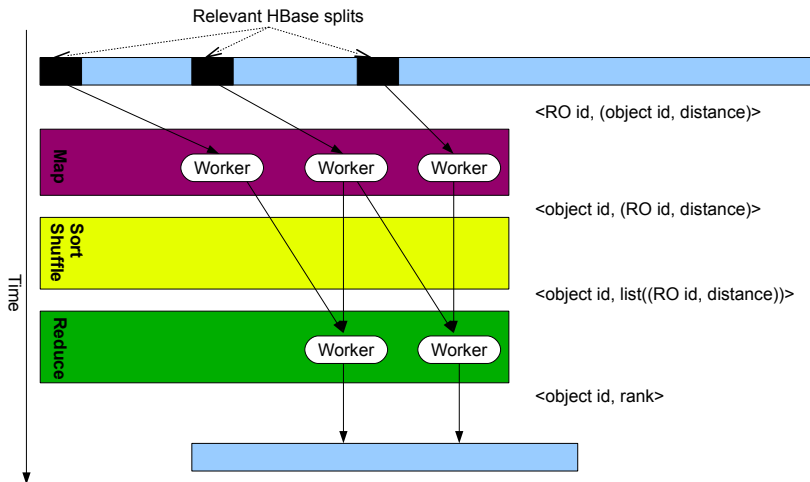
- The index stored in HBase as one big distributed *table*
- The indexation process rewritten as a MapReduce job
 - It takes object representations on the input and feeds the HBase table on the output
- The query processing algorithm rewritten as a MapReduce job
 - No joins in HBase, join is the result of the job
 - Due to a job starting overhead several queries processed in one batch

Storage Transformation



"row id" → list("column id" → "cell value")

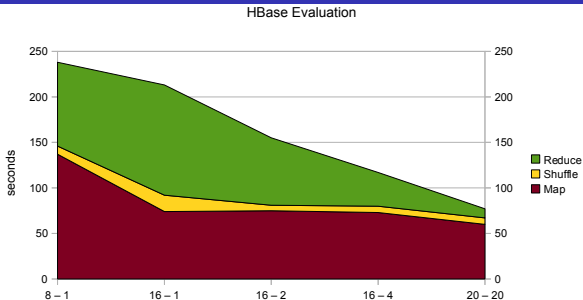
Query Processing – A MapReduce Job



Preliminary Results

- 30 million SIFT data set used
- DB had 240 millions of rows, in HBase 80 millions of rows with on average 4 columns
- 6 machines used, 4 as workers
- 2 four core XEON processors having 2933 MHz per machine
- DB had 36 GB, HBase has 14GB on each worker machine (56GB in total, replication factor 3)
- One batch of 100 queries executed, only filtering measured
- Each input split represents (receives) one Map task, total number is job dependent
- Number of Reducer tasks per job is *predefined*

Preliminary Results



- Map tasks (Mappers) vs. Mapper slots
- First number denotes a number of Mapper slots per cluster
- Second number is the predefined number of Reducer tasks per job
- Always 350 splits to process during Map phase = 350 Map tasks
- All Reduce tasks processed at once
- At the final set up, 1 query per 0.8s processed

Summary

- Parallelization is so far scaling the index performance linearly!
- Twice as much slots gives twice as fast results
- Although rather small cluster
- 30M data set represents only a proof of concept
- Missing large dataset to demonstrate true usability
- Aim is a demo doing real-time search on billions of objects
 - Currently downloading 1TB of data representing 2 billions of SIFTs
- Waiting for resources to test larger clusters (easily done on the SW level – a matter of 3 configuration files)

Thank you for your attention.

Supported by:

- **WISDOM** – the French federation (2007-2010)
- **DISCO** – the French project ANR MDCO (2008-2010)
- **NEUMA** – the French project ANR CONTINT (2008-2011)

References



Amato, Giuseppe, & Savino, Pasquale. 2008.

Approximate similarity search in metric spaces using inverted files.

Pages 1–10 of: *Infoscale '08: Proceedings of the 3rd international conference on scalable information systems*. ICST.



Barton, S., Gouet-Brunet, V., Rukoz, M., Charbuillet, C., & Peeters, G. 2010.

Estimating the indexability of multimedia descriptors for similarity searching.

Pages 1–4 of: *Riao'10 on adaptivity, personalization and fusion of heterogeneous information*.



Chang, Fay, Dean, Jeffrey, Ghemawat, Sanjay, Hsieh, Wilson C., Wallach, Deborah A., Burrows, Michael, Chandra, Tushar, Fikes, Andrew, & Gruber, Robert. 2006.

Bigtable: A distributed storage system for structured data (awarded best paper!).

Pages 205–218 of: *Osd*.



Dean, Jeffrey, & Ghemawat, Sanjay. 2004.

Mapreduce: Simplified data processing on large clusters.

Pages 137–150 of: *Osd*.



Micó, María Luisa, Oncina, José, & Vidal, Enrique. 1994.

A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements.

Pattern recogn. lett., 15(1), 9–17.



Ruiz, E V. 1986.

An algorithm for finding nearest neighbours in (approximately) constant average time.

Pattern recogn. lett., 4(3), 145–157.