Mapreduce and (in) Search

Amund Tveit (PhD)

amund@atbrox.com
http://atbrox.com

TOC

1. Brief about my background
2. Mapreduce Overview
3. Mapreduce in Search
4. Advanced Mapreduce Example

Brief about my background

- Now: Co-founder of Atbrox
    - Search & Mapreduce
- PhD in Computer Science
    - http://amundtveit.info/publications/
- Past: Googler for 4 years:
    - Cluster Infrastructure
    - Nordic Search (and Maps) Quality
    - Google News for iPhone

Details: http://no.linkedin.com/in/amundtveit

- Interested in how mapreduce is used (algorithmic patterns)
    - http://mapreducepatterns.org
    - Working on projects using mapreduce in search and other large-scale problems
- Passionate about search and search technology

- Less known trivia:
    - shared office with your professor back in 2000

Part 1

mapreduce

What is Mapreduce?

atbrox

Mapreduce is a concept and method for typically batch-based large-scale parallelization. It is inspired by functional programming's map() and reduce() functions

Nice features of mapreduce systems include:

- reliably processing job even though machines die
- parallization en-large, e.g. thousands of machines for terasort

Mapreduce was invented by the Google fellows below:
http://labs.google.com/papers/mapreduce.html

Jeff D.

Sanjay G.

Mapper

atbrox

- Processes one key and value pair at the time, e.g.
- **word count**
  - map(key: uri, value: text):
    - for word in tokenize(value)
    - emit(word, 1) // found 1 occurence of word

- **inverted index**
  - map(key: uri, value: text):
    - for word in tokenize(value)
    - emit(word, key)

Reducers

Reducers processes one key and all values that belong to it, e.g.

- **word count**
  - reduce(key: word type, value: list of 1s):
    - emit(key, sum(value))
- **inverted index**
  - reduce(key: word type, value: list of URIs):
    - emit(key, value) *// e.g. to a distr. hash table*

Combiner

atbrox

Combiner functions is the subset of reducer functions where reducer functions can be written as recurrence equations, e.g.
$sum_{n+1} = sum_n + x_{n+1}$

This property happens (surprisingly) often and can be used to speed up mapreduce jobs (dramatically) by putting the combiner function as an "afterburner" on the map functions tail.

But sometimes, e.g. for advanced machine learning algorithms reducers are more advanced and can't be used as combiners

Shuffler - the silent leader

atbrox

When the mapper emits a key, value pair - the shuffler does the actual work in shipping it to the reducer, the addressing is a function of the key, e.g.

    chosen reducer = hash(key)%num reducers

but could basically be any routing function.

Q. When is changing the shuffle function useful?
A. When data distributed is skewed, e.g. according to Zip's law. Examples of this are stop words in text (overly frequent) and skewness in sales numbers (where bestsellers massively outnumbers items in the long tail)

Part 2

mapreduce in search

What is important in search?

1. **Precision**
   - results match query but primarily user intent
2. **Recall**
   - not missing important results
3. **Freshness**
   - timely recall, .e.g for news
4. **Responsiveness**
   - time is money, but search latency is minus-money
5. **Ease-of-use**
   - few input widgets and intelligent query understanding
6. **Safety**
   - not providing malware or web spam

Recap: What is Search?

1. **Getting data**
2. **Processing data**
3. **Indexing data**
4. **Searching data**
5. **Search Quality (maintain and improve)**
6. **Ads Quality (which pays for the fun)**

But:

- **not necessarily in that order**
- **and with feedback loops**

.

