From Several Hundred Million to Some Billion Words: Scaling up a Corpus Indexer and a Search Engine with MapReduce

Jordi Porta
Computational Linguistics Area
Departamento de Tecnología y Sistemas
Centro de Estudios de la Real Academia Española
(porta@rae.es)

Outline

“The more observations we gather about language use, the more accurate description we have about language itself”

Lin & Dyer, 2010: *Data-intensive text processing with MapReduce*

- A Case Study: *Elpais.com*
- Basic Data Structures for Complex Queries
- Introduction to MapReduce
- The Scale out vs. Scale up Dilemma
- Scaling up with Phoenix++
- Experiments introducing MapReduce into a CMS:
  - Query Engine:
    - Word Counting
    - Word Cooccurrencies
    - N-grams
  - Indexer
    - File Inversion
      - Text Inversion
      - Structure Inversion

“The only thing better than big data is bigger data”

A Case Study: *Elpais.com* on-line archive

Base Corpus:

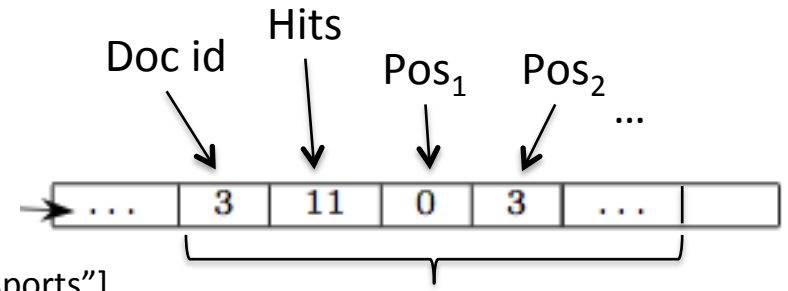
- *Elpais*.com on-line archive:
- 1976–2011
- 2.3 million news
- Cleaned + PoS tagged
- 23 million structural elements
- 878 million tokens

↓

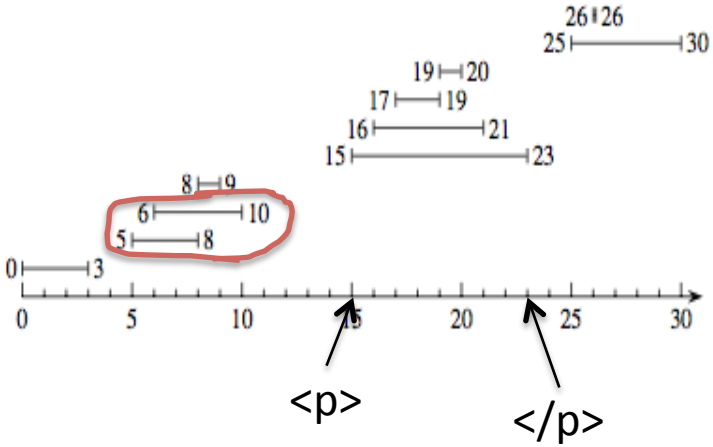
- Lists of linguistic elements:
  - Concordances
  - Counts
  - Associations
  - Distributions
  - …
- Sub-corpora
- Non-trivial processing:
  - Cooccurrence networks / clustering / …
  - Word vectors
  - …

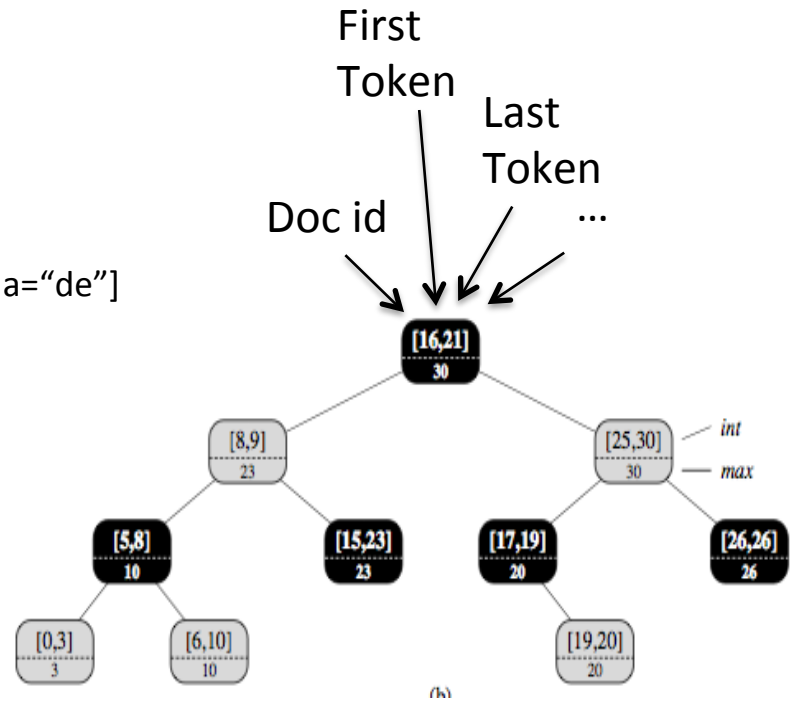
Basic Data Structures for Corpus Indexing Supporting Complex Queries

- Textual indices:
  - Posting Lists:
    - [token="de"]
    - [token="la" and msd="Nfs"]
    - [token="pelota"] within [section="sports"]
    - …



- Structural indices:
  - Interval Trees:
    - P within TEXT
    - P containing HI[REND=IT] containing [lemma="de"]
    - …

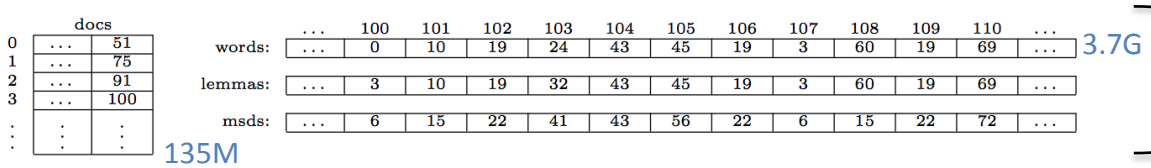




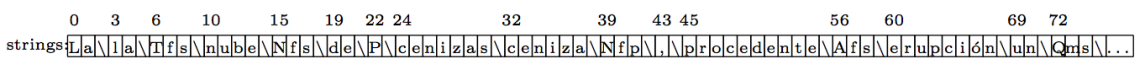
How Text is Indexed

-	-	0	1	2	3	4	5	6	7	8	9	10	...
<text>	<p>	La	nube	de	cenizas	,	procedente	de	la	erupción	de	un	...
<text>	<p>	la	nube	de	ceniza	,	procedente	de	la	erupción	de	un	...
<text>	<p>	Tfs	Nfs	P	Nfp	,	Afs	P	Tfs	Nfs	P	Qms	...

