Inverted Indexing, MT, MapReduce

4/4/2016 CSE 5/624 PSLC

Game plan for today:

Quick overview of inverted indexing

Inverted indexing & Map-Reduce

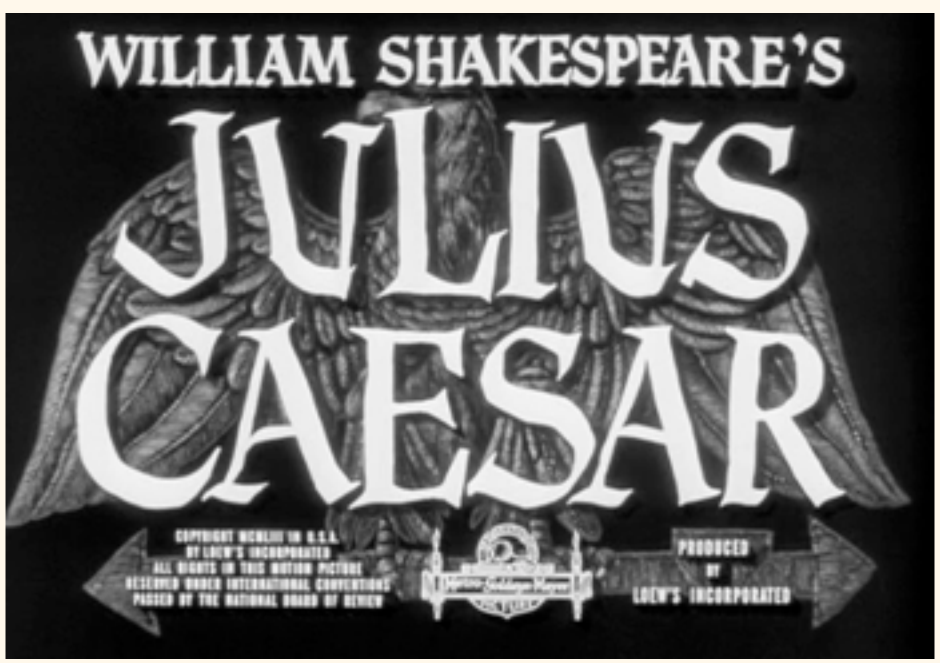
Quick MT overview

Map-Reduce & MT Model Estimation

Information Retrieval 101:

We have a set of documents...

... and we want to be able to search them by the terms that they contain.



	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

To answer the query Brutus AND Caesar AND NOT Calpurnia, we take the vectors for Brutus, Caesar and Calpurnia, complement the last, and then do a bitwise AND:

110100 AND 110111 AND 101111 = 100100

Example from Manning et al.'s "Introduction to Information Retrieval"

Information Retrieval 101:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	…
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
…							

Obviously, this approach can only scale so far...

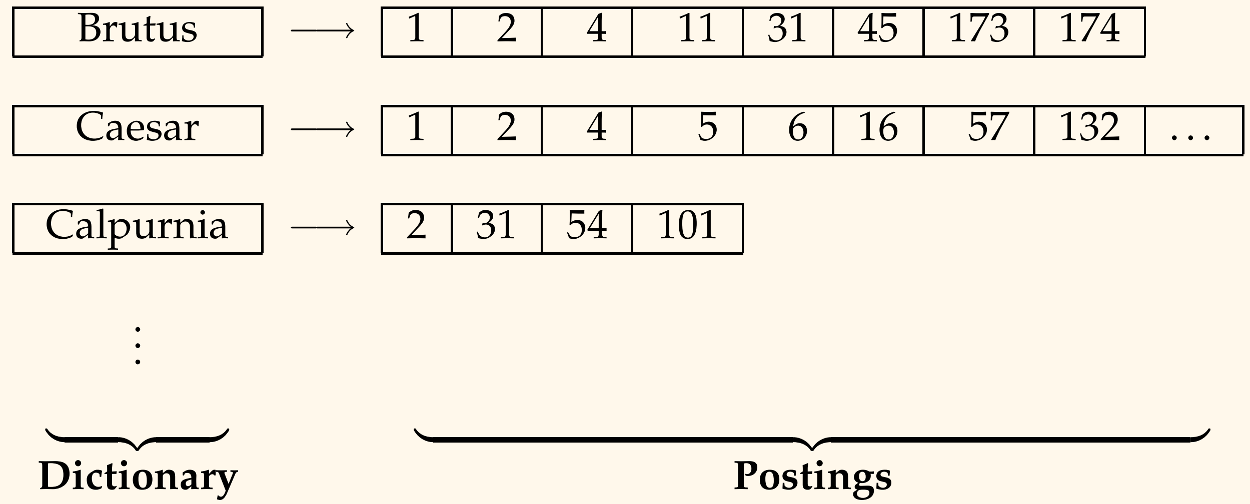
(as it happens, "so far" is actually not very far in this case)

One solution: build an index mapping terms to the documents in which they may be found.

Brutus	⟶	1	2	4	11	31	45	173	174	
Caesar	⟶	1	2	4	5	6	16	57	132	…
Calpurnia	⟶	2	31	54	101					
⋮										

Dictionary **Postings**

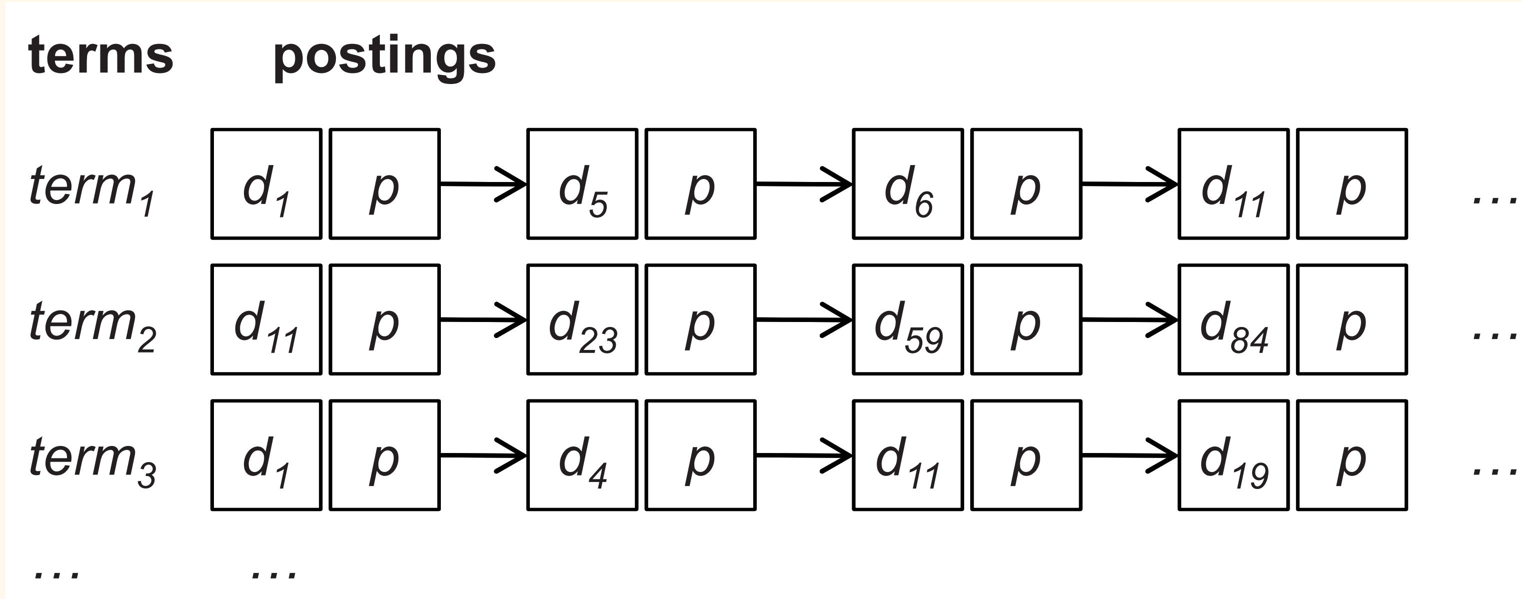
Example from Manning et al.'s "Introduction to Information Retrieval"



Resolving a query now becomes a *set operation* on the posting lists...

