Efficient Query Processing in Web Search Engines

Simon Lia-Jonassen
UiO, 12/04/16

Outline

- Introduction and Motivation
- Query Processing and Optimizations
- Caching
- Parallel and Distributed Query Processing
- Hardware Trends and Their Impact

Search Engines

Popular Science, August 1937.

World's Largest File Has Traveling Desks

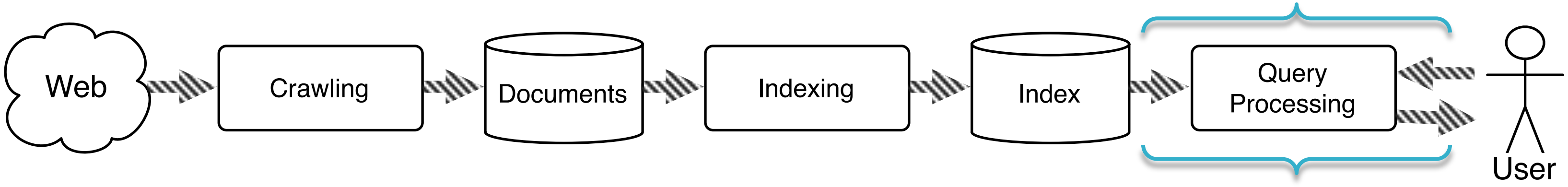
CLERKS ride up and down on "elevator desks" to consult the gigantic correspondence file of a Czechoslovakian insurance organization. Said to be the largest in the world, the huge letter file consists of 3,000 drawers, each ten feet long, covering 4,000 square feet of wall space and extending sixteen feet up from the floor. Clerks press control levers to move their desks up, down, or sidewise until they reach the desired file drawer, which opens automatically.

Clerks examining correspondence in the mammoth file of an insurance company

© pro.clubic.com

- A4 page is ~.005 cm thick. 3000 drawers x 304.8 cm/drawer x 200 pages/cm gives **182.9 million** pages. A4 page fits 350 – 400 words.
- Indexed Web contains at least **4.57 billion** pages. An avg. page is 478 words.
- Google has more than 1 billion global visitors and annually spends billions of USD on data centers.

Web-Search in a Nutshell



- Query processing basics
  - Given an indexed collection of textual documents D and a textual query q, find the k documents with largest similarity score.

- Some similarity model alternatives
  - Boolean model.
  - Vector space models, e.g., TFxIDF, Okapi BM25.
  - Term proximity models.
  - Link analysis models, e.g., PageRank, HITS, Salsa.
  - Bayesian and language models.
  - Machine learned models.

Effectiveness (quality and relevance)

- Boolean, not ranked
  - Precision – how many selected items are relevant.
  - Recall – how many relevant items are selected.
  - F1 – weighted harmonic mean of precision and recall.

- Boolean, ranked
  - Precision and recall at 5, 10, …k.
  - Average Precision and Mean Average Precision.

- Graded, ranked
  - Cumulative Gain (CG), Discounted CG (DCG) and Normalized DCG (NDCG).
  - Mean Reciprocal Rank (MRR).

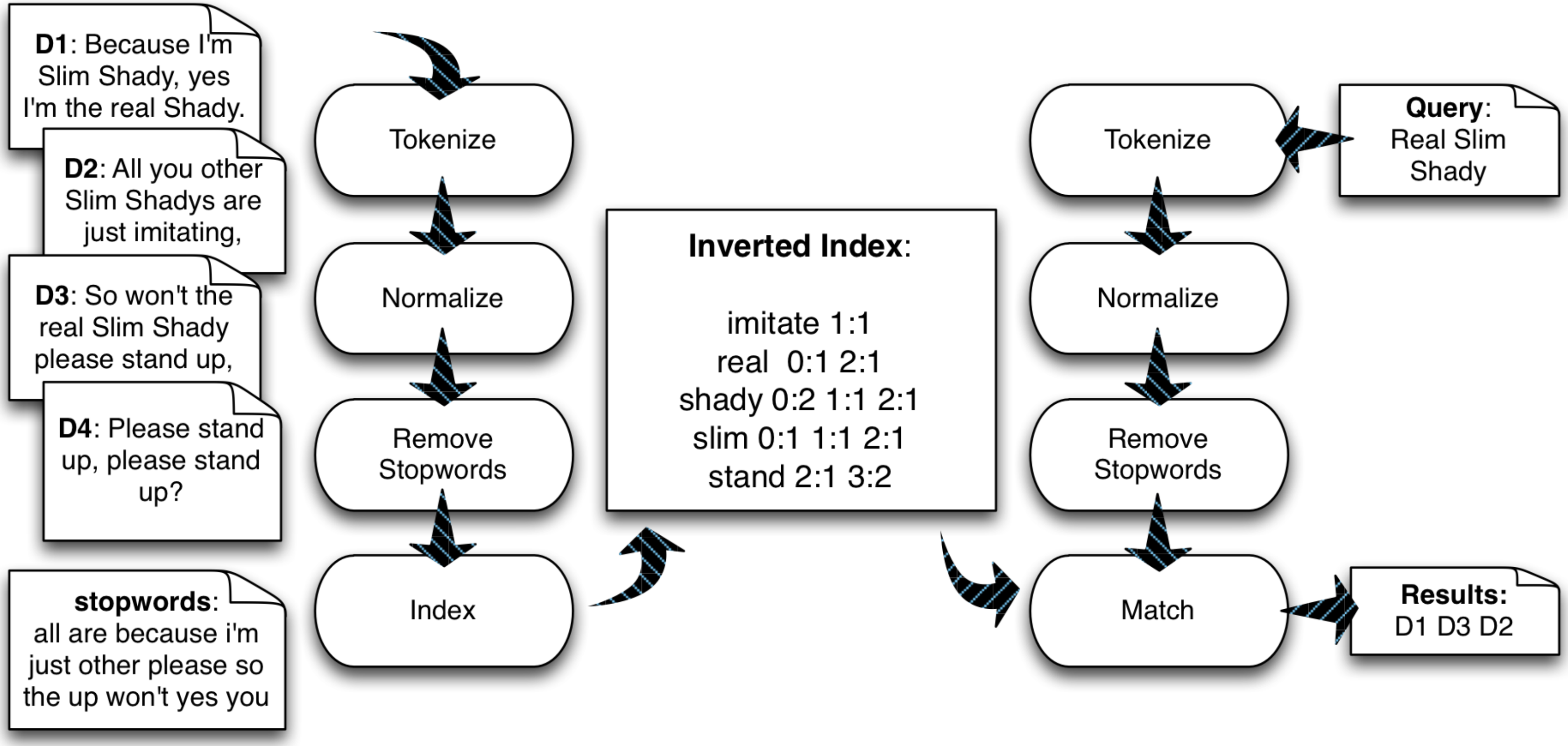
Efficiency (volume, speed and cost)

- Index and Collection Size
  - Number of documents and document size.
  - Index granularity, size and freshness.

- Infrastructure Cost
  - Number of machines, memory/disk size, network speed, etc.
  - Resource utilization and load balancing.

- Queries and Time
  - Latency – time per query.
  - Throughput – queries per time unit.
  - Query volume (offline) and arrival rate (online).
  - Query degradation rate (online).
  - Result freshness (cached results).

Query Processing with Inverted Index



D1: Because I'm Slim Shady, yes I'm the real Shady.

