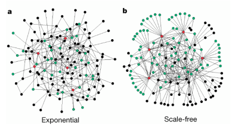
Of Ivory and Smurfs: Loxodontan MapReduce Experiments for Web Search

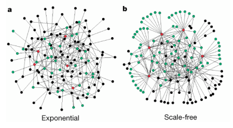
Jimmy Lin, Donald Metzler, Tamer Elsayed,
and Lidan Wang

University of Maryland, College Park and
Yahoo! Research



Retrieval Architecture

- Document partitioning
  - Client issues a query to the broker
  - Broker distributes the query to all partition servers in parallel.
  - Each server computes a ranked list on its assigned document partition independently,
  - Results passed back to the broker
  - Broker merges the results and returns the final ranked list to the client.



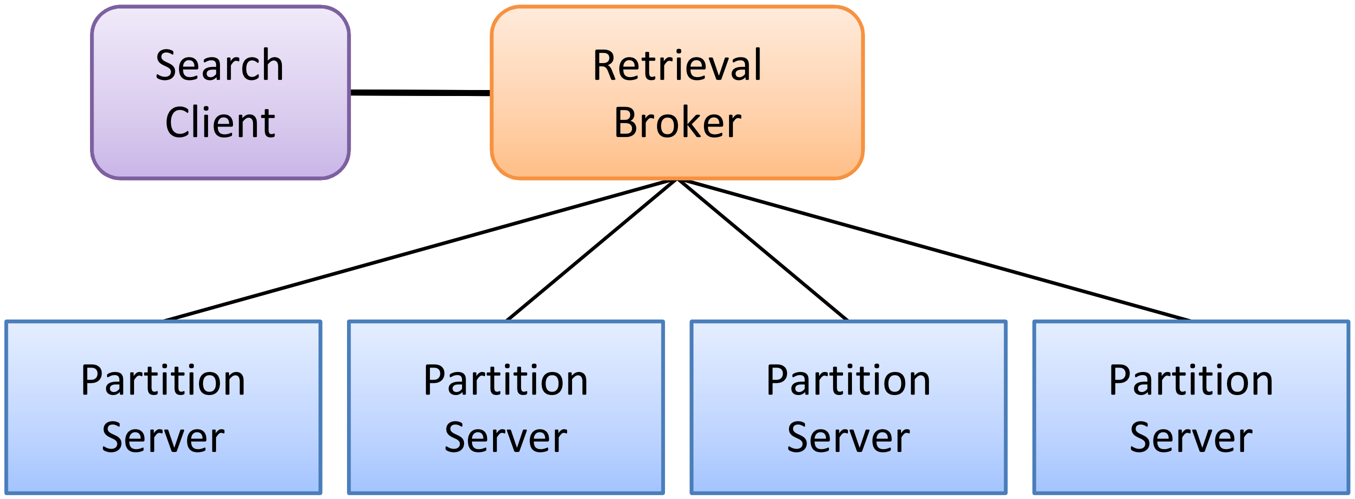
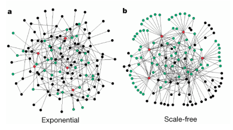
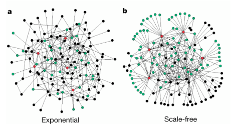


Figure 1: Illustration of a simple broker-mediated, document-partitioned retrieval architecture.



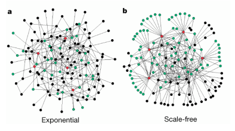
Retrieval Problems in Distributed Environment

- MapReduce designed for batch processing
- DFS: no low latency random access
- In Hadoop, it can take tens of seconds for mappers to launch, since tasks must be queued at the jobtracker before they can be assigned to individual workers.
- Current design of Hadoop limits the rate at which new map tasks can be spawned
- DFS spreads data blocks across nodes in the cluster no single datanode holds an entire file



The typical solution and its problems

- Build indexes with Hadoop (written out to HDFS)
- Copy indexes to standard POSIX file systems
- Two separate architectures
- Splitting hardware resources
- Multiple copies of data
- Workflow management is difficult in a rapidly-evolving research environment.
- Latencies involved in copying indexes over the network