Documents:
- begin,
- end,
- offset
- …



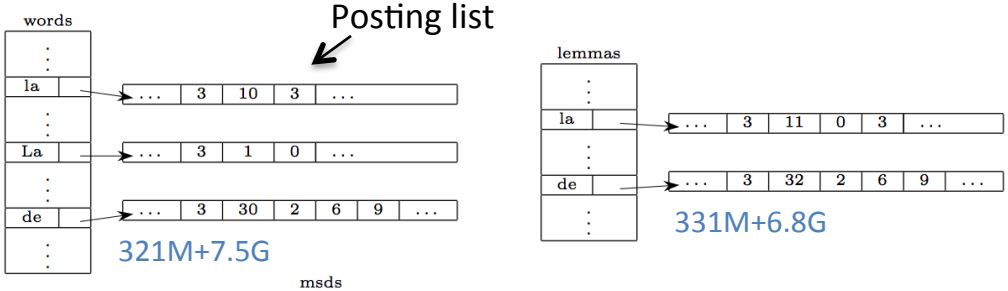
String Singletons



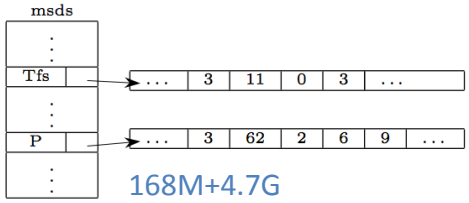
Corpus Layers

Positional Indices

Posting list

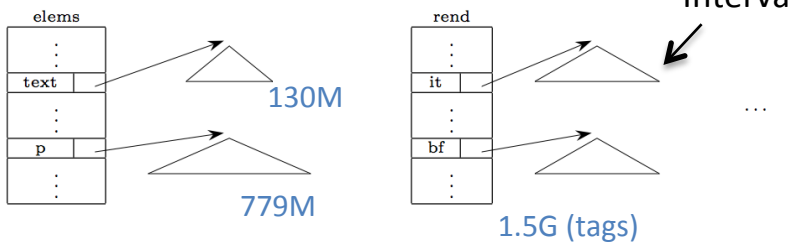


Hash Table



Structural Indices

Interval tree



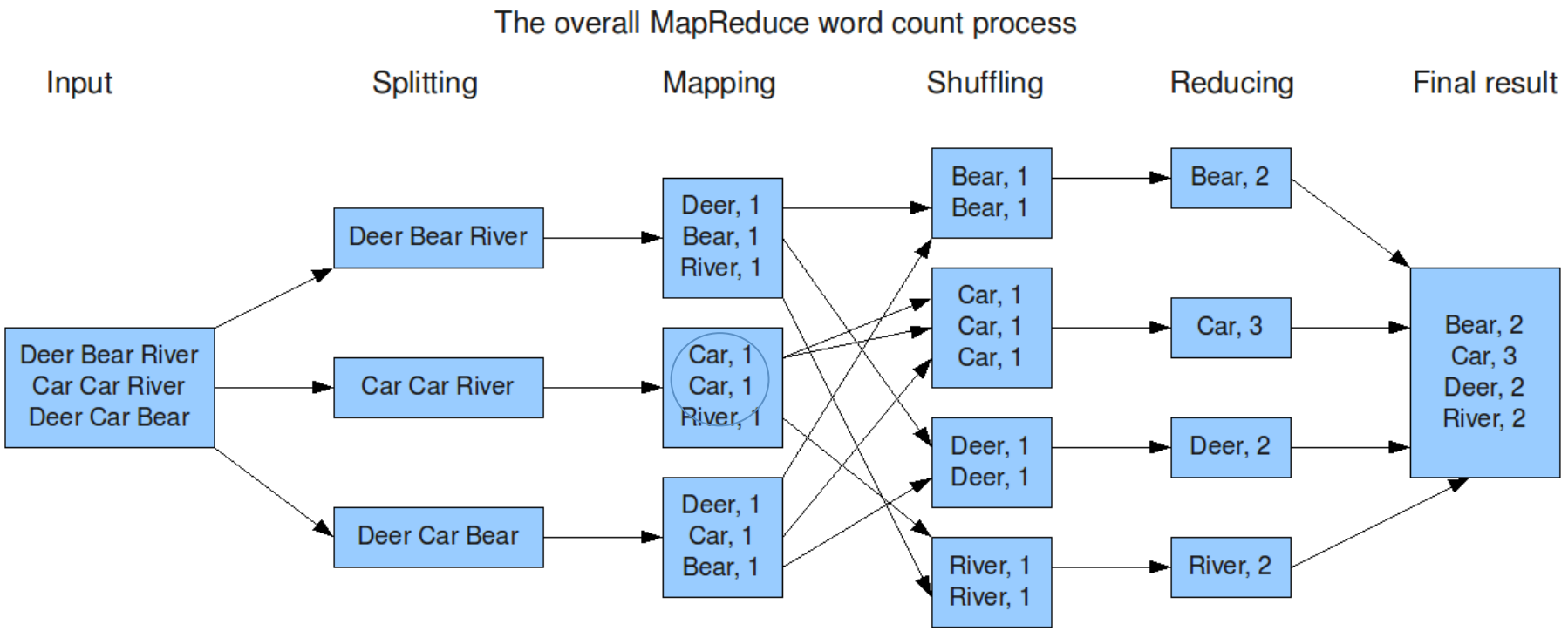
What is MapReduce?

- A widely used parallel computing model
- For batch processing of data on terabyte or petabyte scales
- On clusters of commodity servers (typically)
- Fault tolerance: task monitoring and replication
- Abstracts parallelization, synchronization and communication
- Hadoop is the most popular MapReduce scale out implementation

Hadoop MapReduce

Map(k1, v1) -> list(k2, v2)

Reduce(k2, list(v2)) -> list(v2)



Scaling out vs Scaling up Dilemma

- Scaling out (horiz.)
  - Cluster of commodity servers
  - Pros:
    - Cheaper
    - Fault tolerance
    - Easy to update
  - Cons:
    - Power consumption
    - Cooling
    - Space
    - Difficult to implement
    - Networking equipment
    - Licensing costs
- Scaling up (vert.)
  - Powerful server (+proc./+RAM)
  - Pros:
    - Power consumption
    - Cooling
    - Space
    - Easy to implement
    - Networking equip. ~ 0
    - Licensing costs
  - Cons:
    - Expensive
    - Fault tolerance
    - Difficult to update

Scaling out vs Scaling up Dilemma

- Appuswamy & al. (2014) *"Scale-up vs Scale-out for **Hadoop**: Time to rethink?"* In *Proc. 4th Ann. Symp. on Cloud Computing*.