D2: All you other Slim Shadys are just imitating,

D3: So won't the real Slim Shady please stand up,

D4: Please stand up, please stand up?

stopwords: all are because i'm just other please so the up won't yes you

Inverted Index:

imitate 1:1
real 0:1 2:1
shady 0:2 1:1 2:1
slim 0:1 1:1 2:1
stand 2:1 3:2

Query: Real Slim Shady

Results: D1 D3 D2

Query Processing with Inverted Index

$$\text{Score}(d,q) = \sum_{t \in q} w_{t,q} \cdot w_{t,d}$$

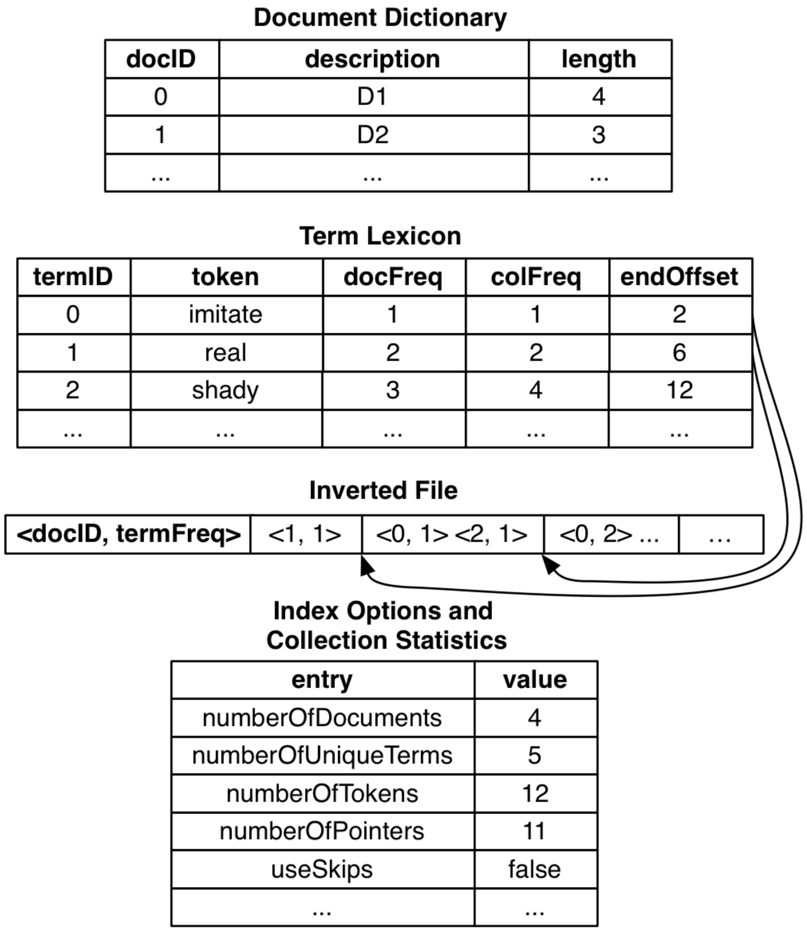
$$w_{t,q} = \frac{(k_3+1)\cdot f_{t,q}/\max(f_{t,q})}{k_3 + f_{t,q}/\max(f_{t,q})}$$

$$w_{t,d} = \text{TF}(t,d)\cdot \text{IDF}(t)$$

$$\text{TF}(t,d) = \frac{f_{t,d}\cdot(k_1+1)}{f_{t,d} + k_1\cdot((1-b)+b\cdot l_d/l_{avg})}$$

$$\text{IDF}(t) = \log(\frac{N - f_t + 0.5}{f_t + 0.5})$$

We could use $k_1 = 1.2$, $b = 8$, $k_3 = 0.75$, but note that IDFs would be quite strange in this particular example.



Document Dictionary

docID	description	length
0	D1	4
1	D2	3
...	...	...

Term Lexicon

termID	token	docFreq	colFreq	endOffset
0	imitate	1	1	2
1	real	2	2	6
2	shady	3	4	12
...	...	...	...	...

Inverted File

<docID, termFreq>	<1, 1>	<0, 1> <2, 1>	<0, 2> ...	…

Index Options and Collection Statistics

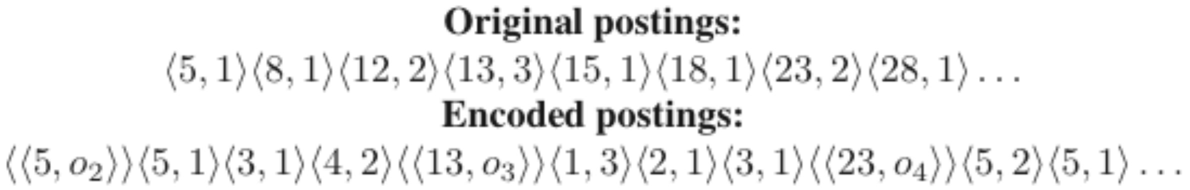
entry	value
numberOfDocuments	4
numberOfUniqueTerms	5
numberOfTokens	12
numberOfPointers	11
useSkips	false
...	...

Posting Lists (*doc. ids and frequencies*)

- Usually represented as one or two integer lists ordered by document id.
  - Alternatives:
    - bitmaps instead of document ids, weights instead of frequencies, or impact ordered lists.

- For better compression document ids are replaced by differences (*deltas*), and the ids can also be remapped/reordered to minimize resulting deltas.

- Usually accessed via an *iterator* (*get* and *next*) and an auxiliary index/set of pointers can be used to *skip* forward in the inverted file.



Original postings:

$\langle 5,1\rangle\langle 8,1\rangle\langle 12,2\rangle\langle 13,3\rangle\langle 15,1\rangle\langle 18,1\rangle\langle 23,2\rangle\langle 28,1\rangle\ldots$

Encoded postings:

$\langle\langle 5,o_2\rangle\rangle\langle 5,1\rangle\langle 3,1\rangle\langle 4,2\rangle\langle\langle 13,o_3\rangle\rangle\langle 1,3\rangle\langle 2,1\rangle\langle 3,1\rangle\langle\langle 23,o_4\rangle\rangle\langle 5,2\rangle\langle 5,1\rangle\ldots$

Posting List Compression

- Bit-Aligned Methods
  - E.g., Unary, Elias, Golomb, Rice, Interpolative, etc.
  - Space efficient but require less efficient bit arithmetic.

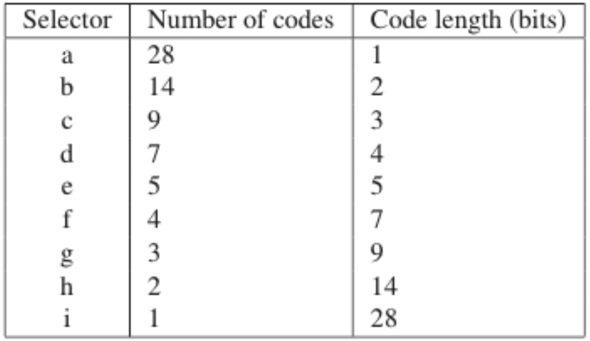
Value	Unary	Elias-γ
1	0	0
2	10	10 0
3	110	10 1
4	1110	110 00
5	11110	110 01
6	111110	110 10
7	1111110	110 11
8	11111110	1110 000
9	111111110	1110 001
10	1111111110	1110 010

- Byte-Aligned Methods
  - E.g., VByte (VB)

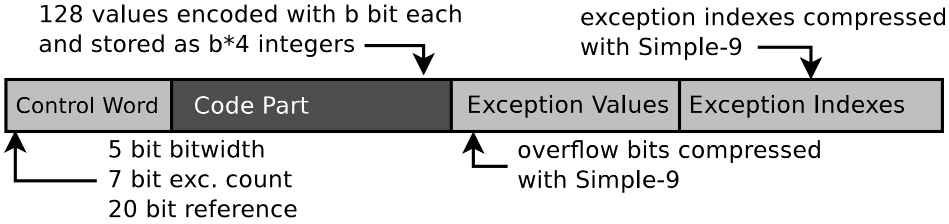
    1 :1000 0001
    13 :1000 1101
    133 :0000 0001 1000 0101
    1337 :0000 1010 1011 1001
  - Fast but less space efficient.