... i.e., resolving the query “Brutus AND Calpurnia” would simply require intersecting their respective posting lists.

Note the order of the postings!



Postings generally include some sort of "payload" or "metadata:"

- Term frequency
- Term positions within the document
- Context surrounding the term
- PoS information
- etc.

DocIDs can be assigned randomly, or according to some scheme (documents from same domain get similar IDs, higher PageRank gets lower IDs, etc.)

Indexing considerations:

A “web scale” corpus will involve billions of pages…

… and remember: some pages (news sites, etc.) become “stale” quickly, and must be reindexed often.

Bottleneck with inverted indexing: having to visit each document.

Can we speed it up by distributing the work?

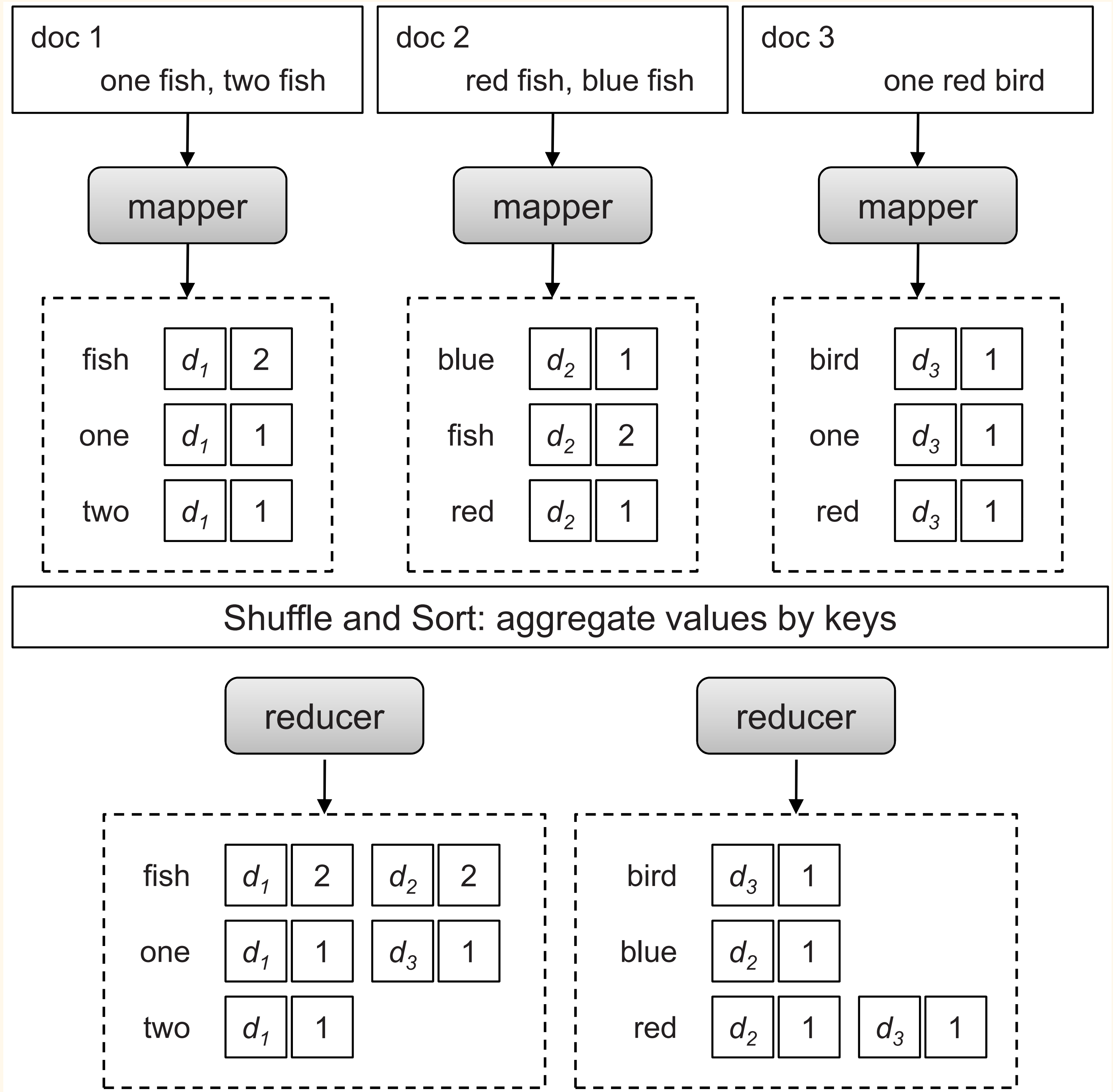
Of course we can!

This is what MapReduce was *invented* for...

Of course we can!

class Mapper
    procedure Map(docid n, doc d)
        $H \leftarrow$ new AssociativeArray
        for all term $t \in$ doc d **do**
            $H\{t\} \leftarrow H\{t\} + 1$
        for all term $t \in H$ **do**
            Emit(term t, posting $\langle n, H\{t\}\rangle$)

class Reducer
    procedure Reduce(term t, postings $[\langle n_1, f_1\rangle, \langle n_2, f_2\rangle \ldots]$)
        $P \leftarrow$ new List
        for all posting $\langle a, f\rangle \in$ postings $[\langle n_1, f_1\rangle, \langle n_2, f_2\rangle \ldots]$ **do**
            P.Add($\langle a, f\rangle$)
        P.Sort()
        Emit(term t, postings P)


doc 1
one fish, two fish
doc 2
red fish, blue fish
doc 3
one red bird
mapper
mapper
mapper
fish d_1 2
one d_1 1
two d_1 1
blue d_2 1
fish d_2 2
red d_2 1
bird d_3 1
one d_3 1
red d_3 1
Shuffle and Sort: aggregate values by keys
reducer
reducer
fish d_1 2 d_2 2
one d_1 1 d_3 1
two d_1 1
bird d_3 1
blue d_2 1
red d_2 1 d_3 1

```
class Mapper
    procedure Map(docid n, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(term t, posting ⟨n, H{t}⟩)
```

```
class Reducer
    procedure Reduce(term t, postings [⟨n1, f1⟩, ⟨n2, f2⟩ . . .])
        P ← new List
        for all posting ⟨a, f⟩ ∈ postings [⟨n1, f1⟩, ⟨n2, f2⟩ . . .] do
            P.Add(⟨a, f⟩)
        P.Sort()
        Emit(term t, postings P)
```

```
class Mapper
    method Map(docid n, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(tuple ⟨t, n⟩, tf H{t})

class Reducer
    method Initialize
        t_prev ← ∅
        P ← new PostingsList
    method Reduce(tuple ⟨t, n⟩, tf [f])
        if t ≠ t_prev ∧ t_prev ≠ ∅ then
            Emit(term t, postings P)
            P.Reset()
        P.Add(⟨n, f⟩)
        t_prev ← t
    method Close
        Emit(term t, postings P)
```

Being clever with our keys lets the runtime do more of the work.