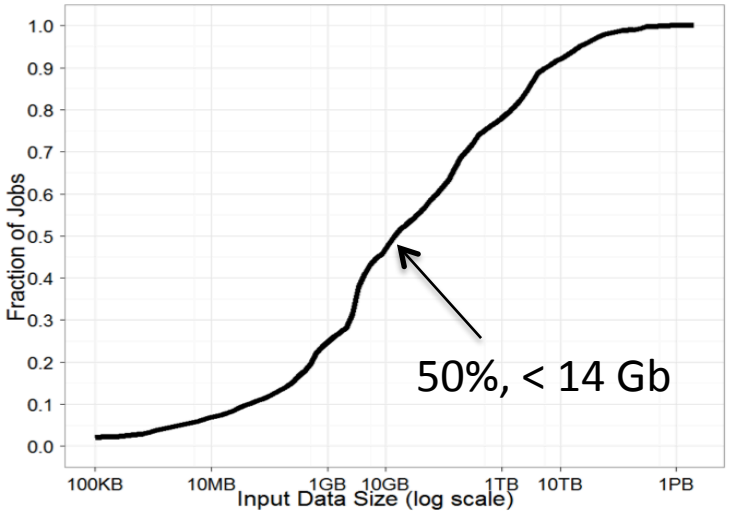


Figure 1: Distribution of input job sizes for a large analytics cluster

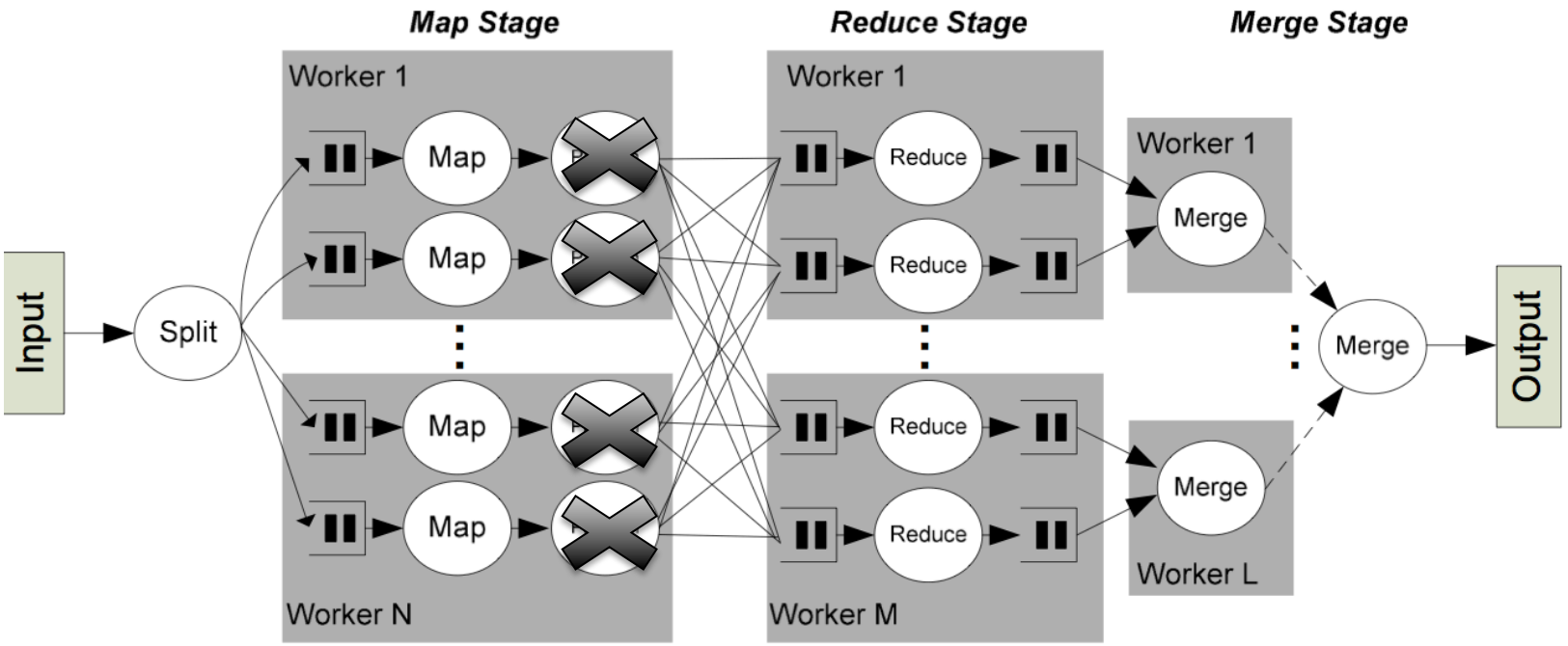
	Workstation[1]	Server[1]
Base spec	E5520	4x E7-4820
- CPU	4 cores, HT, 2.3 GHz	32 cores, HT, 2.0 GHz
- Memory	12 GB	0 GB
- Disk	1 HDD (160 GB)	2 HDD (600 GB)
Base cost	$2130	$17380
SSD cost	$270[2]	$2160[2]
DRAM cost	none	$5440[2,3] (512GB)
Total cost	$2400	$24980
Power	154 W	877 W[4]

[1] Prices from Dell's website, http://www.dell.com/
[2] Prices from http://www.newegg.com
[3] 16 GB 240-Pin DDR3 1333 SDRAM.
[4] Measured with 512GB of RAM and all (HDD and SSD) drives connected.
* All processors are Intel Xeon

Table 3: Comparison of workstation (used for scale-out) and server (scale-up) configurations. All prices converted to US$ as of 8 August 2012.

- Input size in analytics production clusters:
  - Microsoft / Yahoo: median < 14 Gb, Facebook: 90% < 100 Gb
- Conclusion:
  - "Scaling up is **more cost-effective** for **many real-word problems**"

Scaling up with Phoenix++ MapReduce

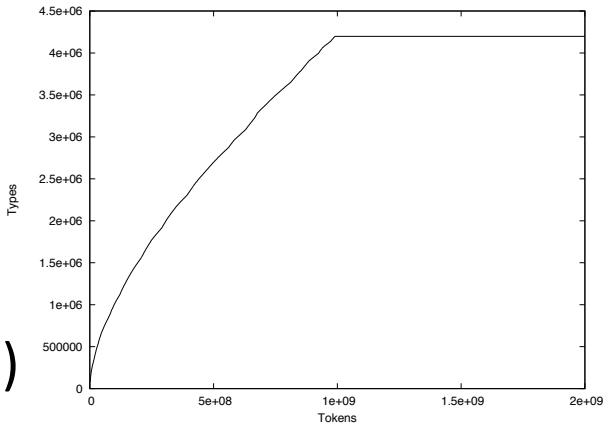


"Phoenix++: Modular MapReduce for Shared-Memory Systems", 2011: Justin Talbot, Richard M. Yoo, and Christos Kozyrakis. *Second International Workshop on MapReduce and its Applications.*
(Code: http://mapreduce.stanford.edu)

- Split: divides input data to chunks for map tasks
- Combiner: are thread-local objects
- Map: invokes the Combiner for each key-value pair emitted (not at the end of Map)
- ~~Partition/Suffle~~: not in Phoenix++
- Reduce: aggregates key-value pairs in Combiners (All Map task are finished)
- Merge: sort key-value pairs (optional phase)
- Problems: introduces rehash between map and reduce phases

Experiments: Data & Machinery

- Base Corpus
  - *Elpais*.com on-line archive (1976–2011)
  - 2.3 million news
  - Cleaned + PoS tagged
  - 23 million structural elements
  - 878 million tokens
- Multiples of *Elpais*.com
  - x2, x4, x8, x16
  - Don't obey Heaps' Law ($V(n) = K \cdot n^{\beta}$)

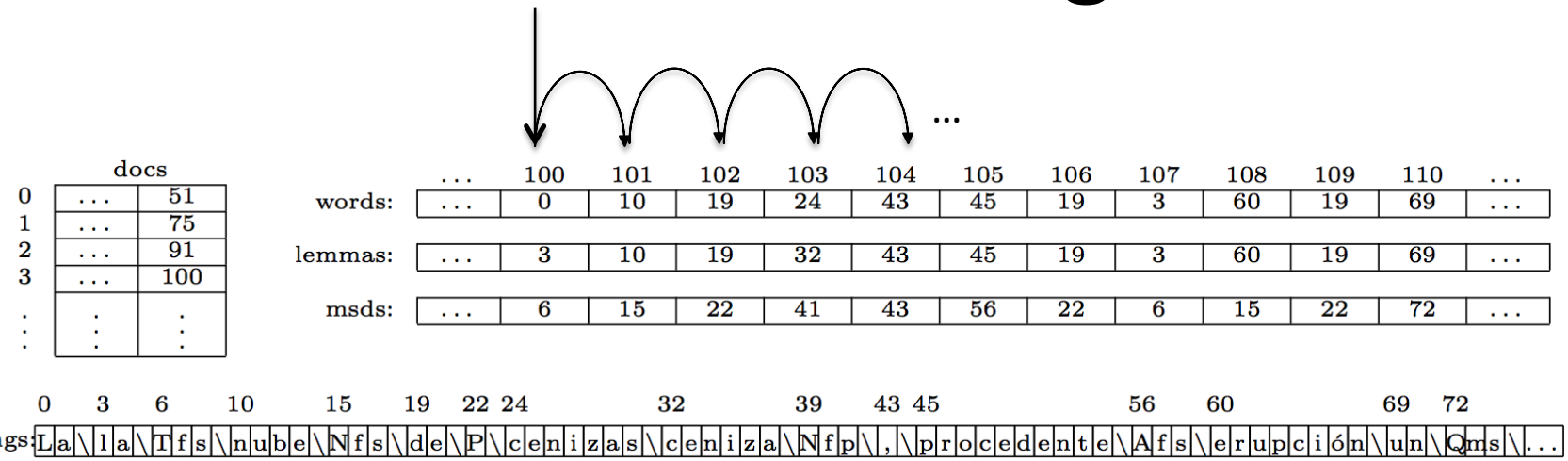


- 1x Dell PowerEdge R615
  - RAM: 256 Gb
  - CPUs: 2 x Xeon E5-2620 / 2 GHz (24 threads)
  - DISKS: 3 x SAS 7.200 rpm, RAID-5

Word Counting

Given a (sub-)corpus,
compute the frequencies
of all the words
(word frequency profile)