Posting List Compression

- Word-Aligned Methods
  - E.g., Simple9 (left) and NewPFoR (right)



Selector	Number of codes	Code length (bits)
a	28	1
b	14	2
c	9	3
d	7	4
e	5	5
f	4	7
g	3	9
h	2	14
i	1	28



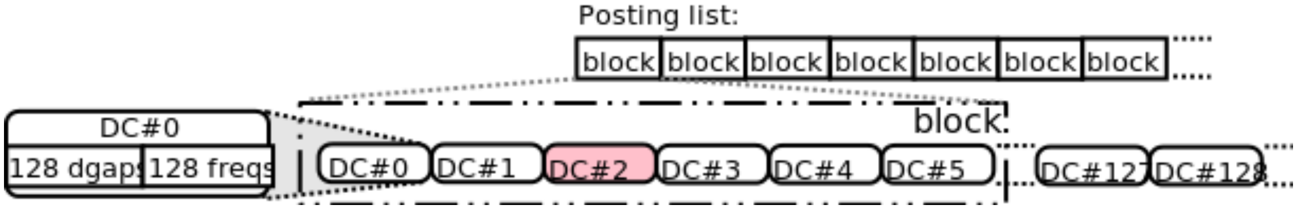
128 values encoded with b bit each
and stored as b*4 integers

exception indexes compressed
with Simple-9

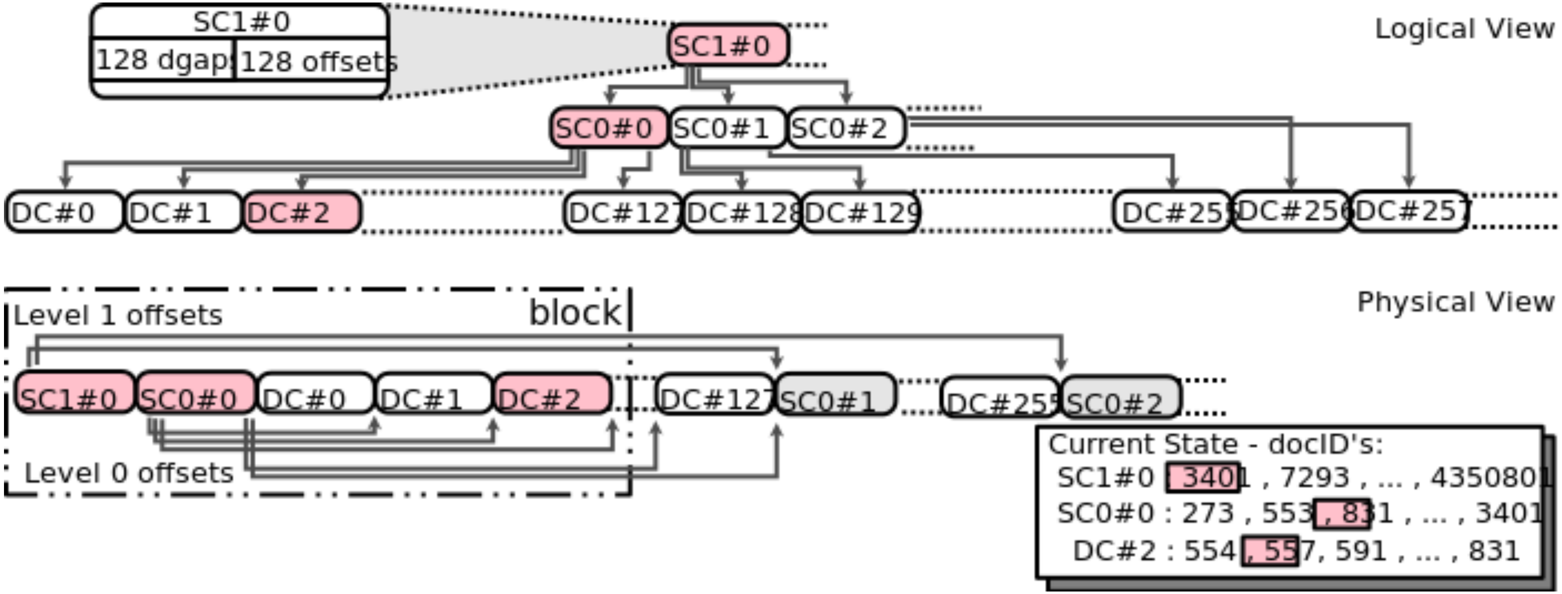
5 bit bitwidth
7 bit exc. count
20 bit reference

overflow bits compressed
with Simple-9

  - Branch-free and superscalar, can be tuned for compression/speed.
  - In the rest, we consider NewPFoR with chunks of 128 deltas followed by 128 frequencies.
    - Alternatives: 32/64 or variable size chunks.



Posting List Layout with Skips



Query Evaluation and Pruning

- Disjunctive (OR) and Conjunctive (AND) Evaluation
  - OR – The documents that match <u>any</u> query term will be considered.
    - Slower, lower precision, higher recall.
  - AND – Only the documents that match <u>all</u> query terms will be considered.
    - Faster, higher precision, lower recall.

- Static Pruning – aka *Index Pruning*
  - Term Pruning – remove less important terms from the index.
  - Document Pruning – remove less important documents from the index.

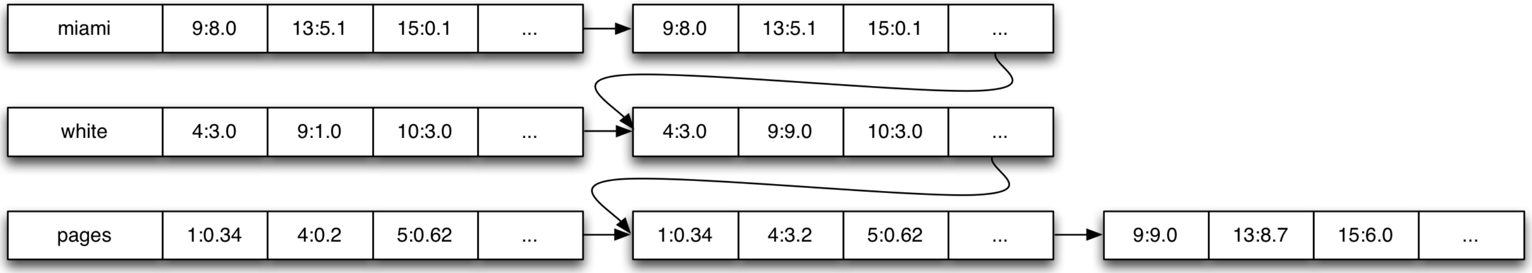
- Dynamic Pruning – aka *Early Exit Optimization*
  - Avoid scoring postings for some of the candidates under evaluation.
  - ✓ Safe pruning – guarantees the same result as a full evaluation (*score*, *rank or set safe*).
  - ✓ Unsafe pruning – does not guarantee the same result as a full evaluation.

Term- vs Document-At-A-Time Evaluation

- Term-At-A-Time (DAAT)
  - Score all postings in the same list and merge with the partial result set (*accumulators*).
  - ✓ Need to iterate only one posting list at a time, but store all possible candidates.

- Document-At-A-Time (DAAT)
  - Score all postings for the same document and add to the result head.
  - ✓ Need to iterate all posting lists at once, but store only the *k* best candidates.

Term-At-A-Time Evaluation (TAAT)



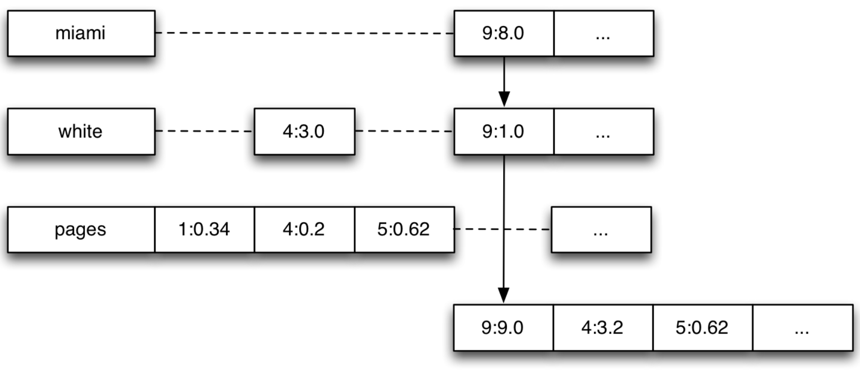
miami | 9:8.0 | 13:5.1 | 15:0.1 | ... → 9:8.0 | 13:5.1 | 15:0.1 | ...

white | 4:3.0 | 9:1.0 | 10:3.0 | ... → 4:3.0 | 9:9.0 | 10:3.0 | ...

pages | 1:0.34 | 4:0.2 | 5:0.62 | ... → 1:0.34 | 4:3.2 | 5:0.62 | ... → 9:9.0 | 13:8.7 | 15:6.0 | ...

- Some optimizations
  - AND – Evaluate query in conjunctive mode, starting with shortest posting list.
  - Quit/Continue – Stop adding new accumulators when a target size is reached.
  - Lester – Dynamically scale a frequency threshold based on the accumulator set size.
  - MaxScore – Ignore postings and remove candidates for documents that are guaranteed not to be in the result set (*need to track the k best candidates seen so far*).