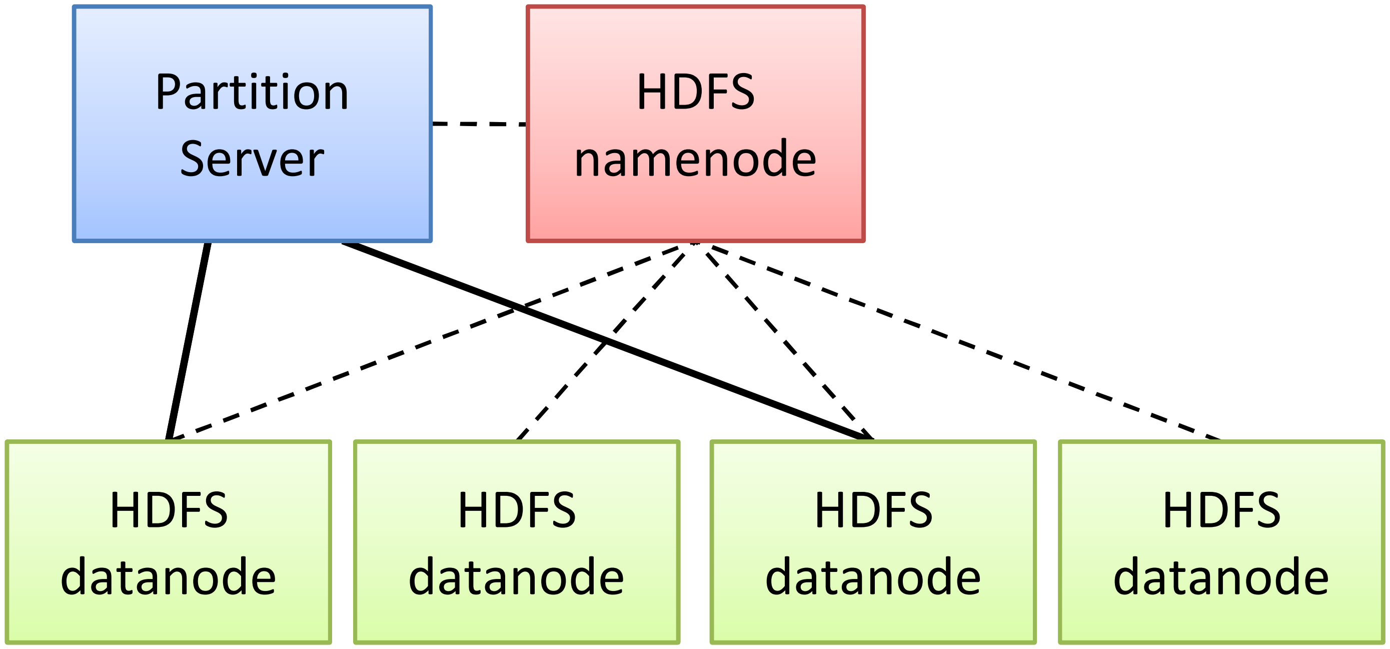
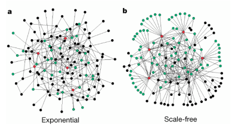
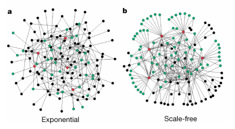


Figure 2: Illustration of Ivory's distributed architecture that involves reading postings directly from HDFS (data transfer shown as solid lines; metadata communication shown as dotted lines).



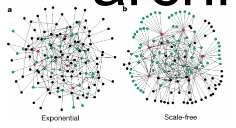
Ivory solution

- Metzler's Search using Markov Random Fields
- The vocabulary holds byte offsets into HDFS-stored index files that correspond to locations of postings lists.
- The fetching of a postings list involves first contacting the namenode for the block location, and then contacting the datanode itself for the actual data.
- The only point of contact between a client and the Hadoop cluster is the jobtracker
  - embedding servers in MapReduce jobs



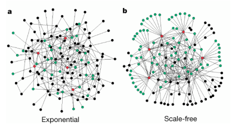
Ivory solution…

- A Map-Reduce to Map over a configuration file specifying the locations of the partition indexes.
- Each mapper reads in the location of the partition index, initializes a query engine, and then launches into an infinite service loop waiting for incoming TCP connections
- When each mapper launches, it first writes its host information into a known DFS location.
- After all the partition servers have been initialized, the broker can be launched as a 1-mapper/0-reducer MapReduce job, reading the host information of all the partition servers and completing the distributed broker architecture.



Ivory MapReduce Indexing

- In a seperate Map-Reduce pass over Reducer output, they construct a postings forward index to store the byte offset position of each postings list.
- Positional indexes: replacing the intermediate term frequency with an array of term positions;
- tuple $\langle t, * \rangle$ sorted to be first to inform the reducer of a term's df before any of its postings arrive.



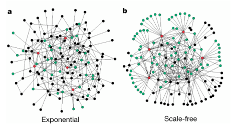
Merging Results Across Partitions

- Independent fusion
  - View results merging as a federated search problem
  - Simplifies index construction
  - Makes document scores across partitions difficult to compare directly
- Global statistics

  1) each of the partition indexes are built independently

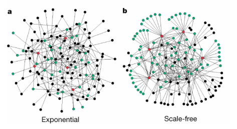
  2) a MapReduce job maps over all the partition indexes to compute global statistics

  3) global statistics are propagated back to partition index



Adult, Spam, and Quality

- They used Yahoo!'s proprietary adult, spam, and document quality classifiers to post-process the ranked lists produced using Ivory.