<u>searching</u> might lead to <u>getting</u> usage data used in both <u>processing</u> and <u>indexing</u>

Getting data - SizeOf(the Web)

- **2005** - Google and Yahoo competing with:
  - ~20 Billion ($20*10^9$) documents in their indices
- **2008** - Google seeing 1 trillion ($1*10^{12}$) simultaneously existing unique urls
  - => PetaBytes (i.e. thousands of harddisks)

Bibliography:

- http://www.ysearchblog.com/?p=172
- http://googleblog.blogspot.com/2005/09/we-wanted-something-special-for-our.html
- http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html
- http://www.chato.cl/research/crawling_thesis
- http://research.google.com/people/jeff/WSDM09-keynote.pdf
- http://www.morganclaypool.com/doi/abs/10.2200/S00193ED1V01Y200905CAC006

MR+Search: Get Data - Dups I

Detect:
- **exact duplicates**
- **near-duplicates**
- **scraped/combined content**

with mapreduce:

Naively $O(N^2)$ problem - compare all N documents with each other, but can be pruned by:
- comparing only document which share substrings
- compare only documents of similar size

MR+Search:Get data - Dups II

atbrox

mapreduce job 1:

```
map(key: uri, value: text)
  create hashes (shingles) of substrings of content
  foreach hash in hashes: emit(key, hash)

reduce(key: uri, values: list of hashes)
  size = number of hashes in values
  outputvalue = concat(key, size)
  foreach hash in hashes: emit(hash, outputvalue)
```

mapreduce job 2:

```
map(key: hash, value: uri-numhashes)
  emit(key, value)

reduce(key: hash, value: list of uri-numhashes)
  emit all unique pairs of uri-numhash in value
  // e.g. (uri0-4, uri1-7), (uri0-4, uri13-5), ..
```

MR+Search: Dups III

atbrox

mapreduce job 3:

```
map(key: uri-numhashes, value: uri-numhashes)
  emit(key, value)

reduce(key: uri-numhashes, value: list of uri-numhashes)
  // e.g. values for key uri0-4 could be
  // uri1-7, uri1-7, uri13-5, uri33-7
  for each unique uri-numhash h in values
    alpha = count of h occurences in values
    doc_similarity = alpha/(key's numhashes + h's numhashes
            - alpha
    output_key = key's uri concatenated with h's uri
    emit(output_key, doc_similarity)
```

References:

- http://www.cs.princeton.edu/courses/archive/spr05/cos598E/bib/Princeton.pdf
- http://www.cs.uwaterloo.ca/~kmsalem/courses/CS848W10/presentations/Hani-proj.pdf
- http://uwspace.uwaterloo.ca/bitstream/10012/5750/1/Khoshdel%20Nikkhoo_Hani.pdf
- http://glinden.blogspot.com/2008/04/detecting-near-duplicates-in-big-data.html
- http://infolab.stanford.edu/~manku/papers/07www-duplicates.ppt
- http://simhash.googlecode.com/svn/trunk/paper/SimHashWithBib.pdf

MR+Search: Processing data I atbrox

E.g. a sequence of structurally similar map(reduce) steps where one updates a package with new fields:

- **Content Normalization** (to text and remove boilerplate)
  - **map(key: URI, value: rawcontent)**
    - **create new documentpackage**
    - **documentpackage.setRawContent(rawcontent)**
    - **documentpackage.setText(stripBoilerPlate(ToText(rawContent))**
    - **emit(key, documentpackage)**
- **Entity Extraction**
  - **map(key: URI, value: documentpackage)**
    - **names = findPersonNameEntities(documentpackage.getText())**
    - **documentpackage.setNames(names)**
    - **emit(key, documentpackage)**
- **Sentiment Analysis**
  - **map(key: URI, value: documentpackage)**
    - **sentimentScore = calculateSentiment(documentpackage.getText())**
    - **documentpackage.setSentimentScore(sentimentScore)**
    - **emit(key, documentpackage)**

MR+Search: Processing data II

Creating a simple hypothetical ranking signal with mapreduce:
=> ratio of wovels per page

```
map(key: URI, value: documentpackage)
   numvowels, numnonvowels = 0
   for each character in documentpackage.getText():
       if isWowel(character)
          ++numvowels
       else
          ++numnonvowels
   vowelratio = numvowels / (numvowels+numnonvowels)
   documentpackage.setVowelRatio( vowelratio)
   emit(key, documentpackage)
```