Document-At-A-Time Evaluation (DAAT)



- Some optimizations
  - And – Don’t score unless all pointers align, skip to the largest document id.
  - MaxScore – Stop candidate evaluation when it is guaranteed not to be in the result set.
  - WAND – Similar to MaxScore but a bit different (see the next slide).

DAAT MaxScore and WAND

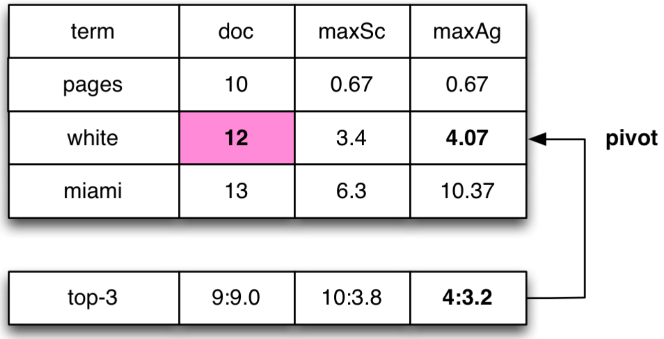
- MaxScore
  - Evaluate lists in maxScore order.
  - Don't advance optional term pointers.
    - Must match at least one term with maxAg >= thres.
  - Discard when score + maxAg < thres.

term	doc	maxSc	maxAg	**optional**
miami	13	6.3	10.37	false
white	**12**	3.4	**4.07**	false
pages	10	0.67	0.67	true

top-3	9:9.0	10:3.8	**4:3.2**

s(white@12): 1.0 + 0.67 = 1.67 (<3.2 --> discard!)

- WAND
  - Evaluate lists in the current document id order.
  - Pivot by first pointer with maxAg >= thres.
  - If first pointer and pivot are equal, evaluate. Otherwise, skip using the pivot.



term	doc	maxSc	maxAg
pages	10	0.67	0.67
white	**12**	3.4	**4.07**
miami	13	6.3	10.37

top-3	9:9.0	10:3.8	**4:3.2**

✓ Both methods can be improved with block-wise maximum scores.

Post-processing and Multistep Processing

- Traditional post-processing
  - Generate snippets.
  - Retrieve page screenshots.
  - Result clustering and/or faceting.

- Two-phase processing
  1. Use a simple ranking model to generate a pool of candidates.
  2. Refine the results using a more expensive model.
     - E.g., machine learned model, phrase matching, etc.

- Query refinement
  - Suggest another query based on the found results.

Caching

- Two-level (*Saraiva et al.*)
  - Result cache and posting list cache.
- Three-level (*Long and Suel*)
  - Adds intersection cache.
- Five-level (*Ozcan et al.*)
  - Adds document cache and score cache.
- Other alternatives
  - Projections instead of intersections.
  - Blocks instead of posting lists.
- Static, dynamic and hybrid caches
  - Static – rebuilt once in a while.
  - Dynamic – updated on each access.
  - Static-dynamic – splits the cache into a static part and a dynamic part.

R. Ozcan et al. / Information Processing and Management 48 (2012) 828–840

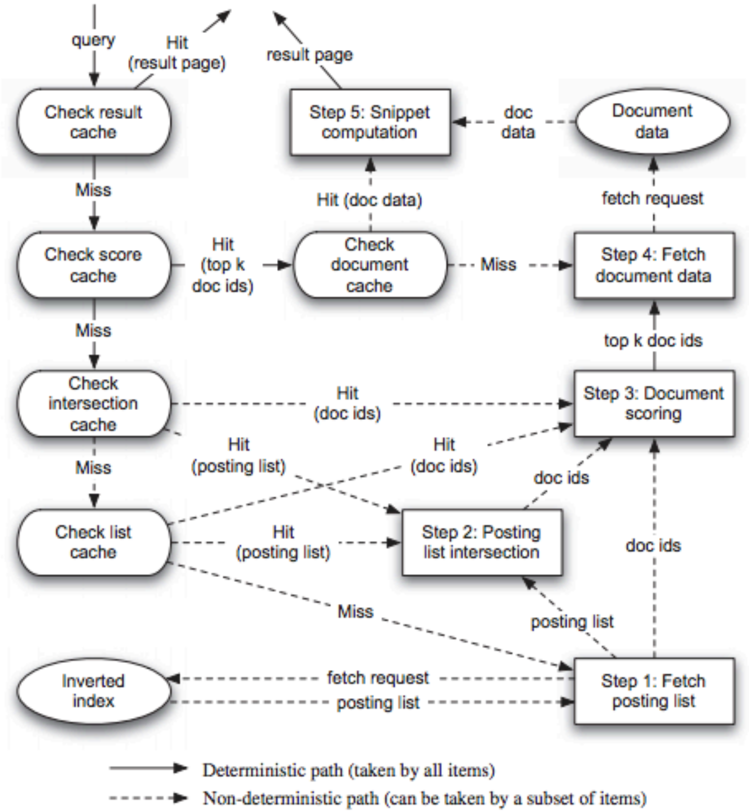
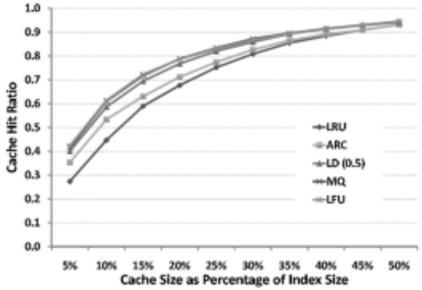


Fig. 3. The workflow used by the simulator in query processing.

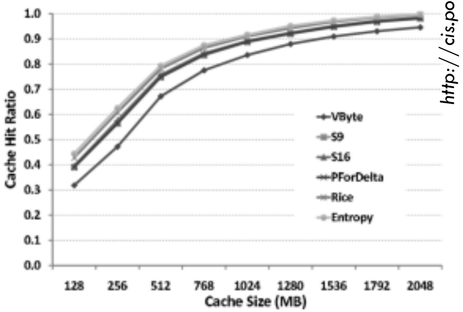
Cache Replacement Strategies

- Some of the most common methods
  - LRU – put newly accessed entries into a queue, discard from the tail.
  - LFU – each entry has a counter, discard entries with lowest count.
    - Other alternatives: Two-Stage LRU, LRU-2, 2Q, Multi-Queue (MQ), Frequency-Based Replacement (FBR), Adaptive Replacement Cache (ARC).
  - Cost – evict entries with the lowest processing cost.
  - Landlord
    - assign H=cost/size on insertion or hit
      - cost = 1 gives most hits, cost = proc. time gives best avg. latency.
    - evict the one with $\min_H$ and subtract it from remaining H.
    - Improvement: use H + alpha * remaining_H on the first renewal, and alpha’ on the second and subsequent renewals.

Comparison of Caching Policies:



Impact of Compression:

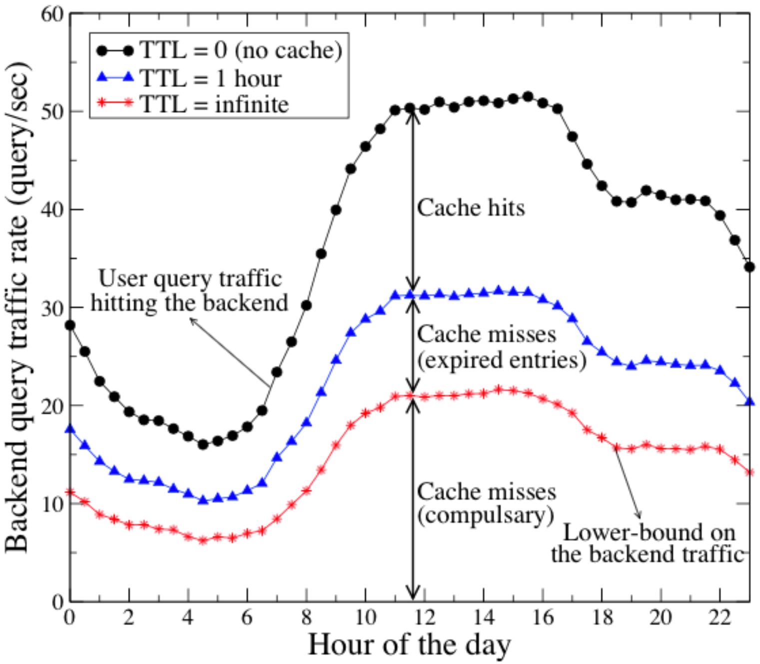


http://cis.poly.edu/cs912/indexcomp.pdf

Freshness with Infinite Result Cache

- Motivation
  - We can cache results infinitely but they will become outdated after a while.