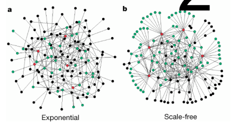


Test system

- NSF's CLuE (Cluster Exploratory) Program
- The Google/IBM Academic Cloud Computing Initiative
- 99 physical nodes;
  - each has two single-core processors (2.8 GHz)
  - 4 GB memory,
  - two 400 GB hard drives.

Compared with

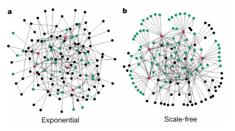
- A laptop with a 2.6 GHz Core 2 Duo processor
- 2 GB of RAM



Efficiency Results

Queries	Model	HDFS	local
Robust04	*bm25*	5.45s	8.25s
Robust04	QL	6.65s	10.0s
Web09	*bm25*	4.73s	6.65s
Web09	QL	5.60s	7.42s

Table 1: Average per-query running time on the first segment of ClueWeb09, comparing indexes stored on HDFS with indexes stored on local disk.



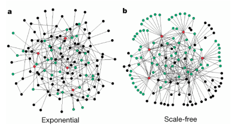
Brute Force and Indexed Approaches to Pairwise Document Similarity Comparisons with MapReduce

Jimmy Lin

The iSchool, College of Information Studies, University of Maryland

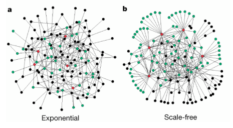
National Center for Biotechnology Information,

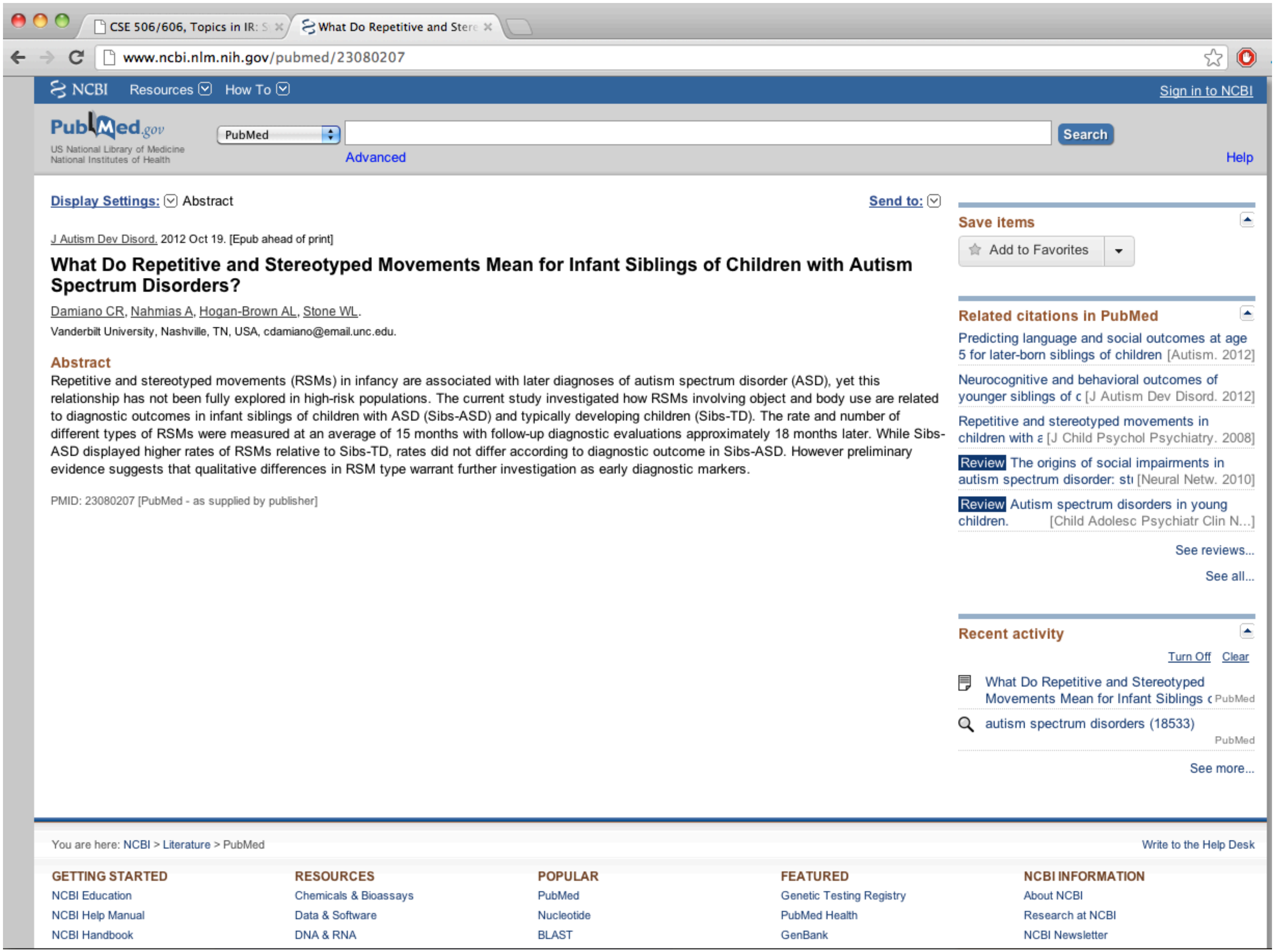
U.S. National Library of Medicine



Introduction

- Computing pairwise similarity on document collections:
  - clustering, unsupervised learning, and text retrieval
- “more like this” queries in PubMed
- PubMed:
  - Web-based search engine for the life sciences literature
  - National Library of Medicine (NLM)
  - PubMed provides access to MEDLINE: about 17 million bibliographic records (abstracts and metadata).