Word Counting



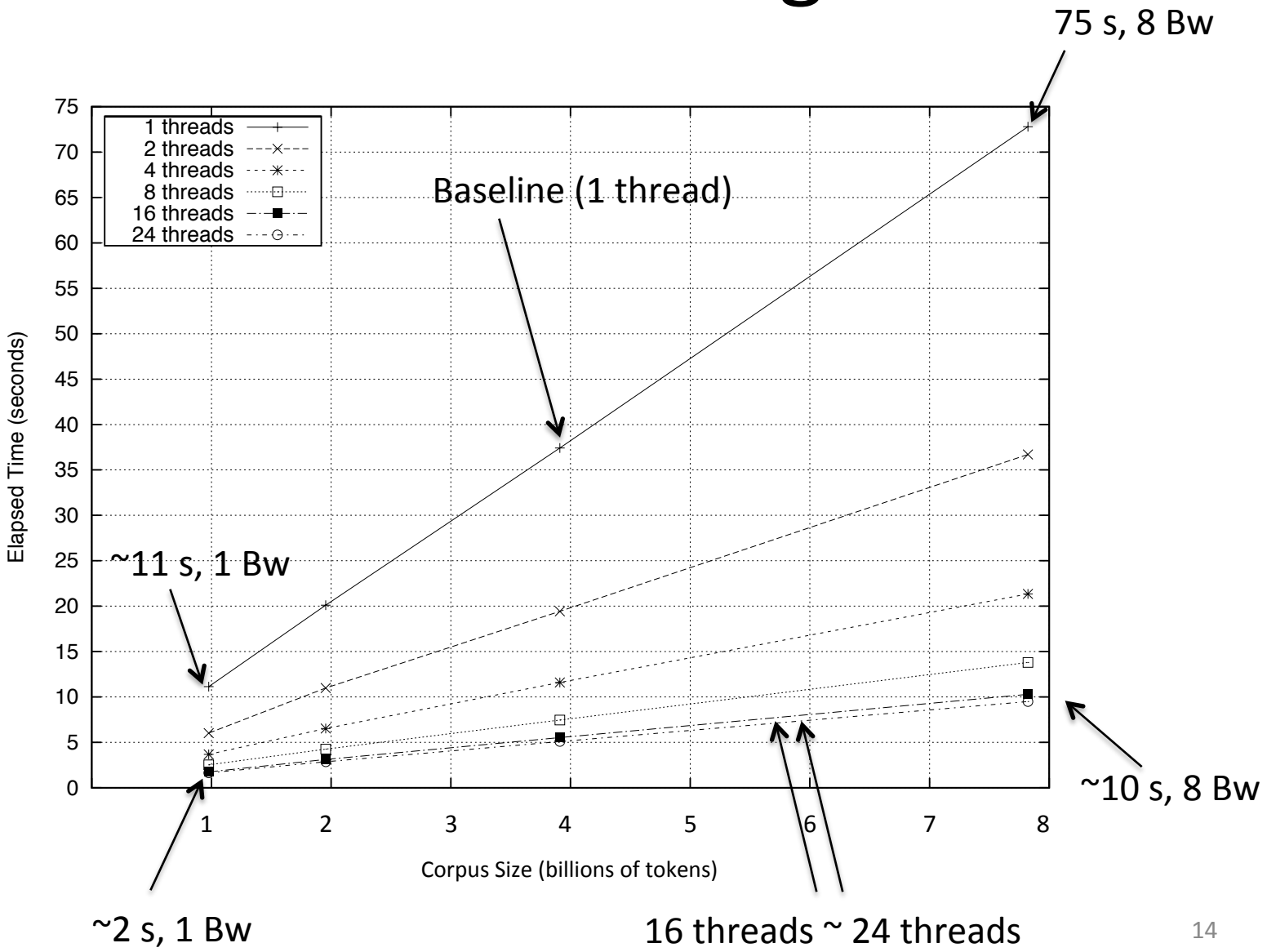
Split(docs) =
  divide docs into <doc, begin, end>

Map(<doc, begin, end>) =
  for i = begin to end do
    emit(words[docs[doc].offset + i], 1)
  end

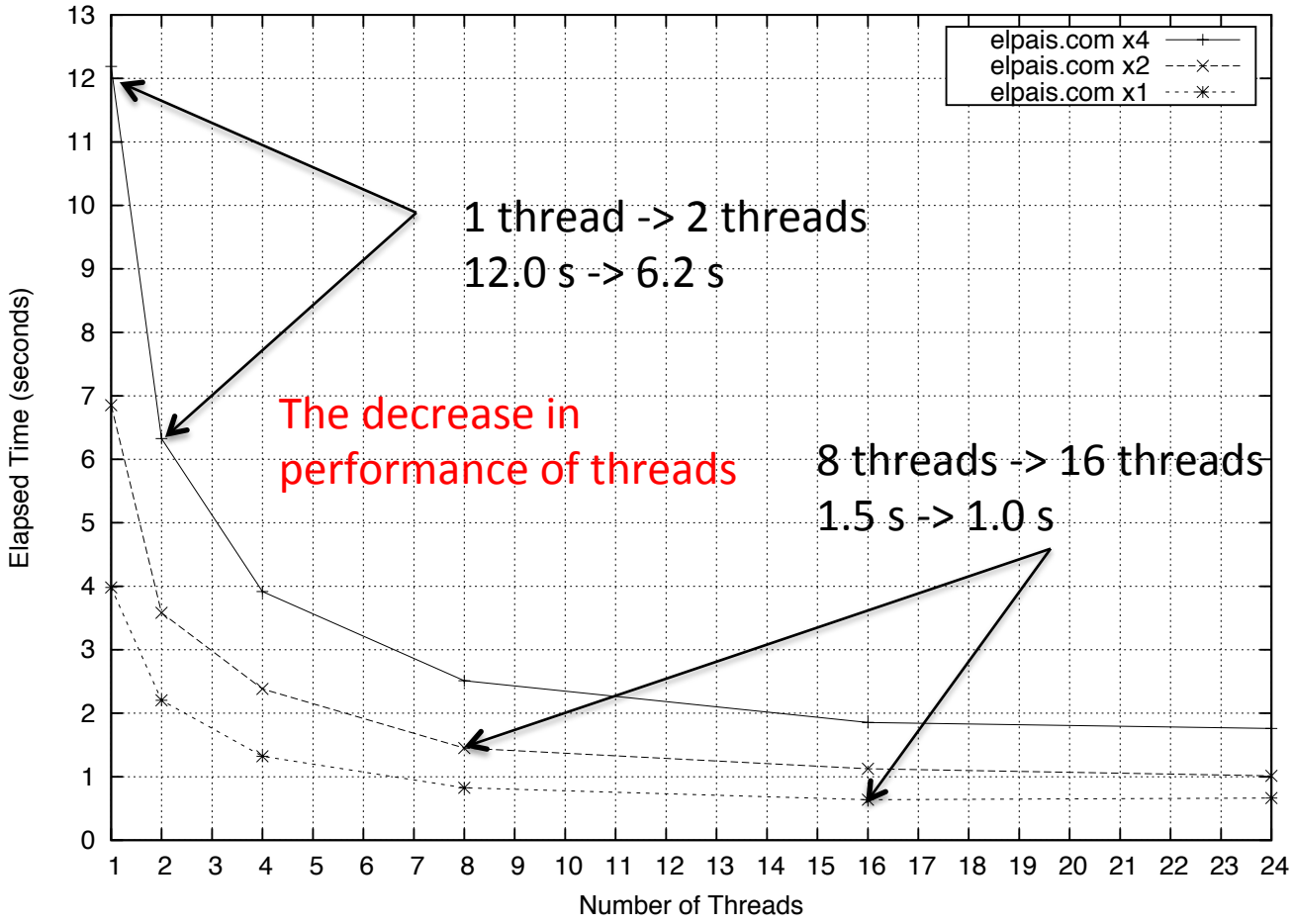
Combiner:

Word	Freq.
0 (la)	456
19 (de)	5667
…	…

Word Counting



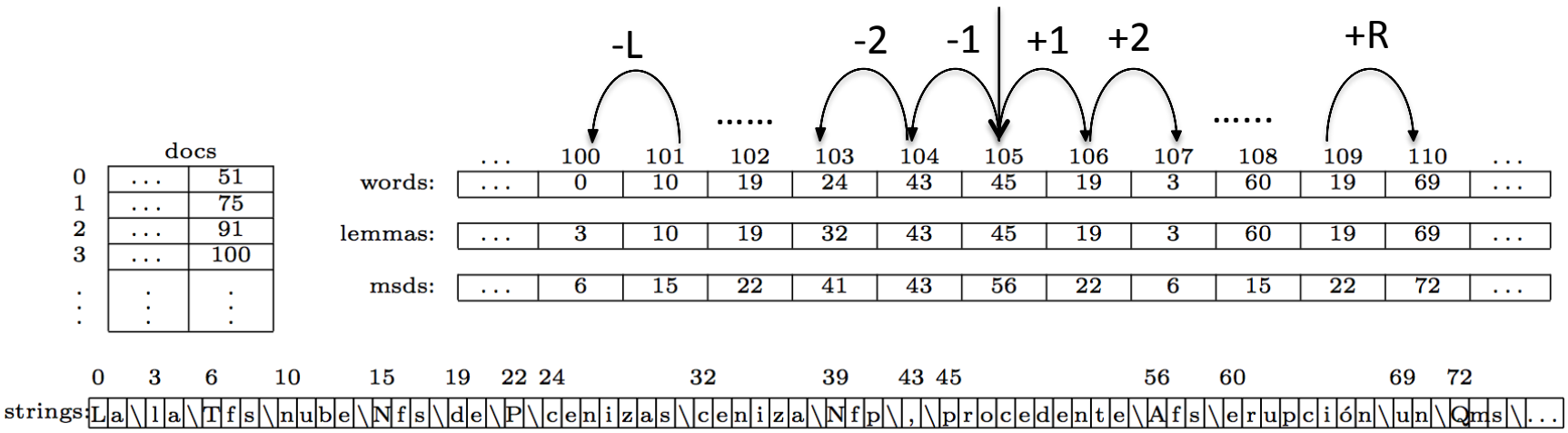
Word Counting



Word Cooccurrencies

Given a word (or a lema or a PoS, …),
compute the frequency of the
surrounding words (left/right)

Word Cooccurrences



Split(Posting list) =
  divide the posting list into <doc, n, positions>

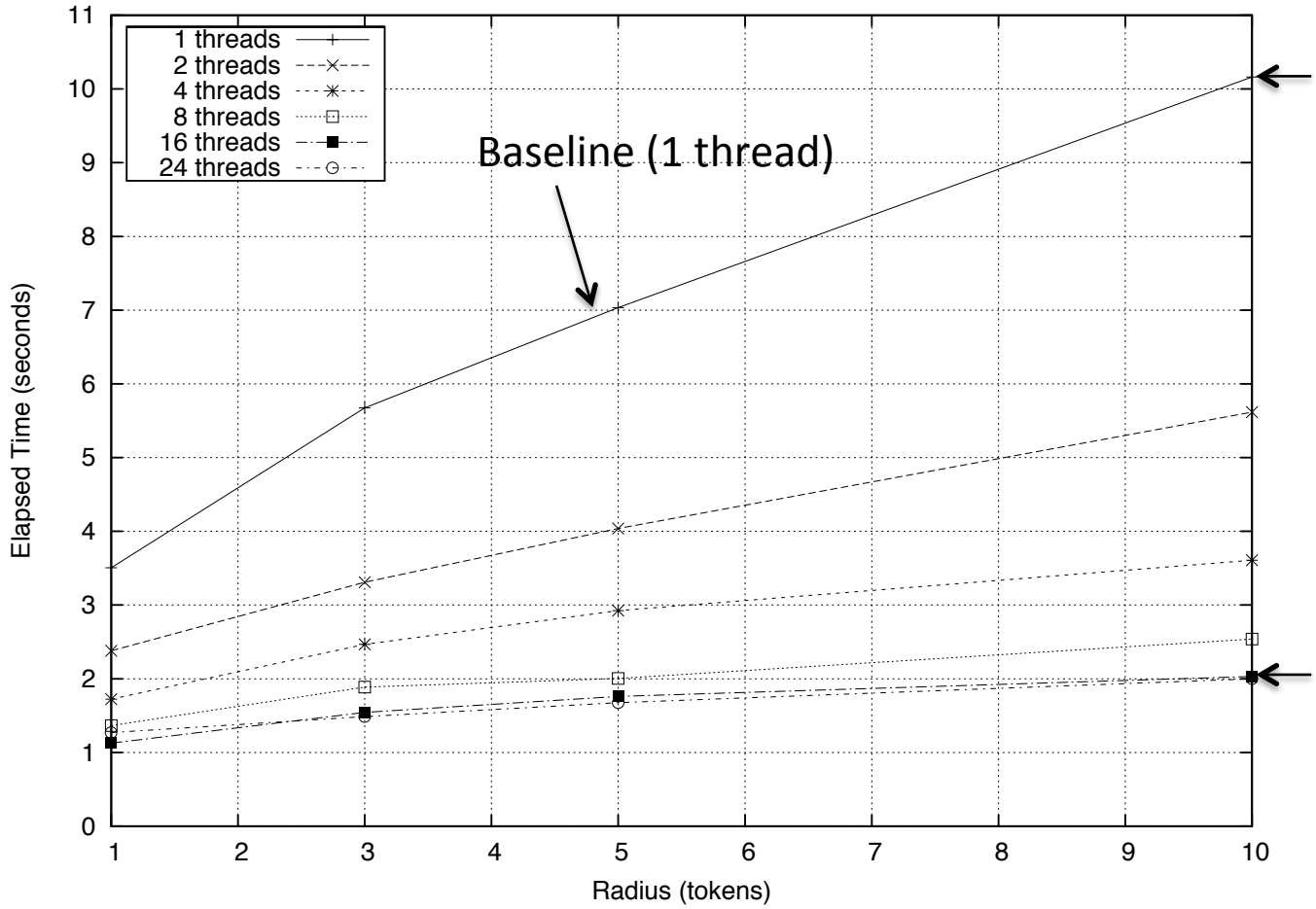
Combiner:

Word	Freq.
0 (la)	456
19 (de)	5667
…	…

```
Map(<doc, n, pos>) =
  for i = 0 to n - 1 do
    for j = 1 to L do                          ┐
      emit(words[docs[doc].offset + pos[i] - j], 1)  ├ Left Context
    end                                        ┘
    for j = 1 to R do                          ┐
      emit(words[docs[doc].offset + pos[i] + j], 1)  ├ Right Context
    end                                        ┘
  end
```