- A solution and the improvements
  1. Set a TTL to invalidate the results after a while.
     - Always rerun the expired queries on request.
  2. Refresh cache results when have capacity.
     - May improves average result freshness.
  3. Proactively refresh expensive queries that are likely to be expired at peak-load hours.
     - May improves result freshness, degradation and latency, but determining such queries is really hard.



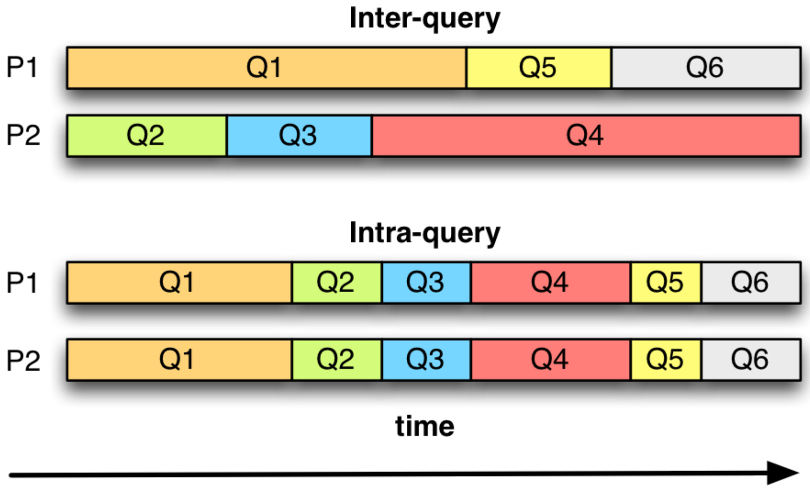
Online and Offline Query Processing

- Offline
    - All queries are available at the beginning.
    - Query throughput must be maximized, individual query latency is unimportant.
    - Specific optimizations
        - Query reordering, clairvoyant intersection and posting list caching, etc.

- Online
    - Queries arrive at different times and the volume varies during the day.
    - Throughput must be sustained and individual query latency minimized.
    - Specific optimizations
        - Degradation, result prefetching, micro-batching, etc.

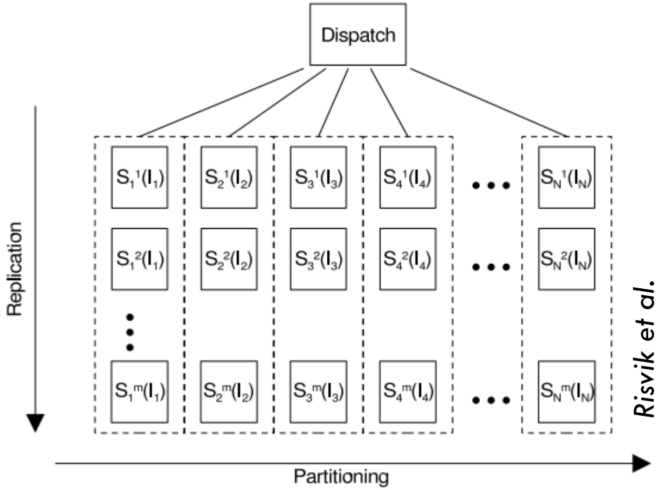
Parallel Query Processing

- Motivation
  - Modern CPUs are multi-core and we would like to utilize this in the best possible way.
- Inter-query concurrency
  - – assign different queries to different cores.
  - ✓ Improves throughput, affects latency.
- Intra-query concurrency
  - – assign different blocks to different cores.
  - ✓ Improves latency, affects throughput.
- Some of the main issues
  - Memory wall (I/O bottleneck).
  - CPU cache related issues
    - Coherence, conflicts, affinity, etc.
  - Amdahl’s and Gustafson's laws.

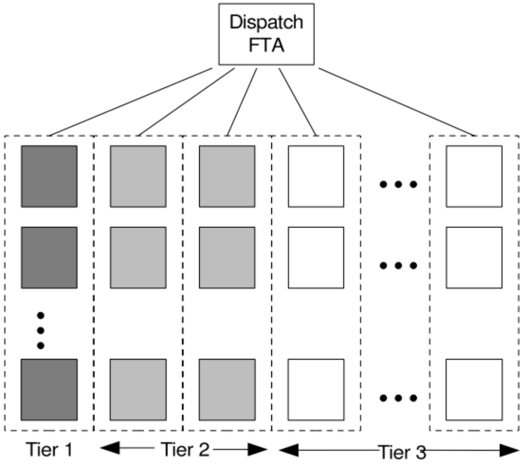


Distributed Query Processing

- Partitioning and Replication
  - ✓ Add more partitions to scale with the collection size.
  - ✓ Add more replicas to scale with the query volume.
- Clustering
  - ✓ Group similar documents in clusters.
  - ✓ Build a different index for each cluster.
- Tiering
  - ✓ Split the collection into several parts, e.g.
    - 1 mil important documents.
    - 10 mil less important documents.
    - 100 mil not really important documents.
  - ✓ Decide when a query should to *fall through*.

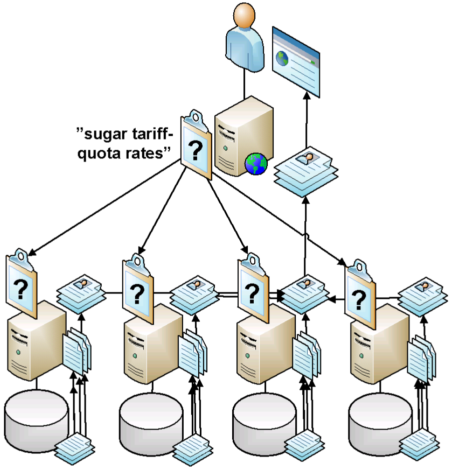


Risvik et al.

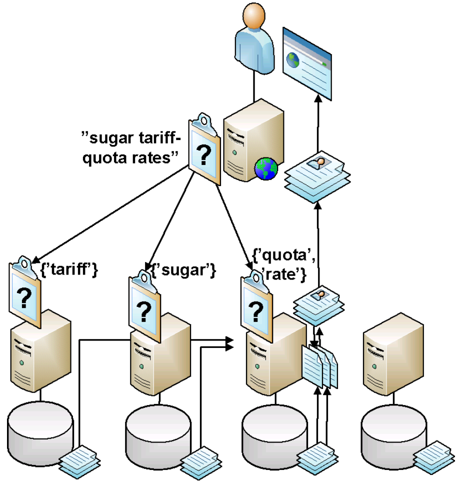


Partitioned Query Processing

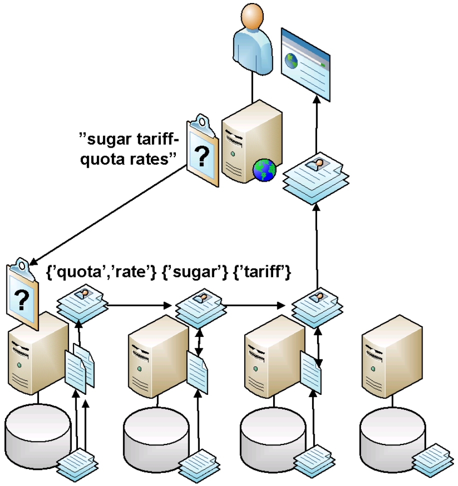
- Document-wise partitioning
  - Each node has a subset of documents.
  - Each query is processed by all of the nodes in parallel, one of the nodes merges the results.
    - ✓ Pros: Simple, fast and scalable.
    - ✓ Cons: Each query is processed by all of the nodes.
    - ✓ Cons: Large number of posting lists to be processed.



- Term-wise partitioning
  - Each node has a subset of terms.
  - Posting lists are fetched and sent to a node that processes all of them.
    - ✓ Pros: Only a few nodes are involved.
    - ✓ Pros: Possibility for inter-query concurrency.
    - ✓ Pros: Only one posting list for each term.
    - ✓ Cons: Only one node does all processing.
    - ✓ Cons: Network load and load balancing are critical.