CSE 506/606, Topics in IR: S
What Do Repetitive and Stere

www.ncbi.nlm.nih.gov/pubmed/23080207

NCBI Resources How To Sign in to NCBI

PubMed.gov
US National Library of Medicine
National Institutes of Health

PubMed Search
Advanced Help

Display Settings: Abstract Send to:

J Autism Dev Disord. 2012 Oct 19. [Epub ahead of print]

What Do Repetitive and Stereotyped Movements Mean for Infant Siblings of Children with Autism Spectrum Disorders?

Damiano CR, Nahmias A, Hogan-Brown AL, Stone WL.

Vanderbilt University, Nashville, TN, USA, cdamiano@email.unc.edu.

Abstract

Repetitive and stereotyped movements (RSMs) in infancy are associated with later diagnoses of autism spectrum disorder (ASD), yet this relationship has not been fully explored in high-risk populations. The current study investigated how RSMs involving object and body use are related to diagnostic outcomes in infant siblings of children with ASD (Sibs-ASD) and typically developing children (Sibs-TD). The rate and number of different types of RSMs were measured at an average of 15 months with follow-up diagnostic evaluations approximately 18 months later. While Sibs-ASD displayed higher rates of RSMs relative to Sibs-TD, rates did not differ according to diagnostic outcome in Sibs-ASD. However preliminary evidence suggests that qualitative differences in RSM type warrant further investigation as early diagnostic markers.

PMID: 23080207 [PubMed - as supplied by publisher]

Save items
Add to Favorites

Related citations in PubMed
Predicting language and social outcomes at age 5 for later-born siblings of children [Autism. 2012]
Neurocognitive and behavioral outcomes of younger siblings of c [J Autism Dev Disord. 2012]
Repetitive and stereotyped movements in children with a [J Child Psychol Psychiatry. 2008]
Review The origins of social impairments in autism spectrum disorder: st [Neural Netw. 2010]
Review Autism spectrum disorders in young children. [Child Adolesc Psychiatr Clin N...]
See reviews...
See all...

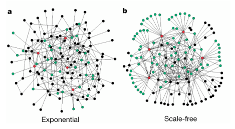
Recent activity
Turn Off Clear
What Do Repetitive and Stereotyped Movements Mean for Infant Siblings c PubMed
autism spectrum disorders (18533) PubMed
See more...

You are here: NCBI > Literature > PubMed Write to the Help Desk

GETTING STARTED	RESOURCES	POPULAR	FEATURED	NCBI INFORMATION
NCBI Education	Chemicals & Bioassays	PubMed	Genetic Testing Registry	About NCBI
NCBI Help Manual	Data & Software	Nucleotide	PubMed Health	Research at NCBI
NCBI Handbook	DNA & RNA	BLAST	GenBank	NCBI Newsletter

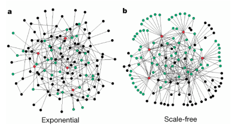
Consider a naive approach

- Indri was used “out of the box” to index a collection of 4.59m MEDLINE abstracts.
- Entire text of each abstract as a “query”, retrieving the top 100 hits takes about a minute per abstract on a laptop with a dual-core 2.6 GHz processor and a single hard drive in Windows XP.
- **77k machine hours for the entire collection!**
  - tremendous number of disk seeks required to look up postings
  - postings are repeatedly retrieved for common terms and then discarded



Approaches in this paper

- Three MapReduce algorithms:
  1) brute force
  2) large-scale ad hoc retrieval
  3) Cartesian product of postings lists
- Each algorithm supports one or more approximations that trade effectiveness for efficiency

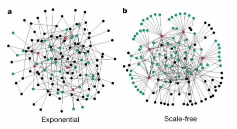


Document similarity algorithms

$$sim(d_i, d_j) = \sum_{t \in V} w_{t,d_i} \cdot w_{t,d_j}$$

for every document d in collection C, compute the top k ranking for similar documents according to a particular term weighting model.