MR+Search: Indexing data I

atbrox

Inverted index among the "classics" of mapreduce example, it resembles word count but outputs the URI instead of occurence count

```
map(key: URI, value: text)
  for word in tokenize(value)
     emit(word, key)

reduce(key: wordtype, value: list of URIs)
  output(key, value) // e.g. to a distributed hash
```

MR+Search: Indexing data II

How about positional information?

```
map(key: URI, value: text)
  wordposition = 0
  foreach word in tokenize(value)
     outputvalue = concat(key, wordposition)
     ++wordposition
     emit(word, outputvalue)

reduce(key: wordtype, value: list of URI combined with word position)
  create positional index for the given word (key) in the format
  output(key, newvalue) // e.g. to a distributed hash
```

MR+Search: Indexing data III

How about bigram or n-gram indices?

```
map(key: URI, value: text)
  bigramposition = 0
  foreach bigram in bigramtokenizer(value)
     outputvalue = concat(key, wordposition)
     ++bigramposition
     emit(bigram, outputvalue)

reduce(key: bigramtype, value: list of URI combined with bigram position)
  create positional bigram index for the given bigram (key) in the format
  output(key, newvalue) // e.g. to a distributed hash
```

MR+Search: Indexing data IV

How about indexing of stems or synonyms?

```
map(key: URI, value: text)
  foreach word in tokenizer(value)

     synonyms = findSynonyms(word)
     stem = createStem(word)
     emit(word, key)
     emit(stem, key)
     foreach synonym in synonyms
       emit(synonym, key)

reduce(key: wordtype, value: list of URI)
  output(key, value) // e.g. to a distributed hash
```

MR+Search: Searching data I

A lot of things typically happens at search time, the query is "dissected", classified, interpreted and perhaps rewritten before querying the index. This can generate clickthrough log data where we can find most clicked-on uris:

mapreducejob 1

```
map(key: query, value: URI clicked on)
  emit(value, key)

reduce(key: URI clicked on, value: list of queries)

emit(count(value), key)) // (count, query)
```

mapreduce job 2

```
map(key: uri count, value: uri)
  emit(key, value)   // (count, query)

reduce(key: uri count, value: list of uris)
   emit(key, value)
 // generate list of uris per clickthrough count
```

MR+Search: Searching data II

atbrox

Or we can find which the number of clicks per query *(note: queries with relatively few clicks probably might have issues with ranking)*

mapreducejob 1

```
map(key: query, value: URI clicked on)
  emit(value, key)

reduce(key: URI clicked on, value: list of queries)
  foreach unique query q in value // get all clicks per query for this uri
     outputvalue = count number of occurences (clicks) of q in value
     emit( q, outputvalue)
```

mapreduce job 2

```
map(key: query, value: count)
  outputvalue = sum(value)
  emit(key, value)

reduce(key: query, value: list of counts)
   emit(sum(value), key)

// get all clickthroughs per query for all urls
```

MR+Search: Quality

atbrox

Changing how processing and indexing mapreduce jobs work is likely to effect search quality (e.g. precision and recall), this can be evaluated with mapreduce, e.g. comparing the old and new set of indices by running querying both set of indices with the same (large) query log and compare differences in results.

Task: how will you do that with mapreduce?

References:

- http://www.cs.northwestern.edu/~pardo/courses/mmml/papers/collaborative_filtering/crowdsourcing_for_relevance_evaluation_SIGIR08.pdf

MR+Search: Ads Quality

Ads are commercial search results, they should have similar requirements to relevancy as "organic" results, but have less text "to rank with" themselves (~twitter tweet size or less), but fortunately a lot of metadata (about advertiser, landing pages, keywords bid for etc.) that can be used to measure, improve and predict their relevancy

References:

- http://www.wsdm-conference.org/2010/proceedings/docs/p361.pdf
- http://web2py.iiit.ac.in/publications/default/download/techreport.pdf.a373bbf4a5b76063.4164436c69636b5468726f75676852617465494949544485265706f72742e706466.pdf
- http://pages.stern.nyu.edu/~narchak/wfp0828-archak.pdf
- http://research.yahoo.com/files/cikm2008-search%20advertising.pdf