Remember, keys arrive at the reducer in sorted order, guaranteed.

For very large corpora, it's important to be clever about how we store our postings lists on disk…

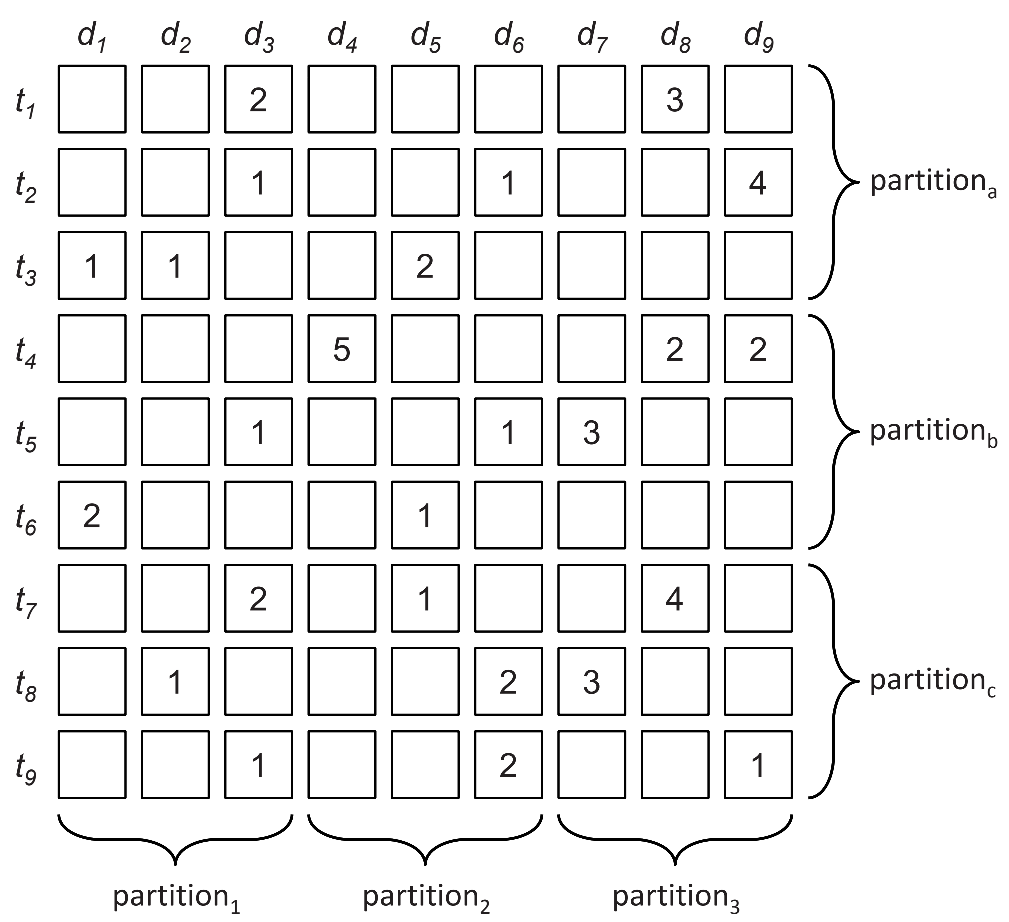
Out of scope for today, but worth reading Lin & Dyer's summary.

What about retrieval?

There are two main strategies for breaking up an index onto multiple machines:

1. Document sharding

2. Term sharding



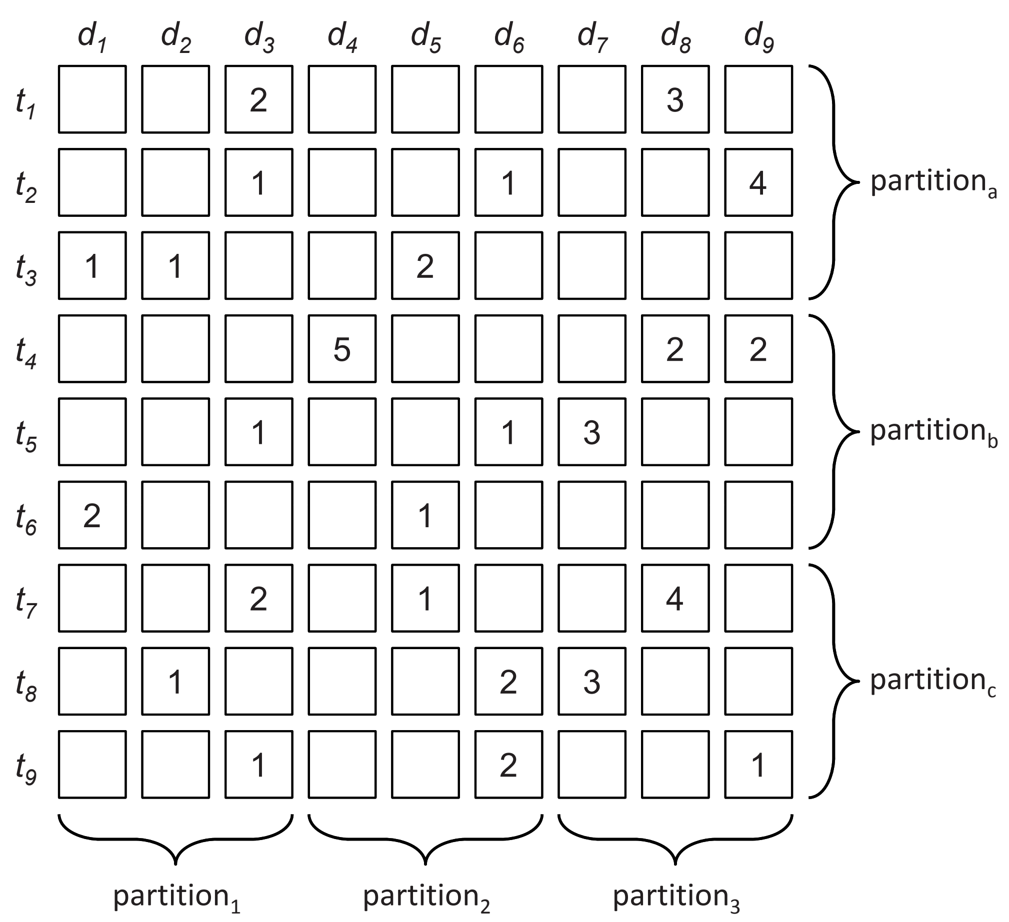
In document sharding, each machine has a portion of the documents, and queries are processed by each machine.