In the case of the PubMed application k = 5

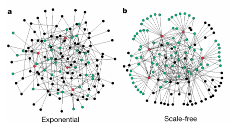


Brute Force

- simply compute the inner product of every document vector with every other document vector
- Divide the collection into blocks and compute document vectors (tokenization, stopword removal, etc.)
- Mapper computes the similarity score of all documents in the block with a single document d, emitting non-zero scores as intermediate key-value pairs.
- Intermediate output will contain scores for all documents in the block
- Reducer: all scores of a doc id are brought together

  the top k results are stored in a priority queue

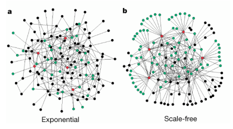
  emitted as the final output



Brute Force

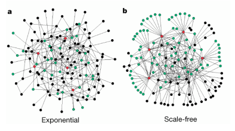
```
1: procedure $\textsc{Map}(a, d)$
2:     $[\langle b_1, e_1\rangle, \langle b_2, e_2\rangle, \ldots \langle b_n, e_n\rangle] \leftarrow \textsc{LoadDocuments}()$
3:     for all $\langle b, e\rangle \in [\langle b_1, e_1\rangle, \langle b_2, e_2\rangle, \ldots \langle b_n, e_n\rangle]$ do
4:         $s \leftarrow \textsc{ComputeScore}(d, e)$
5:         if $s > 0$ then
6:             $\textsc{Emit}(b, \langle a, s\rangle)$

1: procedure $\textsc{Reduce}(b, [\langle a_1, s_1\rangle, \langle a_2, s_2\rangle \ldots])$
2:     $\textsc{Initialize.PriorityQueue}(Q)$
3:     for all $\langle a, s\rangle \in [\langle a_1, s_1\rangle, \langle a_2, s_2\rangle \ldots]$ do
4:         if $Q.\textsc{Size}() < k$ then
5:             $Q.\textsc{Insert}(a, s)$
6:         else if $s > Q.\textsc{Min}()$ then
7:             $Q.\textsc{ExtractMin}()$
8:             $Q.\textsc{Insert}(a, s)$
9:     $\textsc{Emit}(b, Q.\textsc{ExtractAll}())$
```

Figure 2: Pseudo-code of brute force (BF) algorithm for pairwise similarity in MapReduce.



Optimized brute force approach

- Reducing the number of intermediate key-value pairs
- Instead of emitting all non-zero document scores between every d and every e, the mapper can keep track of top k scoring documents for every e (with a priority queue).
- Once the mapper is finished processing all ⟨a, d⟩ input pairs, the contents of the priority queue can be emitted as intermediate key-value pairs.
- Moving part of the reducer functionality into the mapper, dramatically reducing the size of the intermediate output and the time spent shuffling key-value pairs across the network.

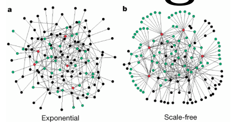


Inverted Indexing

```
1: procedure Map($a, d$)
2:     Initialize.AssociativeArray($H$)
3:     for all $t \in d$ do
4:         $H\{t\} \leftarrow H\{t\} + 1$
5:     for all $t \in H$ do
6:         Emit($t, \langle a, H\{t\} \rangle$)
```

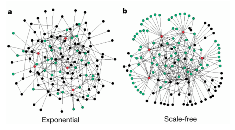
```
1: procedure Reduce($t, [\langle a_1, f_1 \rangle, \langle a_2, f_2 \rangle \ldots]$)
2:     Initialize.List($P$)
3:     for all $\langle a, f \rangle \in [\langle a_1, f_1 \rangle, \langle a_2, f_2 \rangle \ldots]$ do
4:         Append($P, \langle a, f \rangle$)
5:     Sort($P$)
6:     Emit($t, P$)
```

Figure 3: Pseudo-code of the inverted indexing algorithm in MapReduce.



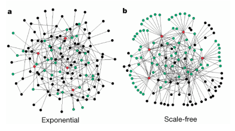
Inverted Index: parallel queries (PQ) Mapper

- Treat pairwise similarity comparisons as a very large ad hoc retrieval problem.
- Divide up the collection into individual blocks of “queries”
- Mapper input: term t and its postings list P
- Mapper loads all the queries at once.
- If the query contains t: for each posting compute partial contribution ($w_{t,q} \cdot w_{t,d}$) and store in associative array H
- Mapper emits query i as the key and H as the value.
- The result of each mapper: partial query-document scores associated with term t for all queries that contain the term.