MR+Search: Other Topics

Plenty of mapreduce+search related topics haven't been covered, but here are some to consider looking at:

- machine translation
- clustering
- graph algorithms
- spam/malware detection
- personalization

References:

- http://www.google.com/research/pubs/och.html
- http://www.usenix.org/event/hotbots07/tech/full_papers/provos/provos.pdf
- https://cwiki.apache.org/confluence/display/MAHOUT/Algorithms
- http://code.google.com/edu/submissions/mapreduce-minilecture/listing.html
    - http://www.youtube.com/watch?v=1ZDybXl212Q (clustering)
    - http://www.youtube.com/watch?v=BT-piFBP4fE (graphs)

Part 3

advanced mapreduce example

Adv. Mapreduce Example I

This gives an example of how to port a machine learning classifier to mapreduce with discussion about optimizations

The Classifier: Tikhonov Regularization with a square loss function (*this family includes: proximal support vector machines, ridge regression, shrinkage regression and regularized least-squares classification)*

(omega, gamma) = (I/mu + E^T*E)$^{-1}$*(E^T*D*e)

D - a matrix of training classes, e.g. [[-1.0, 1.0, 1.0, ..]]
A - a matrix with feature vectors, e.g. [[2.9, 3.3, 11.1, 2.4], ..]
e - a vector filled with ones, e.g [1.0, 1.0, .., 1.0]
E = [A -e]
mu = scalar constant # used to tune classifier
D - a diagonal matrix with -1.0 or +1.0 values (depending on the class)

Adv. Mapreduce Example II

How to classify an incoming feature vector x
class = x^T*omega - gamma

The classifier expression in Python (numerical python):
(omega, gamma) = (I/mu + E.T*E).I*(E.T*D*e)

The expression have a (nice) additive property such that:
(omega, gamma) = (I/mu + E_1.T*E_1 + E_2.T*E_2).I*(E_1.T*D_1*e + E_2.T*D_2*e)

With induction this becomes:
(omega, gamma) = (I/mu + E_1.T*E_1 + .. + E_i.T*E_i).I*
(E_1.T*D_1*e + .. + E_i.T*D_i*e)