What about retrieval?

There are two main strategies for breaking up an index onto multiple machines:

1. Document sharding

2. Term sharding



In term sharding, each machine has a the full posting list for a subset of the terms, and only sees the part of the query that is relevant to it.

What about retrieval?

There are tradeoffs to each:

Document sharding is simpler to process, but requires a lot of wasted work…

- Each shard can process independent queries
- Easy to keep around more per-document information
- Each query must involve every shard (O(k*N) disk accesses, for k query terms and N document shards)

What about retrieval?

There are tradeoffs to each:

Term sharding is arguably more efficient to process, but requires a lot more complexity.

- *Far* less time spent doing I/O to access index (only *k* shards are involved for a *k*-word query)...

- Harder to have per-doc word-level information

- Much more network bandwidth needed (data from each matching doc must be aggregated)

What about retrieval?

There's no single right answer for all cases, but in general, document sharding seems better...

... largely due to lower search latency, which is the most relevant metric.

Game plan for today:

Quick overview of inverted indexing

Inverted indexing & Map-Reduce

Quick MT overview

Map-Reduce & MT Model Estimation

Quick MT overview:

$$P(f_1^m|e_1^l) = \sum_{a_1^m} P(f_1^m, a_1^m|e_1^l)$$

$$= \sum_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j})$$

Foreign Sentence

Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

Quick MT overview:

$$
\begin{aligned}
P(f_1^m|e_1^l) &= \sum_{a_1^m} P(f_1^m, a_1^m|e_1^l) \\
&= \sum_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j})
\end{aligned}
$$

Original Sentence

Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

Quick MT overview:

$$P(f_1^m|e_1^l) = \sum_{a_1^m} P(f_1^m, a_1^m|e_1^l)$$

$$= \sum_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j})$$

Language model

Quick MT overview:

$$P(f_1^m|e_1^l) = \sum_{a_1^m} P(f_1^m, a_1^m|e_1^l)$$

$$= \sum_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j})$$

Translation model

$$\hat{a}_1^m = \arg\max_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j})$$

Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

Quick MT overview:

$$P(f_1^m|e_1^l) = \sum_{a_1^m} P(f_1^m, a_1^m|e_1^l)$$

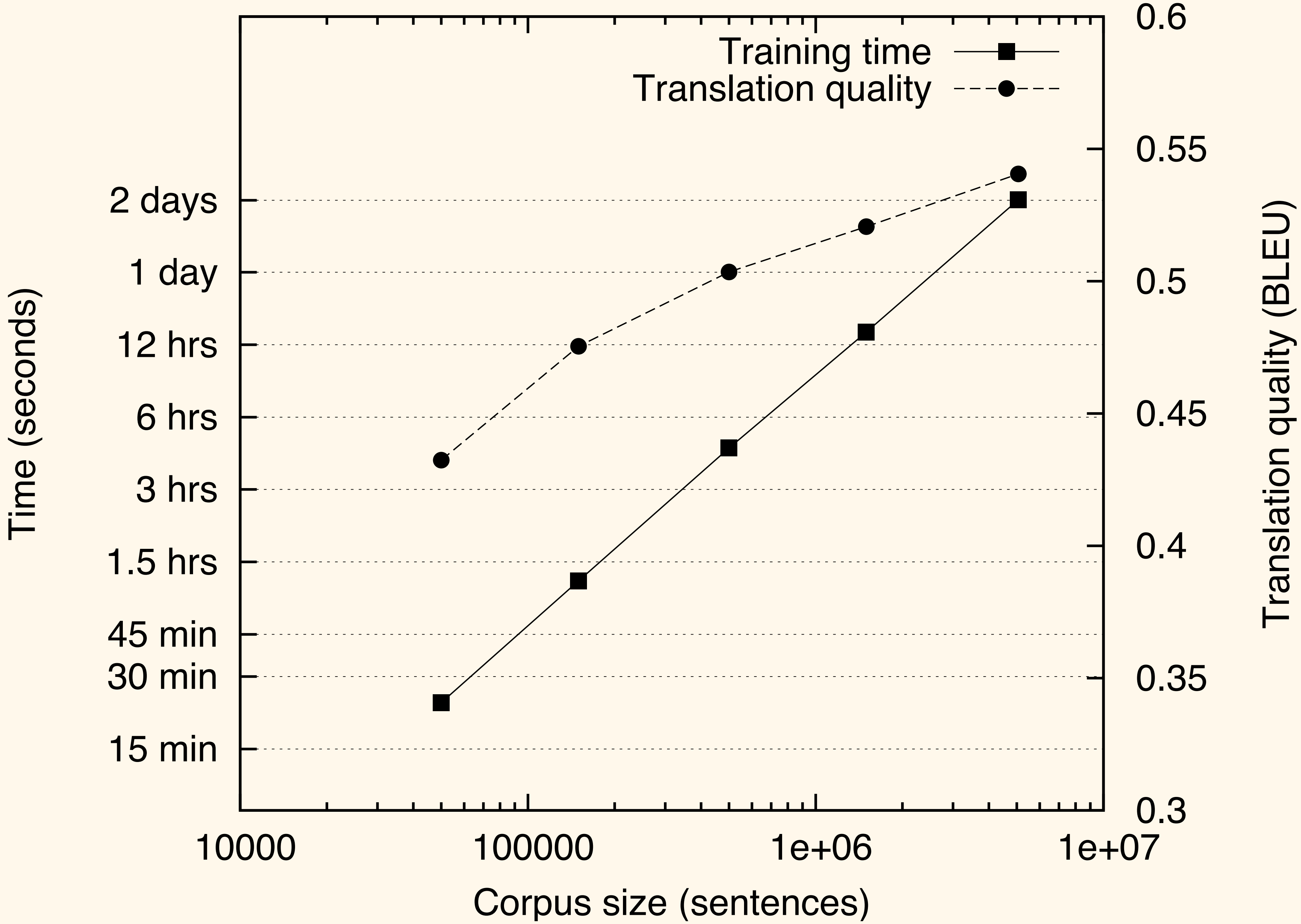
$$= \sum_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j})$$

1-best translation guess

$$\hat{a}_1^m = \arg\max_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j})$$