Inverted Index: parallel queries (PQ) Reducer

- The reducer performs an element-wise sum of all the associative arrays (merge)
- This adds up the contributions for each query term across all documents.
- The final result is an associative array holding complete query-document scores
- This algorithm replaces random access to the postings with a parallel scan of all postings.
- In processing a set of queries, each postings list is accessed only once

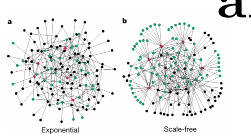


Inverted Index: parallel queries (PQ)

```
procedure $\textsc{Map}(t, P)$
    $[Q_1, Q_2, \ldots Q_n] \leftarrow \textsc{LoadQueries}()$
    for all $Q_i \in [Q_1, Q_2, \ldots Q_n]$ do
        if $t \in Q_i$ then
            $\textsc{Initialize.AssociativeArray}(H)$
            for all $\langle a, f \rangle \in P$ do
                $H\{a\} \leftarrow w_{t,q} \cdot w_{t,d}$
            $\textsc{Emit}(i, H)$
```

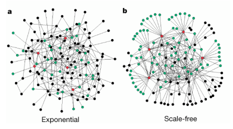
```
procedure $\textsc{Reduce}(i, [H_1, H_2, H_3, \ldots])$
    $\textsc{Initialize.AssociativeArray}(H_f)$
    for all $H \in [H_1, H_2, H_3, \ldots]$ do
        $\textsc{Merge}(H_f, H)$
    $\textsc{Emit}(i, H_f)$
```

Figure 4: Pseudo-code of the parallel queries (PQ) algorithm in MapReduce.



Inverted Index: postings Cartesian product (PCP)

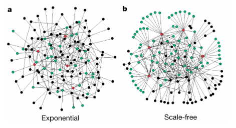
- The basic idea: to generate partial score contributions for a particular term by taking the Cartesian product of each postings list with itself;
- The resulting document pairs account for all similarity comparisons in which that term contributes.
- Mapper implements an outer loop and an inner loop
- For each posting in the outer loop, partial score contributions for each posting in the inner loop stored in an associative array (serving as accumulators)
- intermediate result for a_i for term t



Inverted Index: postings Cartesian product (PCP)

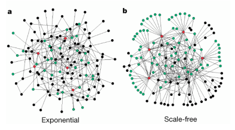
```
1: procedure MAP($t, P$)
2:     for all $\langle a_i, f_i \rangle \in P$ do
3:         INITIALIZE.ASSOCIATIVEARRAY($H$)
4:         for all $\langle a_j, f_j \rangle \in P$ do
5:             if $a_i \neq a_j$ then
6:                 $H\{a_j\} \leftarrow w_{t,a_i} \cdot w_{t,a_j}$
7:         EMIT($a_i, H$)

1: procedure REDUCE($a_i, [H_1, H_2, H_3, \ldots]$)
2:     INITIALIZE.ASSOCIATIVEARRAY($H_f$)
3:     for all $H \in [H_1, H_2, H_3, \ldots]$ do
4:         MERGE($H_f, H$)
5:     EMIT($a_i, H_f$)
```

Figure 5: Pseudo-code of the postings Cartesian product (PCP) algorithm in MapReduce.



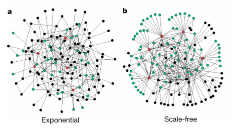
Inverted Index: postings Cartesian product (PCP)

- PCP algorithm cannot be directly implemented. It attempts to process the entire collection at once, and generates far too much intermediate output.
- Pairwise similarity is computed on a block of documents at a time, by modifying the outer loop in line 2 of the mapper.
- PCP and PQ generate the same intermediate output
- the restrictions imposed on the outer loop in the PCP algorithm serve the same as in the PQ mapper
- The differences in control logic, however, have an effect on algorithm running time.
- The two algorithms support different approximations



Approximations

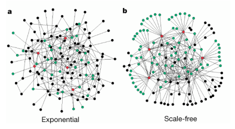
- PQ and PCP: accumulator limit
  - postings are frequency-sorted
- BF and PQ: term limit
  - top n terms with lowest df
- PQ and PCP: df limit
  - ignored terms with df > threshold



Experiments

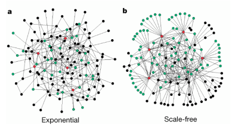
Setting	Effectiveness	Efficiency
0: BF exact	0.3993, 0.2230	862m, 1058m
1: PQ a80k	−1.0%, −1.3%	489m, 619m
2: PCP a80k	−1.0%, −1.3%	460m, 586m
3: BF t80	−0.2%, −0.7%	638m, 757m
4: PQ a80k,t80	−1.9%, −1.9%	455m, 578m
5: PCP a80k,df600k	−1.7%, −1.2%	408m, 524m
6: PQ a80k,t80,df600k	−2.3%, −1.7%	389m, 491m
7: PCP a60k	−2.1%, −3.7%	371m, 506m
8: PQ a80k,t60	−4.3%, −3.7%	338m, 448m
9: PCP a80k,df200k	−3.6%, −4.8%	258m, 339m

Table 1: Summary of algorithms at key settings. Effectiveness reported as relative differences in micro, macro P5; efficiency reported as 99%, 100% completion times. (a=accumulator, t=term, df=*df*)



Discussion

- First, if exact similarity is desired, the brute force approach is the most efficient of the three algorithms.
- Second, by limiting accumulators, both the PQ and PCP algorithms can achieve large efficiency gains without significant decreases in effectiveness.
- Third, additional gains in efficiency can be achieved with the term and df limits, although there is overlap in the computations they save.