Partitioned Query Processing



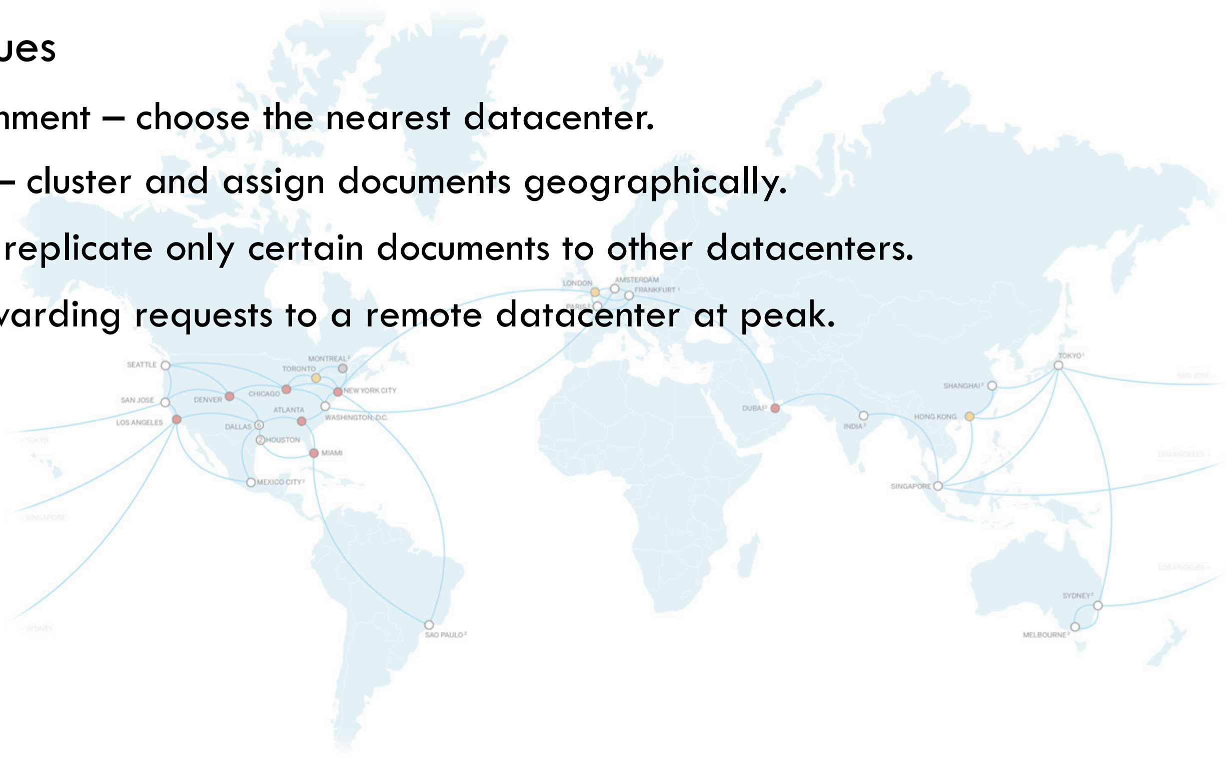
- Pipelined query processing (on top of TP)
  - Route a bundle from one node to next, process on each node and extract the results on the last one.
    - ✓ Pros: The work is dived among the nodes.
    - ✓ Pros: The network load is reduced.
    - ✓ Cons: Sequential dependency and load balancing.
  - Possible improvements
    - *Do early exit to reduce the amount of transferred data.*
    - *Apply fragment based or semi-pipelined processing to reduce query latency.*
    - *Optimize term assignment to improve network load and load balancing.*

- Other techniques
  - Hybrid partitioning – divide posting lists into chunks and distribute the chunks.
  - 2D partitioning – apply both document- and term-wise partitioning (m x n).

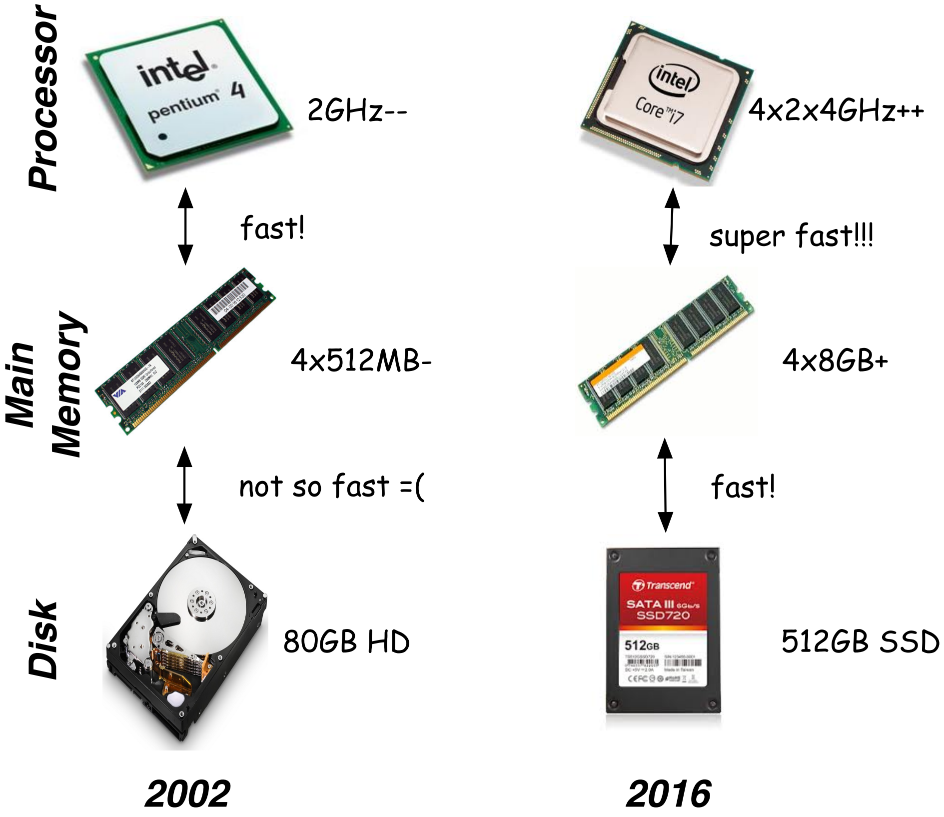
Multi-Site Query Processing

- Motivation
  - Search engines spanning across multiple datacenters open a range of new possibilities.

- Some of the techniques
  - User-to-center assignment – choose the nearest datacenter.
  - Corpus partitioning – cluster and assign documents geographically.
  - Partial replication – replicate only certain documents to other datacenters.
  - Reduce costs by forwarding requests to a remote datacenter at peak.