Word Cooccurrences

of [token = “de”] in Elpais.com x1 => 55.7 Millions



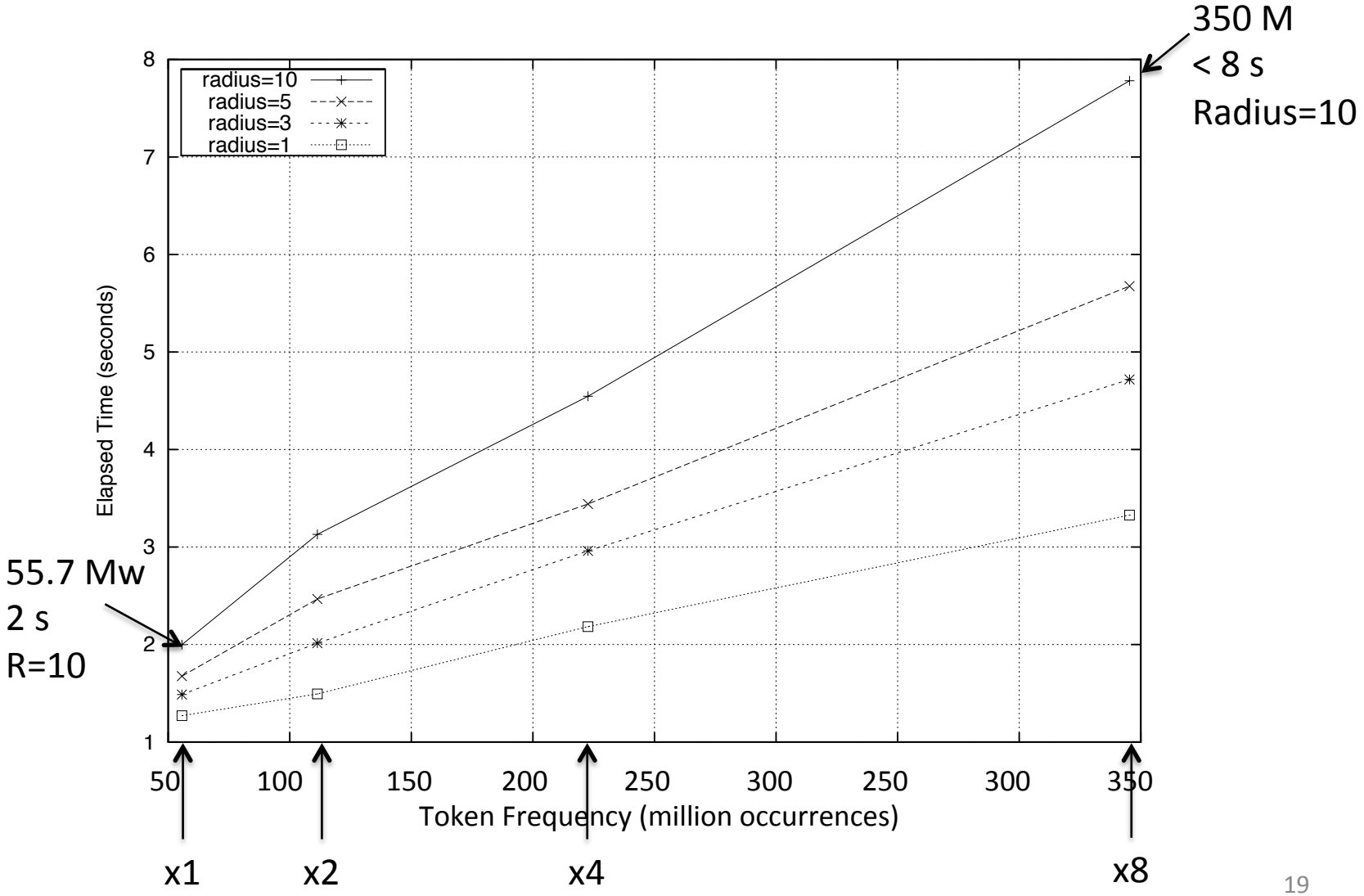
10s
Radius=10
1 Thread

2s
Radius<=10
24 Threads

L = R = Radius

Word Cooccurrences

of [token = “de”] in Elpais.com xN using 24 threads



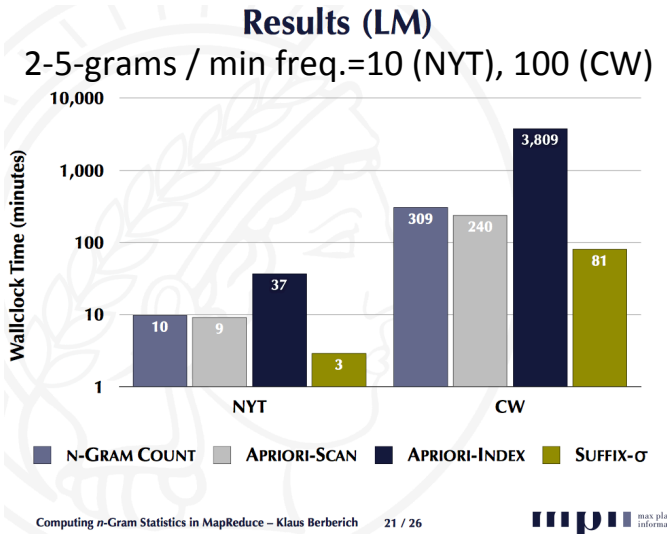
N-grams

Given a (sub-)corpus,

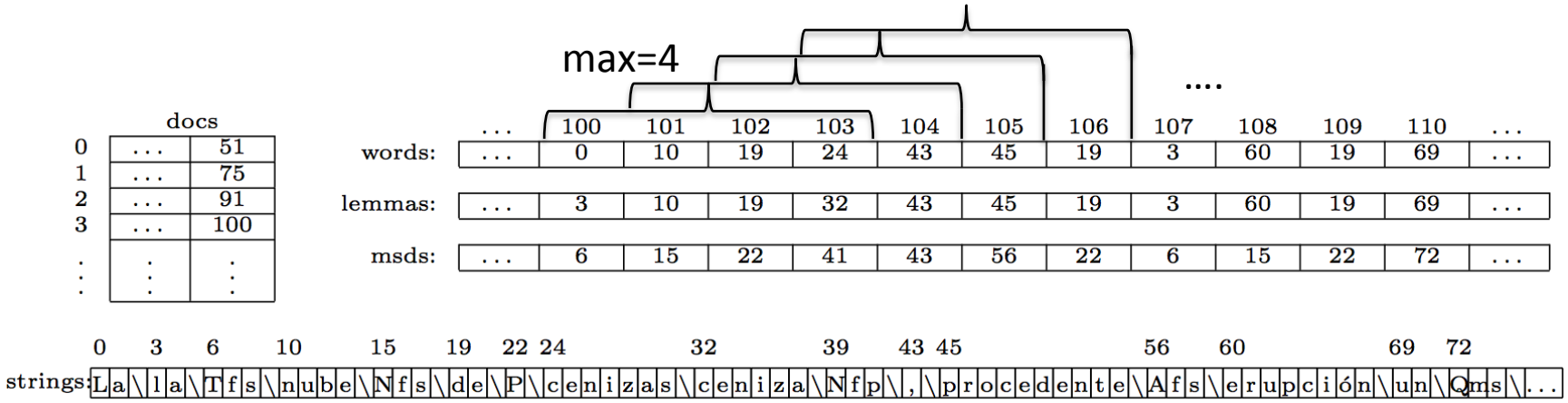
compute the frequency of all the sequences of words (lemma, PoS, …) with lengths between a minimum and a maximum

MapReduce N-grams
The Suffix-σ Algorithm

- Klaus Berberich and Srikanta Bedathur (2013): "Computing n-Gram Statistics in MapReduce". In *Proceedings of 16th International Conference on Extending Database Technology* (EDBT 2013)
  - Stage 1: Maximal n-gram counts are computed and sorted
  - Stage 2: Shorter n-gram counts are computed from the suffix array of maximal n-grams in the reducer tasks
- New York Times Annotated Corpus (NYT)
  - 1.8 million newspaper articles, 1987-2007, ~3 Gb
  - 1 billion tokens (345,827 types)
- ClueWeb09-B (CW)
  - 50 million web documents, 2009, ~246 Gb
  - ~21 billion tokens (979,935 types)
- Hadoop Cluster: 10 nodes
  - 2x6 cores, 64 Gb RAM, 4x2 Tb HDD



MapReduce N-grams



Split(Interval tree) =
  divide the interval tree into intervals:
    <doc, begin, end>