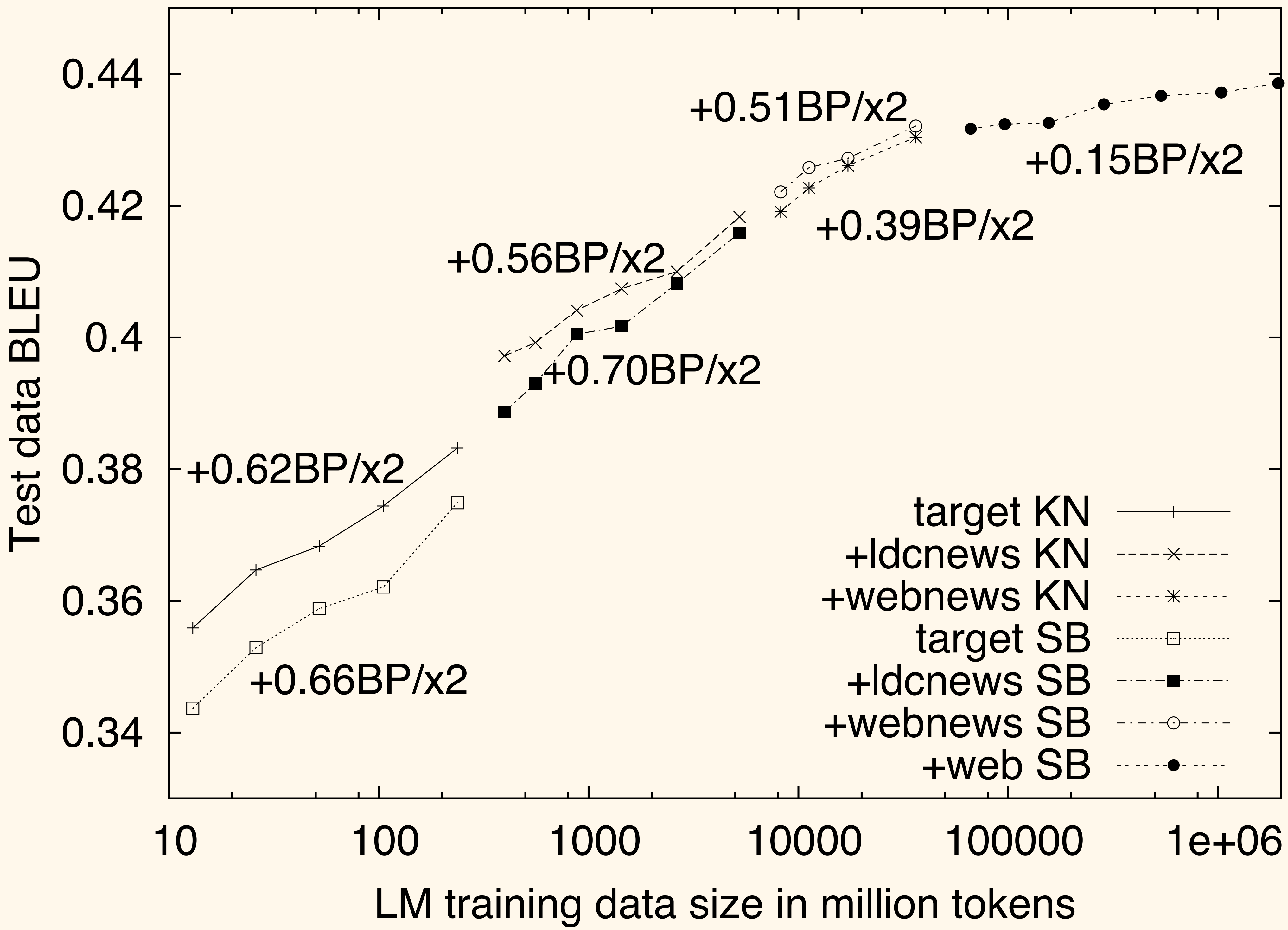
Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

More data == better LM (and MT)



Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

More data == better LM (and MT)



Large Language Models in Machine Translation. Thorsten Brants et. al., Proc. Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP), pg. 858-867, 2007.

How to estimate the LM?

$$P(w_1^L) = \prod_{i=1}^{L} P(w_i | w_1^{i-1}) \approx \prod_{i=1}^{L} \hat{P}(w_i | w_{i-n+1}^{i-1})$$

$$P_{MLE}(B|A) = \frac{c(A,B)}{c(A)} = \frac{c(A,B)}{\sum_{B'} c(A,B')}$$

When corpora get big...

Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

Large Language Models in Machine Translation. Thorsten Brants et. al., Proc. Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP), pg. 858-867, 2007.

MLE estimation in Map-Reduce:

Method 1

Map_1	$\langle A, B\rangle \rightarrow \langle\langle A, B\rangle, 1\rangle$
Reduce_1	$\langle\langle A, B\rangle, c(A, B)\rangle$
Map_2	$\langle\langle A, B\rangle, c(A, B)\rangle \rightarrow \langle\langle A, ^*\rangle, c(A, B)\rangle$
Reduce_2	$\langle\langle A, ^*\rangle, c(A)\rangle$
Map_3	$\langle\langle A, B\rangle, c(A, B)\rangle \rightarrow \langle A, \langle B, c(A, B)\rangle\rangle$
Reduce_3	$\langle A, \langle B, \frac{c(A,B)}{c(A)}\rangle\rangle$

Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

MLE estimation in Map-Reduce:

Method 1

Map_1	$\langle A, B\rangle \rightarrow \langle\langle A, B\rangle, 1\rangle$
Reduce_1	$\langle\langle A, B\rangle, c(A, B)\rangle$
Map_2	$\langle\langle A, B\rangle, c(A, B)\rangle \rightarrow \langle\langle A, ^*\rangle, c(A, B)\rangle$
Reduce_2	$\langle\langle A, ^*\rangle, c(A)\rangle$
Map_3	$\langle\langle A, B\rangle, c(A, B)\rangle \rightarrow \langle A, \langle B, c(A, B)\rangle\rangle$
Reduce_3	$\langle A, \langle B, \frac{c(A,B)}{c(A)}\rangle\rangle$

Method 2

Map_1	$\langle A, B\rangle \rightarrow \langle\langle A, B\rangle, 1\rangle; \langle\langle A, ^*\rangle, 1\rangle$
Reduce_1	$\langle\langle A, B\rangle, \frac{c(A,B)}{c(A)}\rangle$

Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

MLE estimation in Map-Reduce:

Method 1

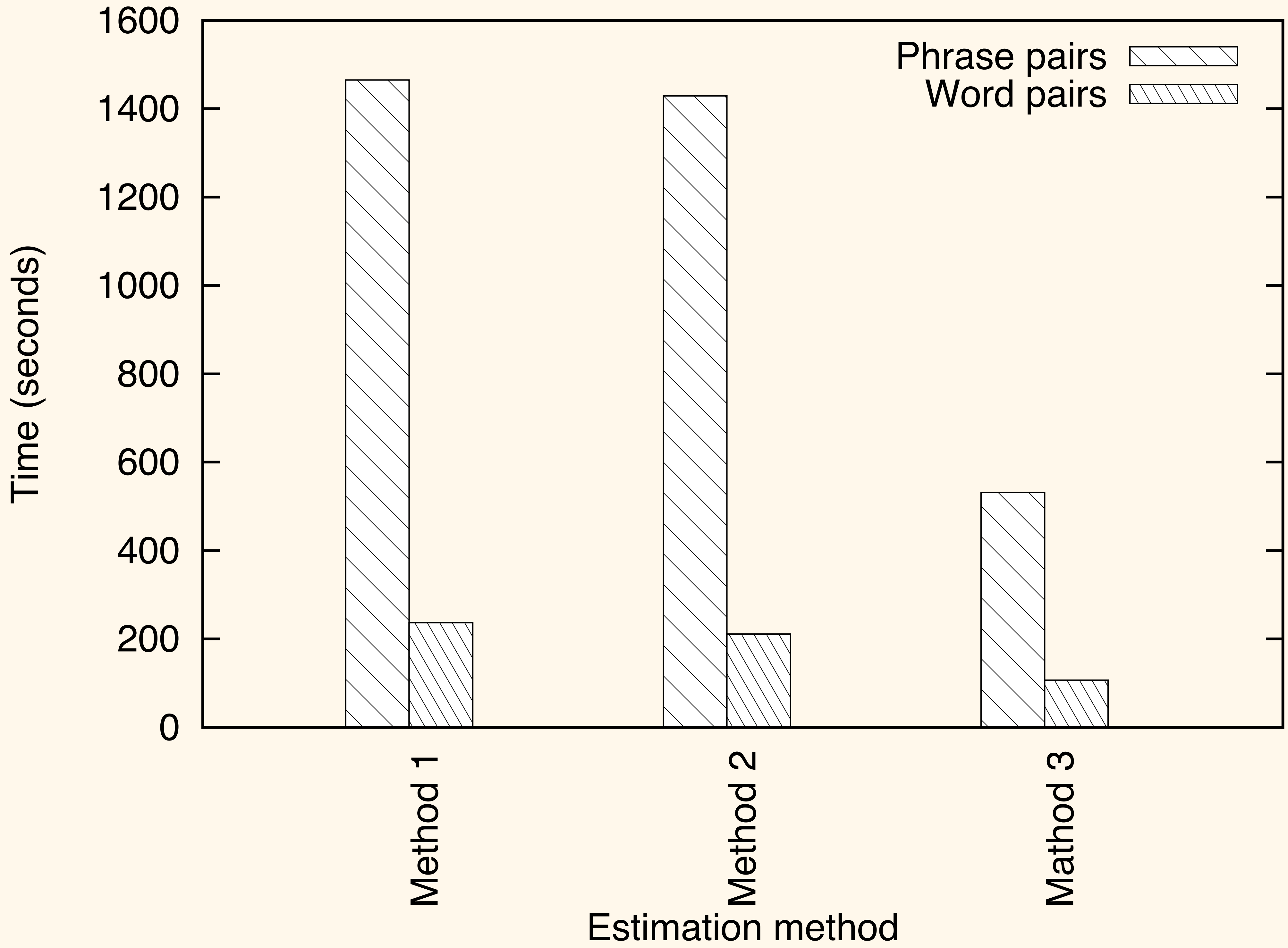
Map_1	$\langle A, B\rangle \rightarrow \langle\langle A, B\rangle, 1\rangle$
Reduce_1	$\langle\langle A, B\rangle, c(A, B)\rangle$
Map_2	$\langle\langle A, B\rangle, c(A, B)\rangle \rightarrow \langle\langle A, ^*\rangle, c(A, B)\rangle$
Reduce_2	$\langle\langle A, ^*\rangle, c(A)\rangle$
Map_3	$\langle\langle A, B\rangle, c(A, B)\rangle \rightarrow \langle A, \langle B, c(A, B)\rangle\rangle$
Reduce_3	$\langle A, \langle B, \frac{c(A,B)}{c(A)}\rangle\rangle$

Method 2

Map_1	$\langle A, B\rangle \rightarrow \langle\langle A, B\rangle, 1\rangle; \langle\langle A, ^*\rangle, 1\rangle$
Reduce_1	$\langle\langle A, B\rangle, \frac{c(A,B)}{c(A)}\rangle$

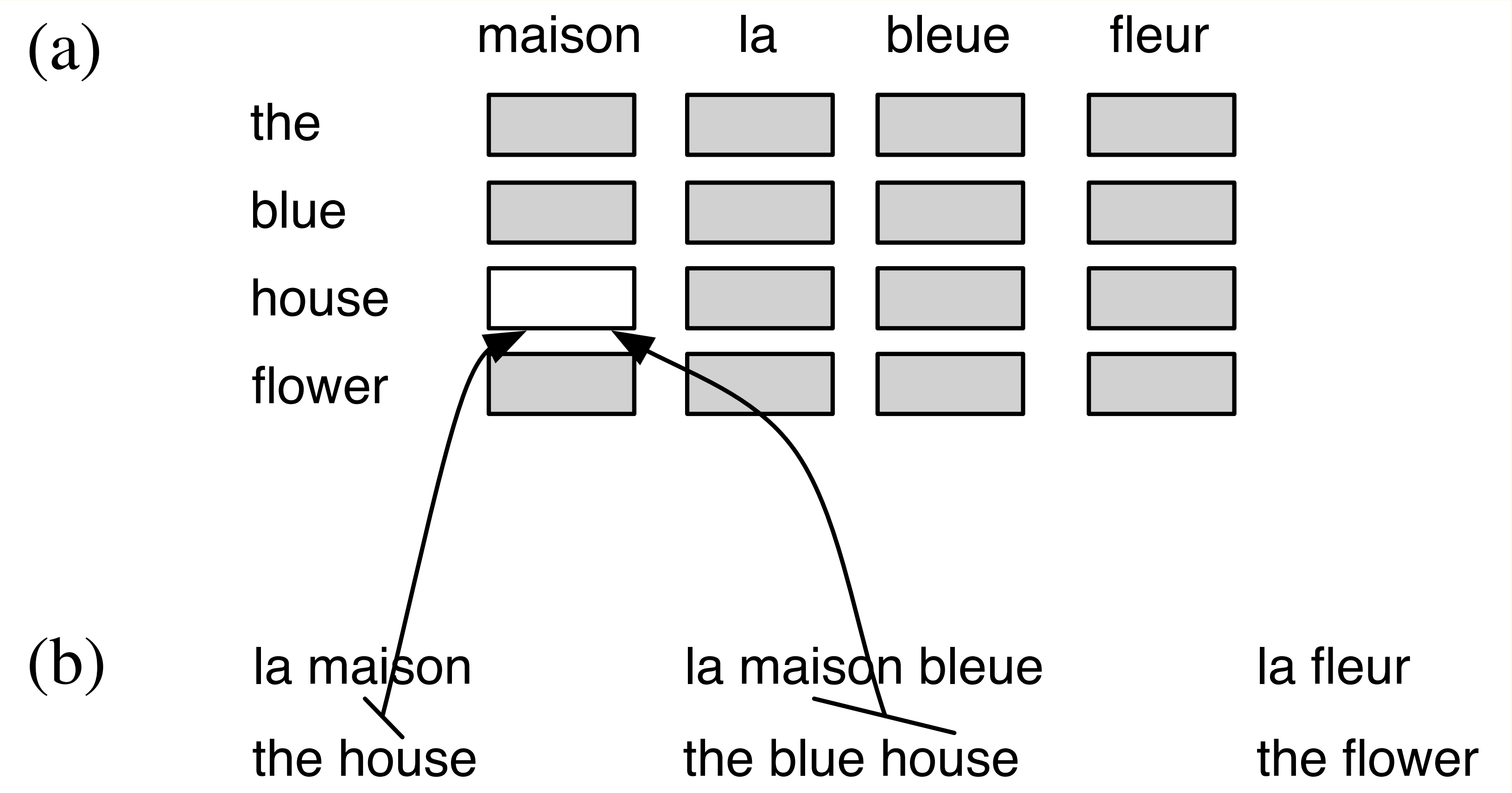
Method 3

Map_1	$\langle A, B_i\rangle \rightarrow \langle A, \langle B_i : 1\rangle\rangle$
Reduce_1	$\langle A, \langle B_1 : \frac{c(A,B_1)}{c(A)}\rangle, \langle B_2 : \frac{c(A,B_2)}{c(A)}\rangle \cdots\rangle$