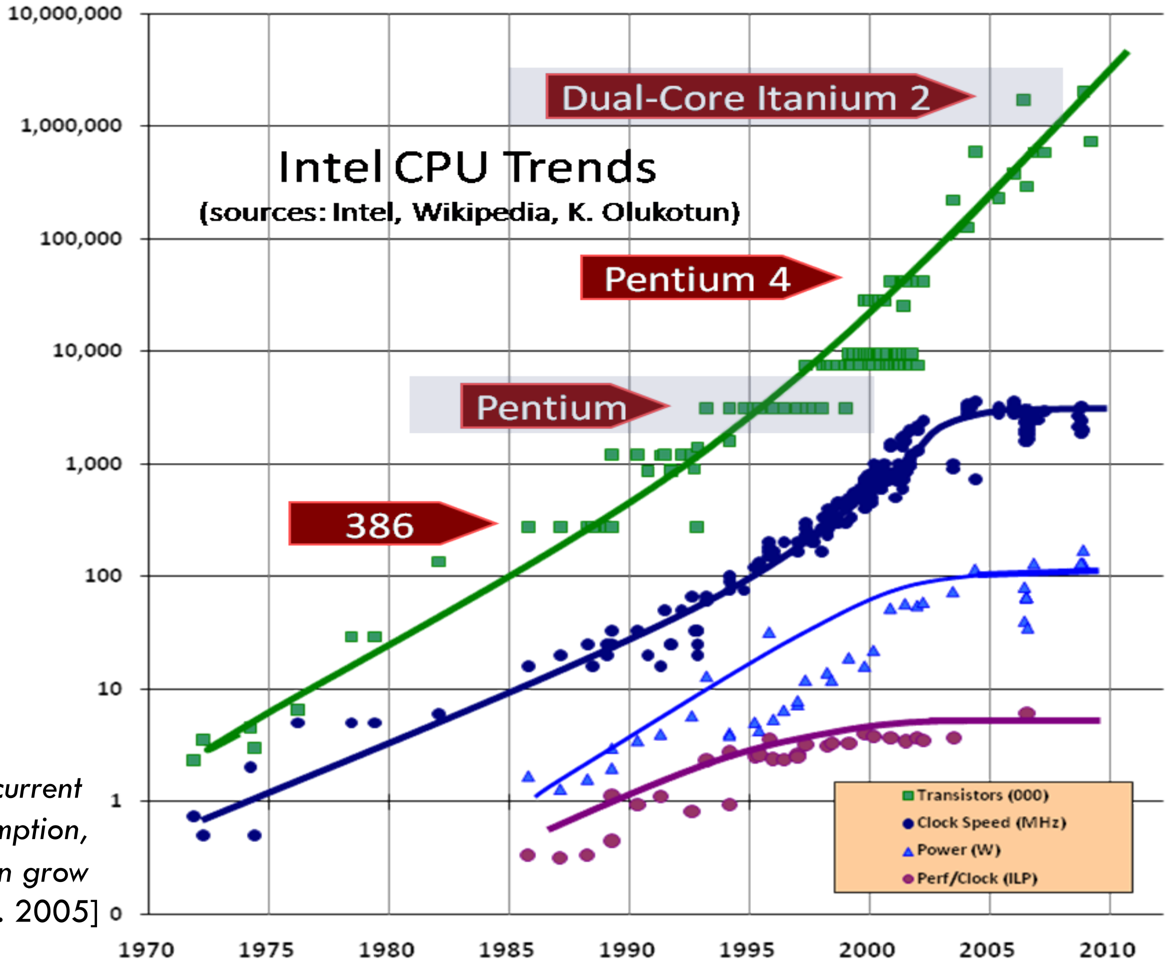
Impact of the Hardware Trends



	2002	2016
Processor	2GHz--	4x2x4GHz++
	fast!	super fast!!!
Main Memory	4x512MB-	4x8GB+
	not so fast =(	fast!
Disk	80GB HD	512GB SSD

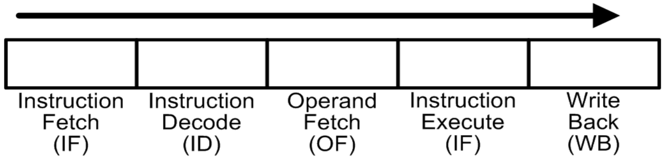
CPU: From GHz to multi-core

- Moore’s Law:
  - *~ the number of transistors on an IC doubles every two years.*
    - Less space, more complexity.
    - Shorter gates, higher clock rate.
- Strategy of the 80s and 90’s:
  - Add more complexity!
  - Increase the clock rate!
- Pollack’s Rule:
  - *The performance increase is ~ square root of the increased complexity.* [Borkar 2007]
- The Power Wall:
  - *Increasing clock rate and transistor current leakage lead to excess power consumption, while RC delays in signal transmission grow as feature sizes shrink.* [Borkar et al. 2005]

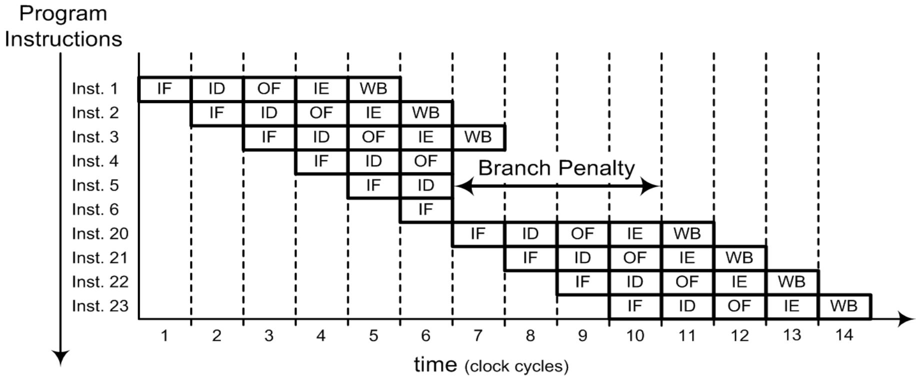


Instruction-level parallelism

- Pipeline length: 31 (P4) vs 14 stages (i7).
- Multiple execution units and out-of-order ex.:
  - **i7**: 2 load/store address, 1 store data, and 3 computational operations can be executed simultaneously.
- Dependences and hazards:
  - **Control**: branches.
    - Dean 2010: a branch misprediction costs ~5ns
  - **Data**: output dependence, antidependence (naming).
  - **Structural**: access to the same physical unit of the processor.
- Simultaneous multi-threading ("*Hyper-threading*"):
  - Duplicate certain sections of a processor (registers etc., but not execution units).
  - Reduces the impact of cache miss, branch misprediction and data dependency stalls.
  - Drawback: logical processors are most likely to be treated just like physical processors.



5 Stage Instruction Pipeline

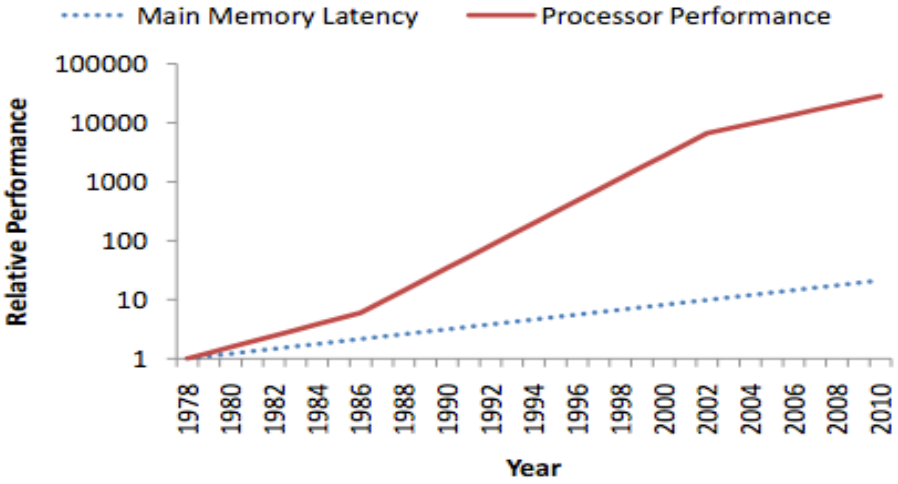


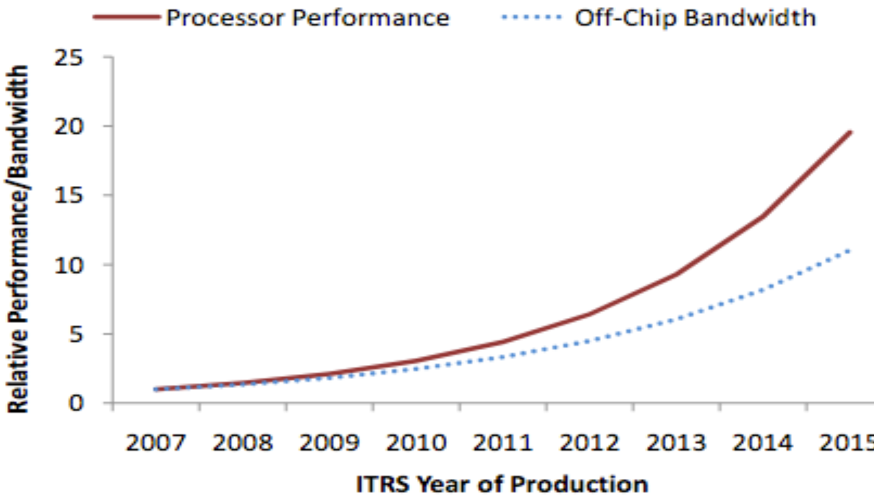
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Inst. 1	IF	ID	OF	IE	WB									
Inst. 2		IF	ID	OF	IE	WB								
Inst. 3			IF	ID	OF	IE	WB							
Inst. 4				IF	ID	OF								
Inst. 5					IF	ID								
Inst. 6						IF								
Inst. 20							IF	ID	OF	IE	WB			
Inst. 21								IF	ID	OF	IE	WB		
Inst. 22									IF	ID	OF	IE	WB	
Inst. 23										IF	ID	OF	IE	WB