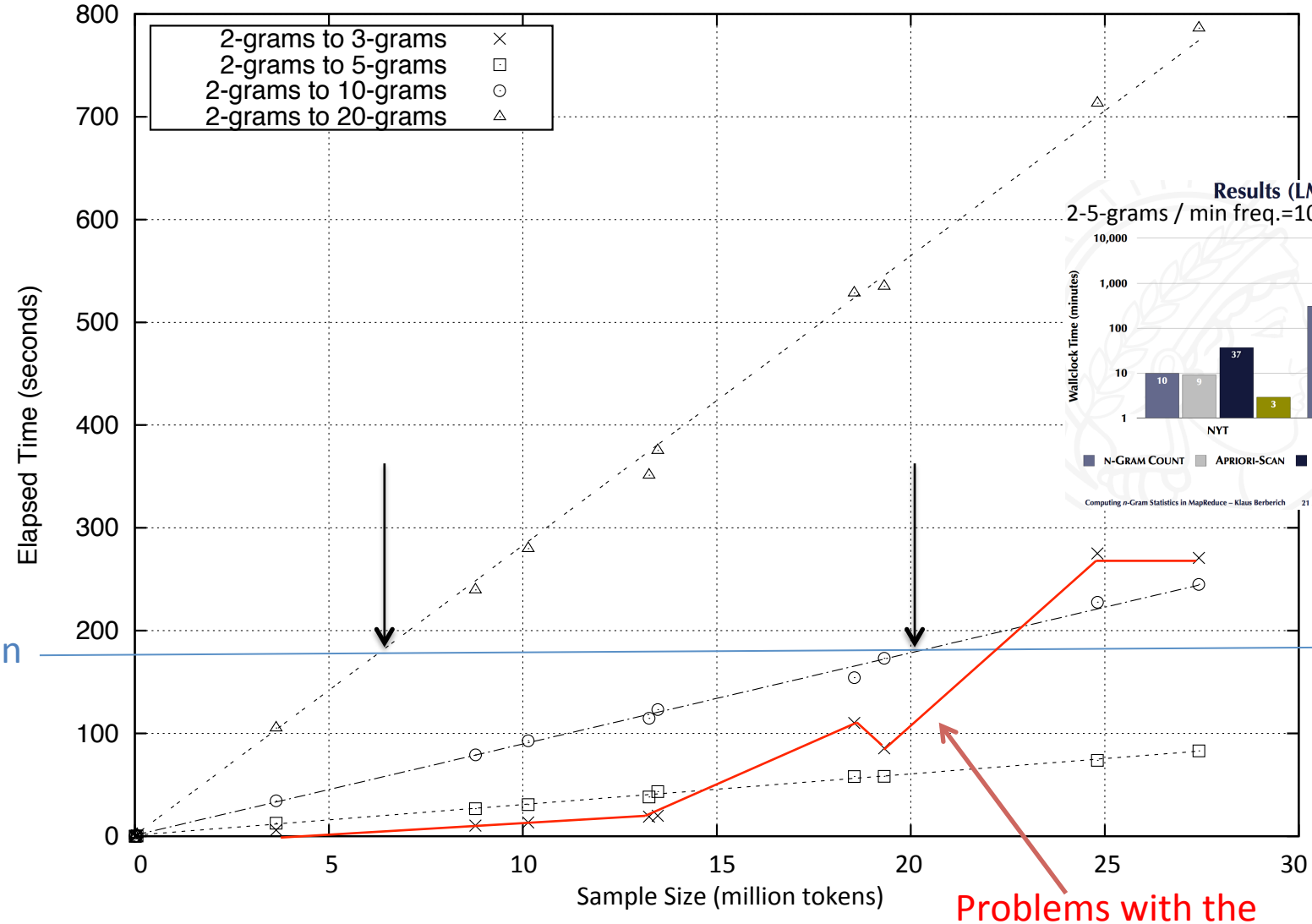
Combiner:

Word	Freq.
<0,10,19,24> (la,nube,de,cenizas)	456
<19,24,43,45> (de,cenizas,",",procedente)	5667
…	…

```
Map(<doc, begin, end>) =
  for pos = begin to end do
    emit(<words[docs[doc].offset + pos,
            words[docs[doc].offset + pos + 1,
            …
            words[docs[doc].offset + pos + max>,
          1)
  end
```