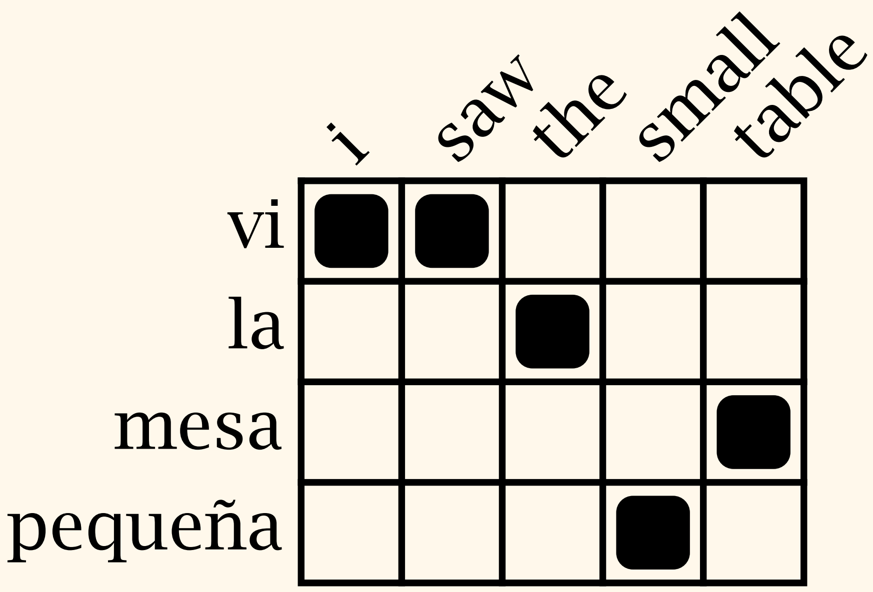


Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce, Christopher Dyer et. al., Proc. ACL Workshop on Statistical Machine Translation, pg. 199-207, 2008. slides

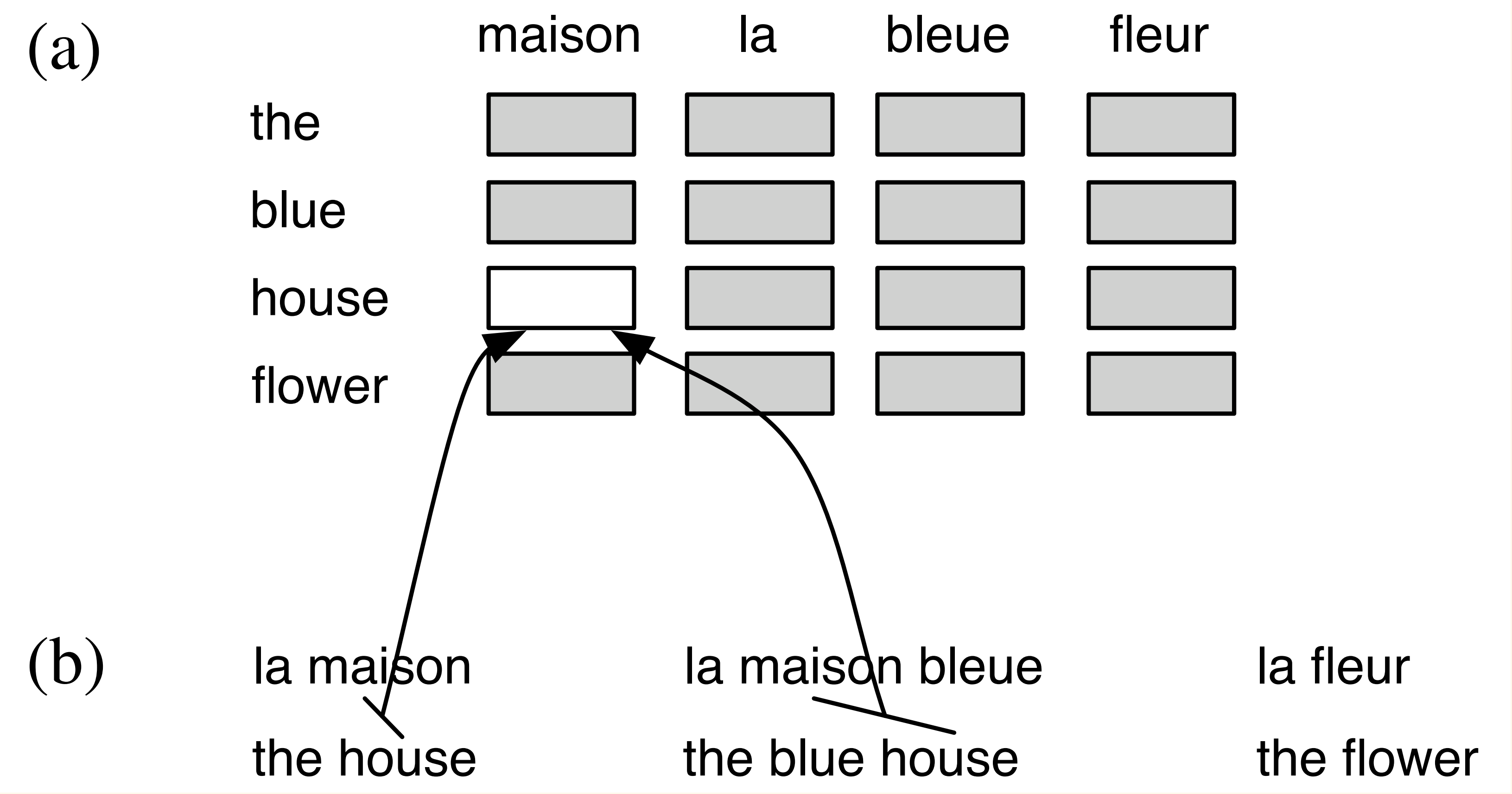
We can use a similar method for estimating parameters for alignments:

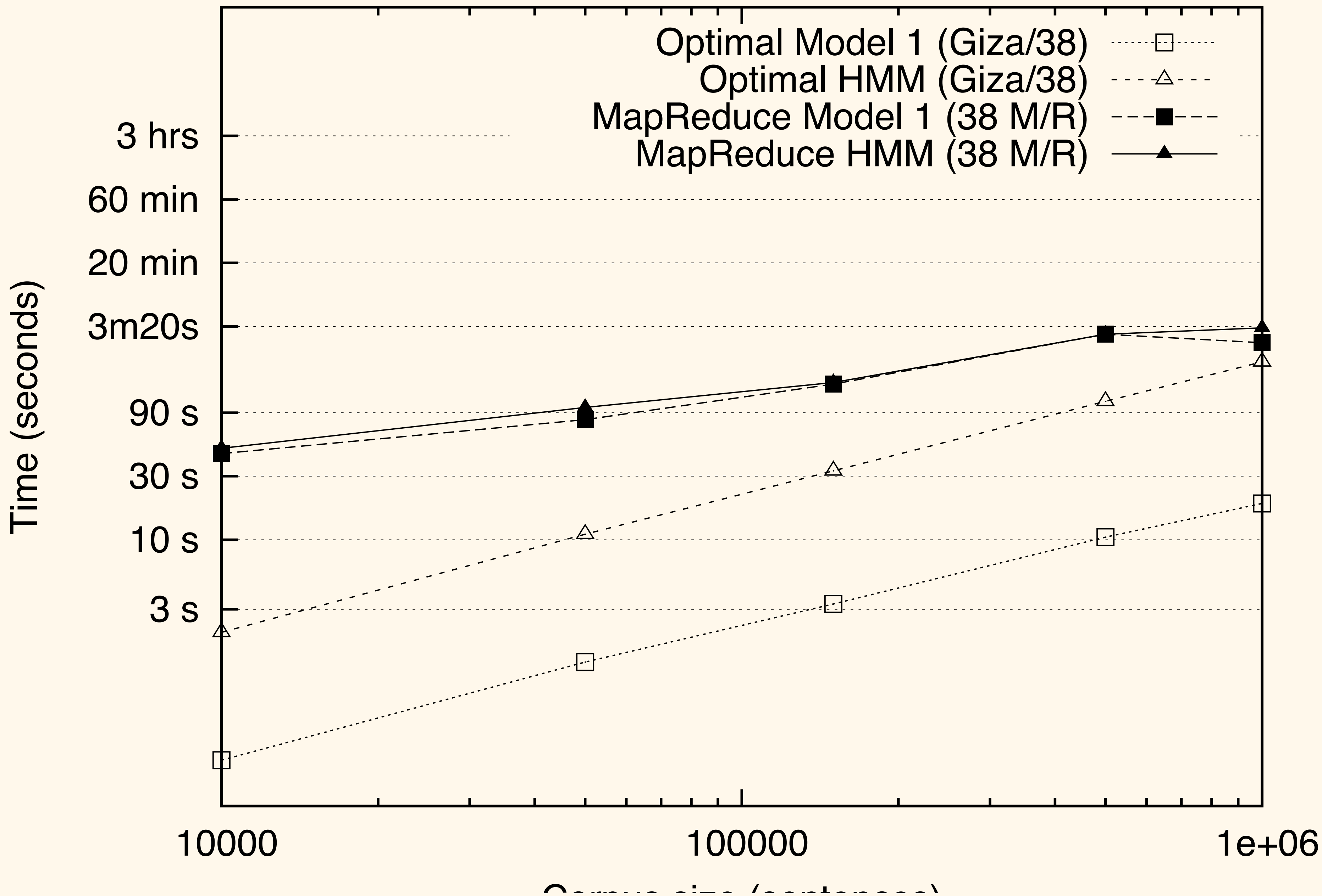


We can use a similar method for estimating parameters for alignments:

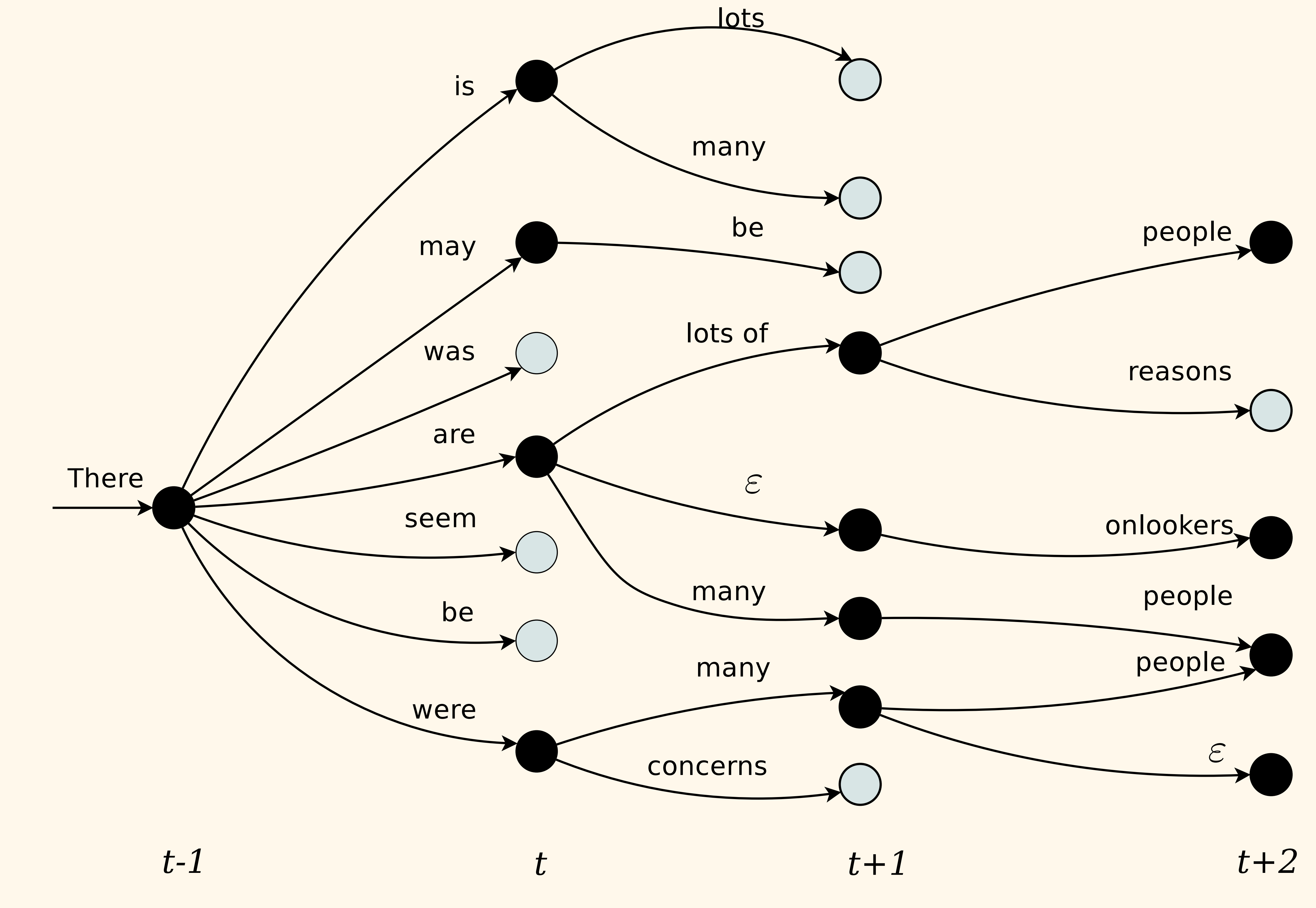


$$P(a_1^m | e_1^l, f_1^m)$$


(a)
maison
la
bleue
fleur
the
blue
house
flower
(b)
la maison
the house
la maison bleue
the blue house
la fleur
the flower



Optimal Model 1 (Giza/38)
Optimal HMM (Giza/38)
MapReduce Model 1 (38 M/R)
MapReduce HMM (38 M/R)
3 hrs
60 min
20 min
3m20s
90 s
30 s
10 s
3 s
Time (seconds)
10000
100000
1e+06


There
is
may
was
are
seem
be
were
lots
many
be
lots of
ε
many
many
concerns
people
reasons
onlookers
people
people
ε
t-1
t
t+1
t+2