Computer memory hierarchy

(Intel Core i7-2600K)

Level	Latency	Size	Technology	Managed by
Registers	<<1ns	?1KB	CMOS	Compiler
L1 Cache (on-chip)	<1ns	4x32KBx2	SRAM	Hardware
L2 Cache (off-chip)	2.5ns	4x256KB	SRAM	Hardware
L3 Cache (shared)	5ns	8MB	SRAM	Hardware
Main Memory	50ns	4x8GB+	DRAM	OS
Solid-State Drive	<100µs	512GB-	NAND Flash	Hardware/OS/User
Hard-Disk Drive	3-12ms	1TB+	Magnetic	Hardware/OS/User



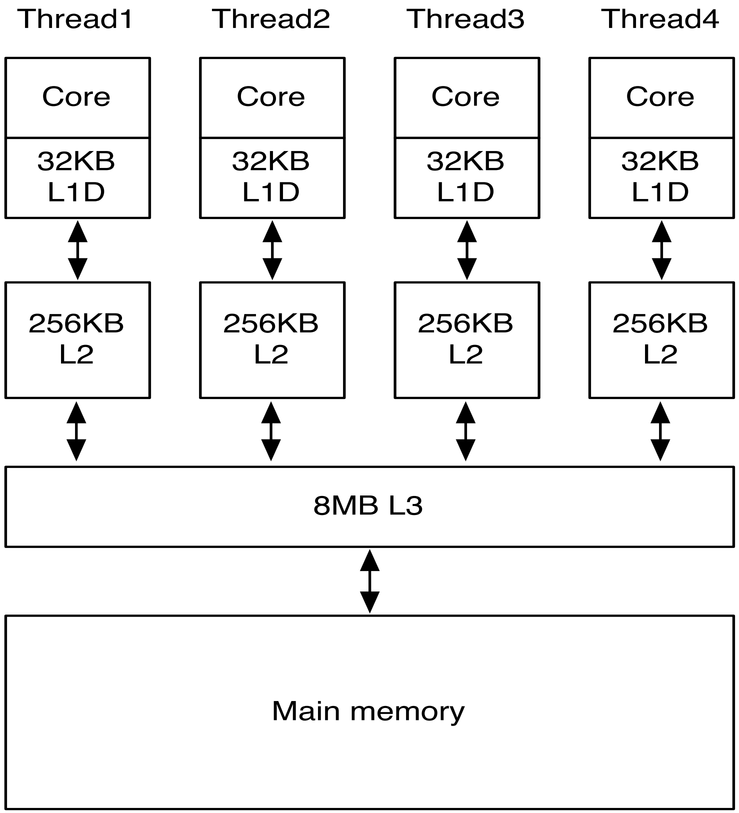


source: [Jahre 2010]

(1ms = 1 000 µs = 1 000 000 ns; 1ns = 4 clock cycles at 4GHz or 29.8cm of light travel)

Performance implications

- Some of the main challenges of CMP:
  - Cache coherence.
  - Cache conflicts.
  - Cache affinity.
- Other important cache-related issues:
  - Data size and cache line utilization.
    - i7 has 64B cache lines.
  - Data alignment and padding.
  - Cache associativity and replacement.
- Additional memory issues:
  - A large span of random memory accesses may have additional slowdown due to TLB misses.
  - Some of the virtual memory pages can also be swapped out to disk.



Cache- and processor-efficient query processing

- Modern compression methods for IR:
  - S9/S16, PFOR/NewPFD, etc.
  - Fast, superscalar and branch-free.
  - Loops/methods can be generated by a script.
- While compression works on chunks of postings, processing itself remains posting-at-a-time.
- So what about:
  - Branches and loops?
  - Cache utilization?
  - ILP utilization?
- Some interesting alternatives and trade-offs:
  - Term vs document-at-a-time processing.
  - Posting list iteration vs random access.
  - Bitmaps vs posting lists.

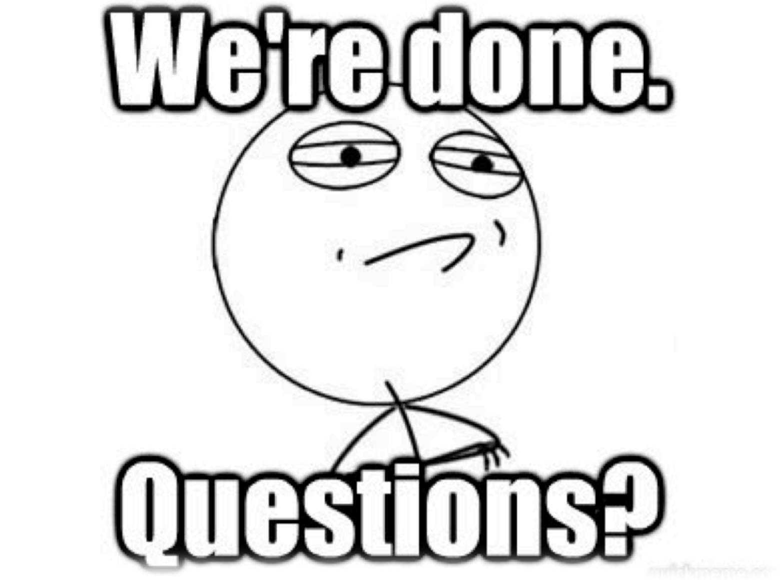
```
{
int i, w;
int outOffset = 0;
final int mask = 15;
for(i=0, w=1; i<4; ++i, w+=4){
int curInputValue0 = in[w];
int curInputValue1 = in[w+1];
int curInputValue2 = in[w+2];
int curInputValue3 = in[w+3];
out[0+outOffset] = curInputValue0 & mask;
out[1+outOffset] = (curInputValue0 >>> 4) & mask;
out[2+outOffset] = (curInputValue0 >>> 8) & mask;
out[3+outOffset] = (curInputValue0 >>> 12) & mask;
out[4+outOffset] = (curInputValue0 >>> 16) & mask;
out[5+outOffset] = (curInputValue0 >>> 20) & mask;
out[6+outOffset] = (curInputValue0 >>> 24) & mask;
out[7+outOffset] = curInputValue0 >>> 28;
out[8+outOffset] = curInputValue1 & mask;
out[9+outOffset] = (curInputValue1 >>> 4) & mask;
out[10+outOffset] = (curInputValue1 >>> 8) & mask;
out[11+outOffset] = (curInputValue1 >>> 12) & mask;
out[12+outOffset] = (curInputValue1 >>> 16) & mask;
out[13+outOffset] = (curInputValue1 >>> 20) & mask;
out[14+outOffset] = (curInputValue1 >>> 24) & mask;
out[15+outOffset] = curInputValue1 >>> 28;
out[16+outOffset] = curInputValue2 & mask;
out[17+outOffset] = (curInputValue2 >>> 4) & mask;
out[18+outOffset] = (curInputValue2 >>> 8) & mask;
out[19+outOffset] = (curInputValue2 >>> 12) & mask;
out[20+outOffset] = (curInputValue2 >>> 16) & mask;
out[21+outOffset] = (curInputValue2 >>> 20) & mask;
out[22+outOffset] = (curInputValue2 >>> 24) & mask;
out[23+outOffset] = curInputValue2 >>> 28;
out[24+outOffset] = curInputValue3 & mask;
out[25+outOffset] = (curInputV
out[26+outOff
```

code: https://github.com/javasoze/kamikaze

Acknowledgements

- For details and references see Chapter 2 of my PhD thesis:
  - *“Efficient Query Processing in Distributed Search Engines” – http://goo.gl/vDNGGb*
- Also a great tutorial by Barla Cambazoglu and Ricardo Baeza-Yates:
  - *“Scalability And Efficiency Challenges In Large-Scale Web Search Engines”* – http://goo.gl/oyWDqU
- A very good book:
  - *“Information Retrieval: Implementing and Evaluating Search Engines”* – Büttcher et al., The MIT Press, 2010
- Any publication co-authored or even cited by these guys:
  - Ricardo Baeza-Yates, B. Barla Cambazoglu, Mauricio Marin, Alistair Moffat, Fabrizio Silvestri, Torsten Suel, Justin Zobel (in alphabetic order).

Thank you!



simon.jonassen@gmail.com