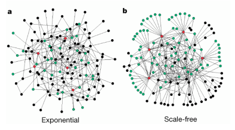


Web-Scale Distributional Similarity and Entity Set Expansion

Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, Vishnu Vyas

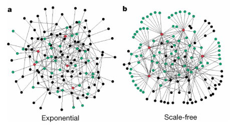
Yahoo! Labs

Yandex Labs



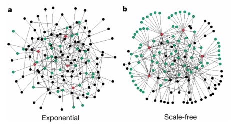
Computing semantic similarity between terms

- word classification and WSD
- context spelling correction
- fact extraction
- semantic role labeling
- query expansion
- textual advertising
- This paper deployed over a 200B word crawl of the Web



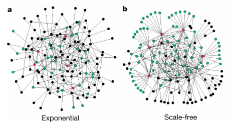
Large-Scale Similarity Model

- Term semantic models: the meaning of terms linked to their context
- Differences in methods:
  - Context: text window, syntactic relations
  - How to weigh contexts: frequency, TF/IDF, PMI
  - measuring the similarity: Cosine, Dice



Paper's methodology for computing term similarity

- Their web crawls, are POStagged and chunked
- Terms: NP chunks with some modifiers removed
- Terms contexts: rightmost and leftmost stemmed chunks
- They weigh each context f using pointwise mutual information

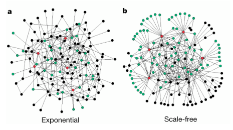


Input: Two matrices *A* and *B* of feature vectors.

```
## Build an inverted index for A (optimiza-
## tion for data sparseness)
AA = an empty hash-table
for i in (1..n):
   F2[i] = f2(A[i]) ## cache values of $f_2(x)$
   for k in non-zero features of A[i]:
      if k not in AA: AA[k] = empty-set
      ## append <vector-id, feature-value>
      ## pairs to the set of non-zero
      ## values for feature k
      AA[k].append( (i,A[i,k]) )
## Process the elements of B
for b in B:
   F1 = {} ## the set of $A_i$ that have non-
        zero similarity with b
   for k in non-zero features of b:
      for i in AA[k]:
         if i not in sim: sim[i] = 0
         F1[i] += f1( AA[k][i], b[k])
   F3 = f3(b)
   for i in sim:
      print i, b, f0( F1[i], F2[i], F3)
```

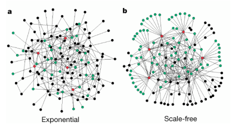
Output: A matrix containing the similarity between all elements in *A* and in *B*.

Figure 1. Similarity computation algorithm.



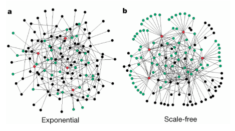
MapReduce

- M×N Map tasks: 1/Mth part of A, 1/Nth part of B
- Each part of A is processed N times, and each part of B is processed M times.
- M is determined by the amount of memory dedicated for the inverted index,
- N trade off: As N increases, more parallelism can be obtained at the increased cost of building the same inverse index N times.
- Reducer groups the output by bi



Application to Set Expansion

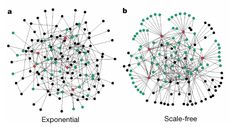
- Task Definition: Given a set of seed entities S = {s1, s2, …, sk} of a class C = {s1, s2,…, sk, …, sn} and an unlabeled textual corpus T, find all members of the class C.
- Bottled Water Brands:
  - S = {Volvic, San Pellegrino, Gerolsteiner Brunnen, Bling H2O},
  - our task is to find all other members of this class, such as {Agua Vida, Apenta, Culligan, Dasani, Ethos Water, Iceland Pure Spring Water, Imsdal, …}



Set Expansion Algorithm

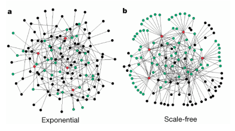
- Given the seed elements {Volvic, San Pellegrino, Gerolsteiner Brunnen, Bling H2O},
- Resulting centroid consists :

  water, monitor, lake, water, take over,
- A scored and ranked set for expansion is ultimately generated by sorting all terms according to their similarity to the seed set centroid, and applying a cutoff



Evaluation Methodology

- Gold Standard Entity Sets (GS)
  - “List of” pages in Wikipedia
  - “List of World War II aces from Denmark”
  - “List of people who claimed to be God”
- Trials 23 seed sizes (1, 2, 5, 10, 20,… 200)
  - 30 copies 50 sets = 1500 (20,220 trials)
- Judgments
  - each of the 20,220 trials expanded
  - Then judged as correct or incorrect against the GS

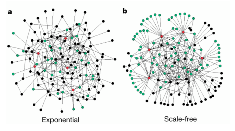


Experimental Setup

Table 2. Corpora used to build our expansion models.

Corpora	Unique Sentences (Millions)	Tokens (Millions)	Unique Words (Millions)
Web100	5,201	217,940	542
Web020†	1040	43,588	108
Web004†	208	8,717	22
Wikipedia[6]	30	721	34

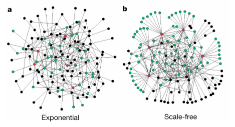
†Estimated from *Web100* statistics.