Maximal n-gram

MapReduce N-grams



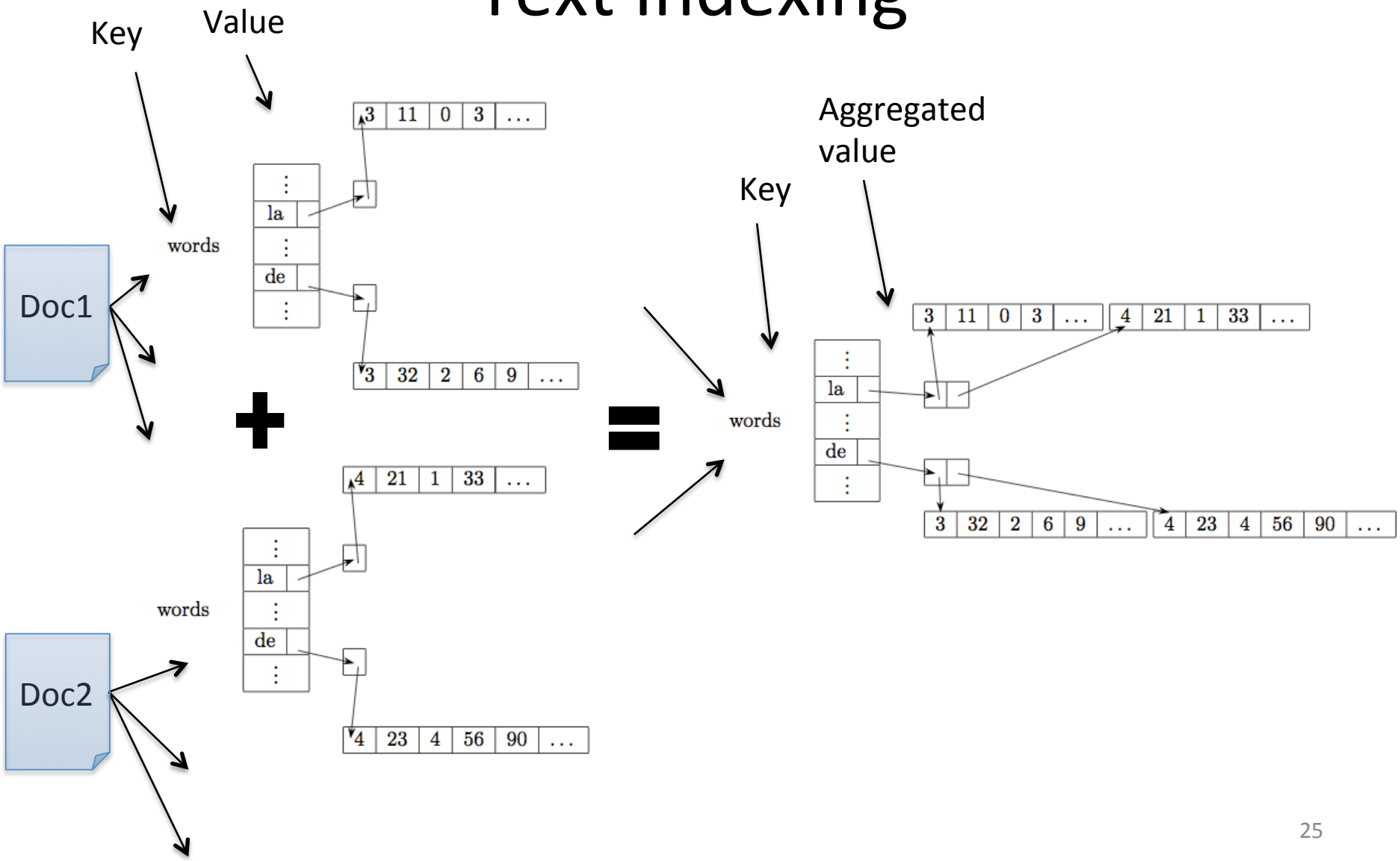
3 min

Problems with the hash table combiner

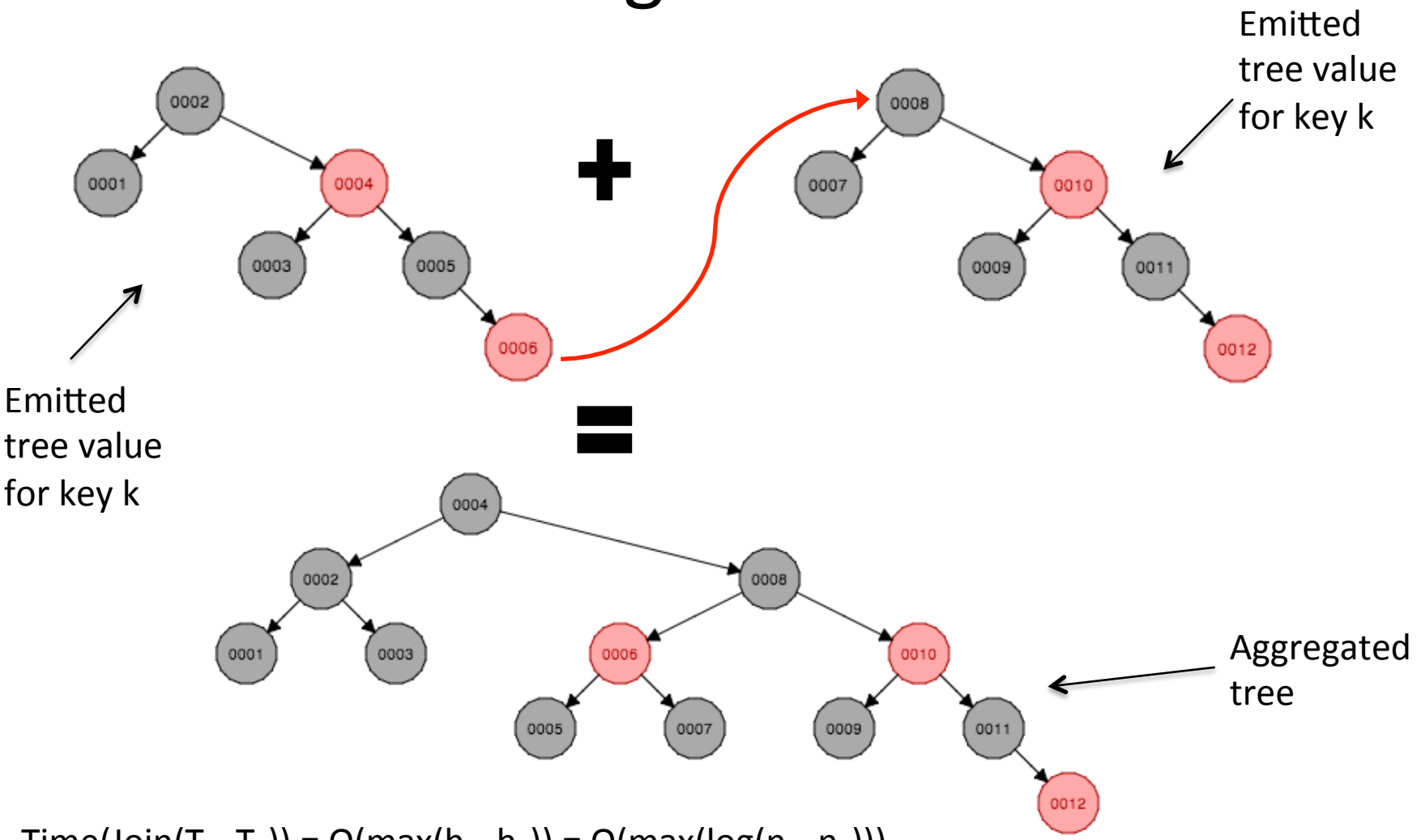
MapReduce into the Indexer: File Inversion

- Text inversion
- Structural inversion

MapReduce File Inversion
Text Indexing

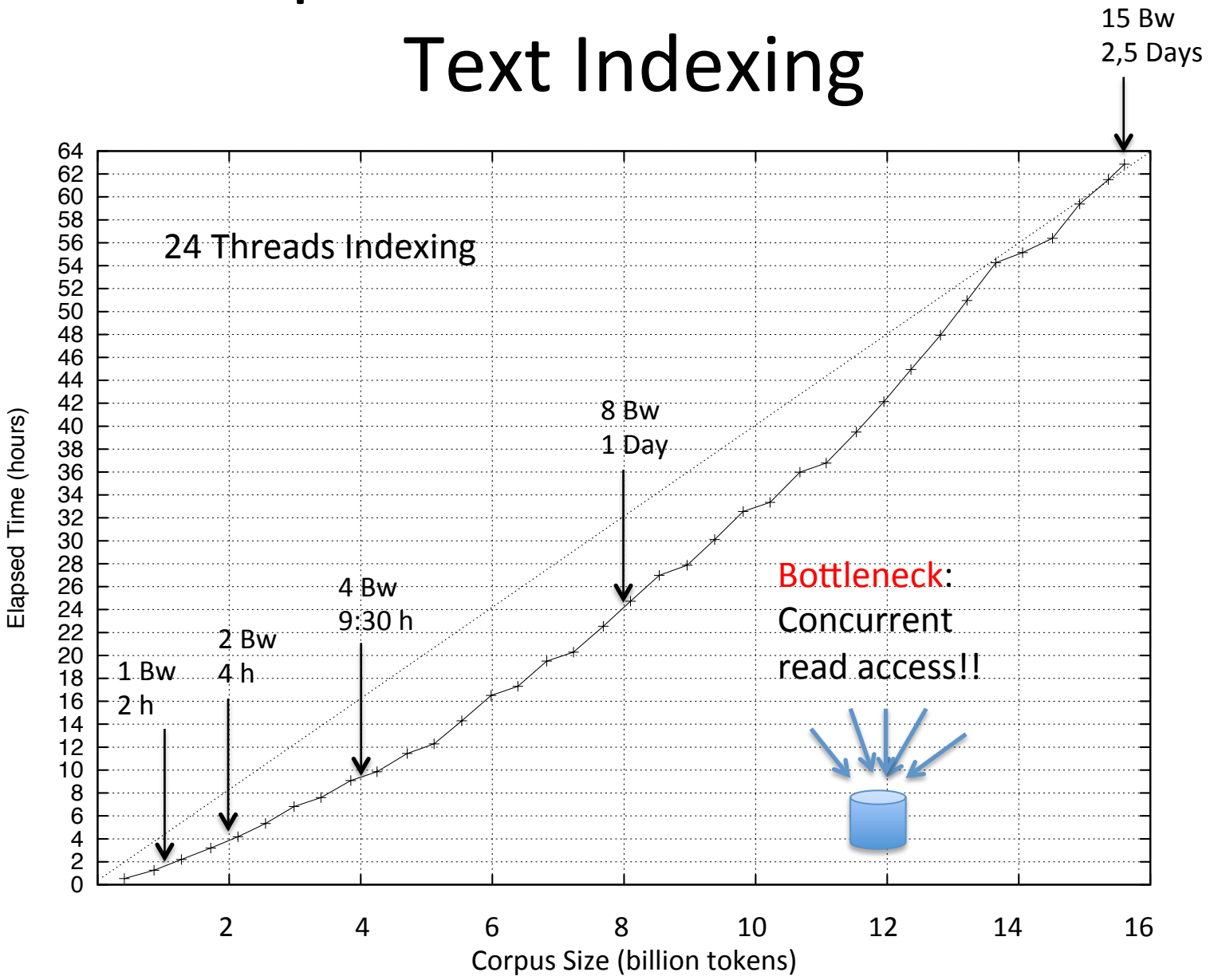


MapReduce File Inversion: Indexing Structure



Time(Join(T_1, T_2)) = O(max(h_1, h_2)) = O(max(log(n_1, n_2)))

MapReduce File Inversion: Text Indexing



Conclusions & Future Work

- Conclusions:
  - Applications scale well to the sizes of interest using MapReduce
  - MapReduce in multicore servers is a cost-effective solution for corpus management tools
- Future Work:
  - Improve hash containers
  - Introduce new associative containers
  - Implement other non-trivial statistics

Thank you !!

Questions ??