Quantitative Analysis

- For Web100: pairwise similarity between over 500 million words in 50 hours using 200 four-core machines.

- Web004 over two hours all the terms whereas vs Ravichandran et al. (2005) 570 hours 73% of the top-1000 similar terms



Corpus Size and Corpus Quality Effect

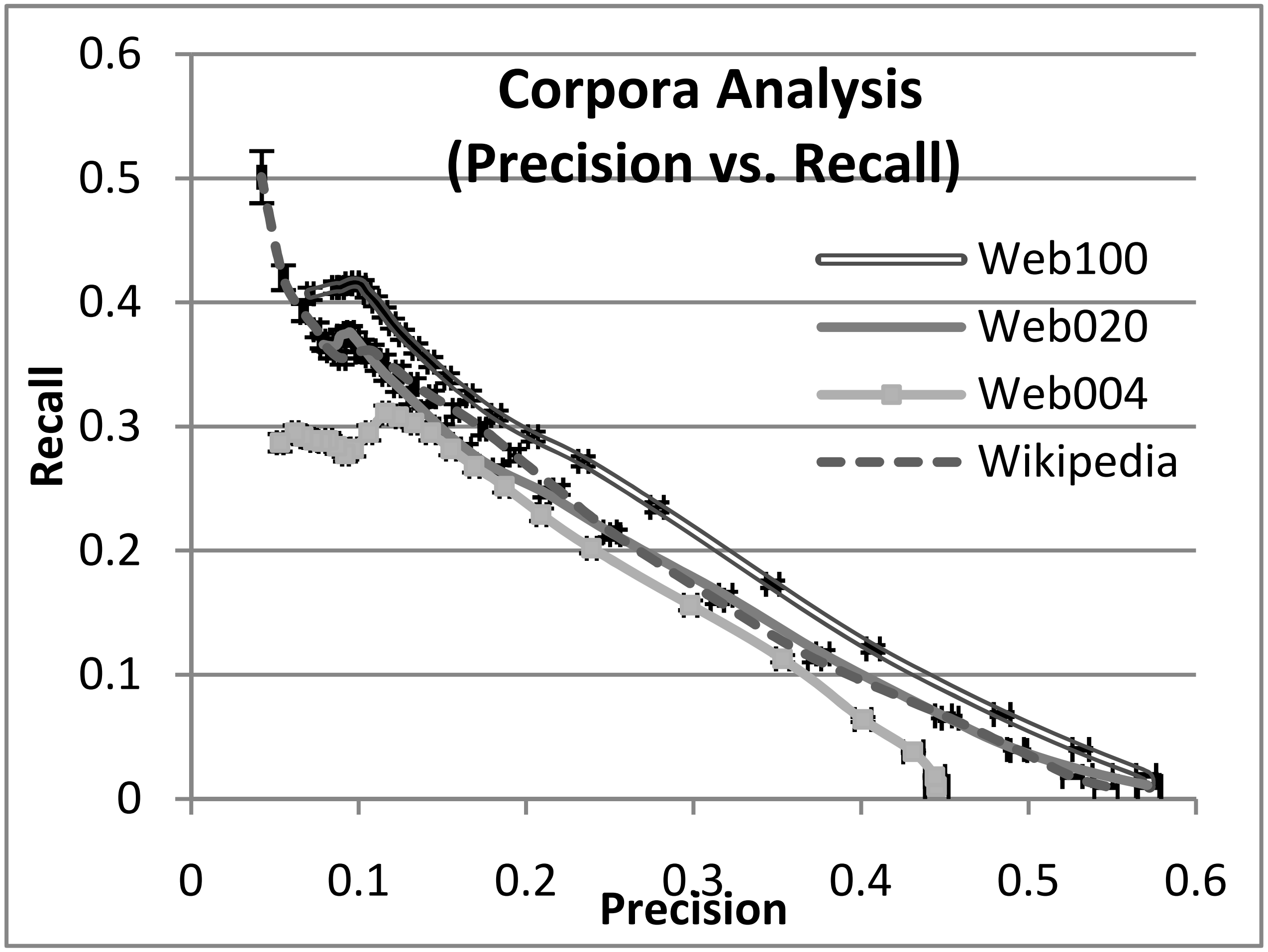
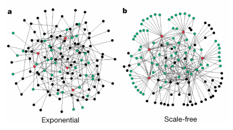
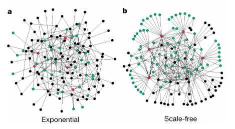


Figure 2. Corpus size and quality improve performance.



Other observations

- Seed set composition greatly affects system performance (with 30 different seed samples of size 10)
- Increasing the seed set size beyond 20 or 30 on average does not find any new correct instances.
- But error rate does not increase with increased seed set size



Questions?