Adv. Mapreduce Example III

atbrox

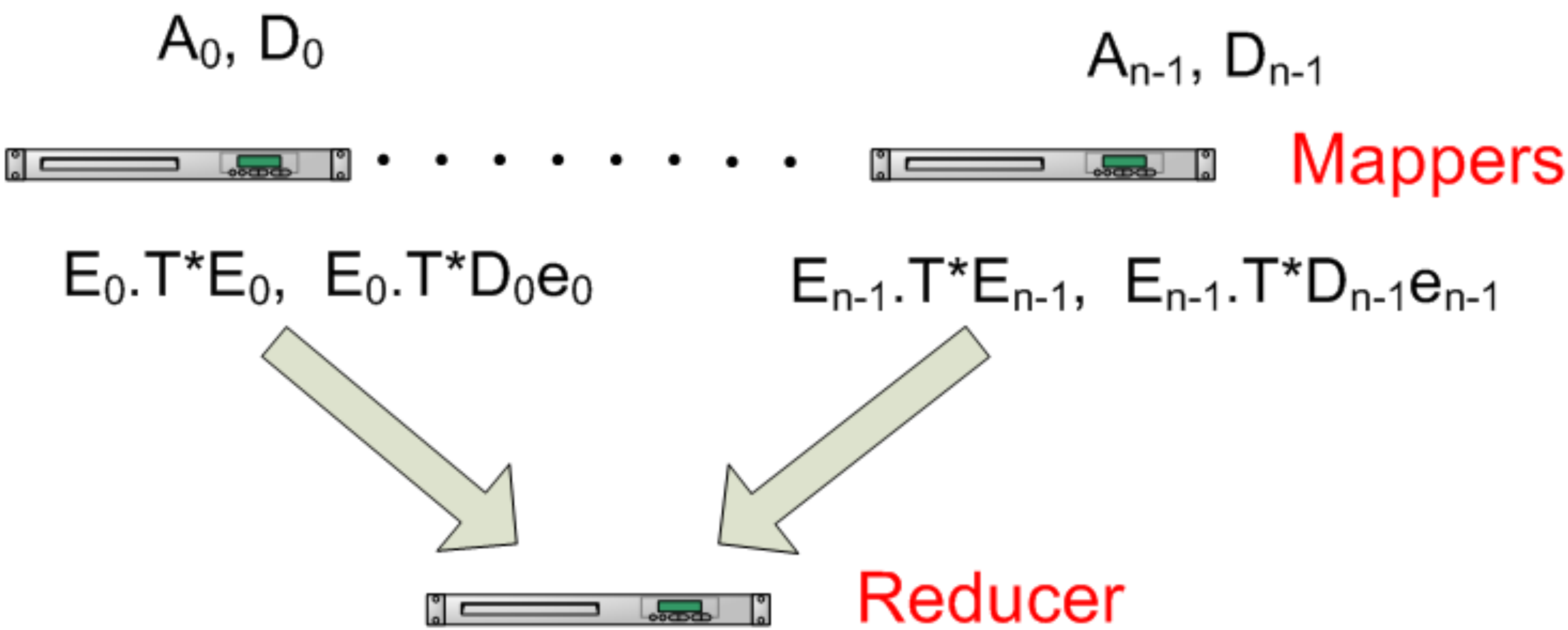
Brief Recap

D - a matrix of training classes, e.g.
[[-1.0, 1.0, 1.0, ..]]
A - a matrix with feature vectors, e.g.
[[2.9, 3.3, 11.1, 2.4], ..]
e - a vector filled with ones, e.g [1.0, 1.0, .., 1.0]
E = [A -e]
mu = scalar constant # used to tune classifier

A and D represent distributed training data, e.g. spread out on many machines or on distributed file system. Given the additive nature of the expression we can parallelize the calculation of E.T*E and E.T*De

Adv. Mapreduce Example IV

atbrox



(omega, gamma) = (I/mu+E.T*E).I*(E.T*D*e)

Adv. Mapreduce Example V

```
def map(key, value):
   # input key= class for one training example, e.g. "-1.0"
   classes = [float(item) for item in key.split(",")]   # e.g. [-1.0]
   D = numpy.diag(classes)

   # input value = feature vector for one training example, e.g. "3.0, 7.0, 2.0"
   featurematrix = [float(item) for item in value.split(",")]
   A = numpy.matrix(featurematrix)

   # create matrix E and vector e
   e = numpy.matrix(numpy.ones(len(A)).reshape(len(A),1))
   E = numpy.matrix(numpy.append(A,-e,axis=1))

   # create a tuple with the values to be used by reducer
   # and encode it with base64 to avoid potential trouble with '\t' and '\n' used
   # as default separators in Hadoop Streaming
   producedvalue = base64.b64encode(pickle.dumps( (E.T*E, E.T*D*e) )

   # note: a single constant key "producedkey" sends to only one reducer
   # somewhat "atypical" due to low degree of parallism on reducer side
   print "producedkey\t%s" % (producedvalue)
```

Adv. Mapreduce Example VI

atbrox

```
def reduce(key, values, mu=0.1):
  sumETE = None
  sumETDe = None

  # key isn't used, so ignoring it with _ (underscore).
  for _, value in values:
    # unpickle values
    ETE, ETDe = pickle.loads(base64.b64decode(value))
    if sumETE == None:
      # create the I/mu with correct dimensions
      sumETE = numpy.matrix(numpy.eye(ETE.shape[1])/mu)
    sumETE += ETE

    if sumETDe == None:
      # create sumETDe with correct dimensions
      sumETDe = ETDe
    else:
      sumETDe += ETDe

    # note: omega = result[:-1] and gamma = result[-1]
    # but printing entire vector as output
    result = sumETE.I*sumETDe
    print "%s\t%s" % (key, str(result.tolist()))
```

Adv. Mapreduce Example VII

Mapper Increment Size really makes a difference

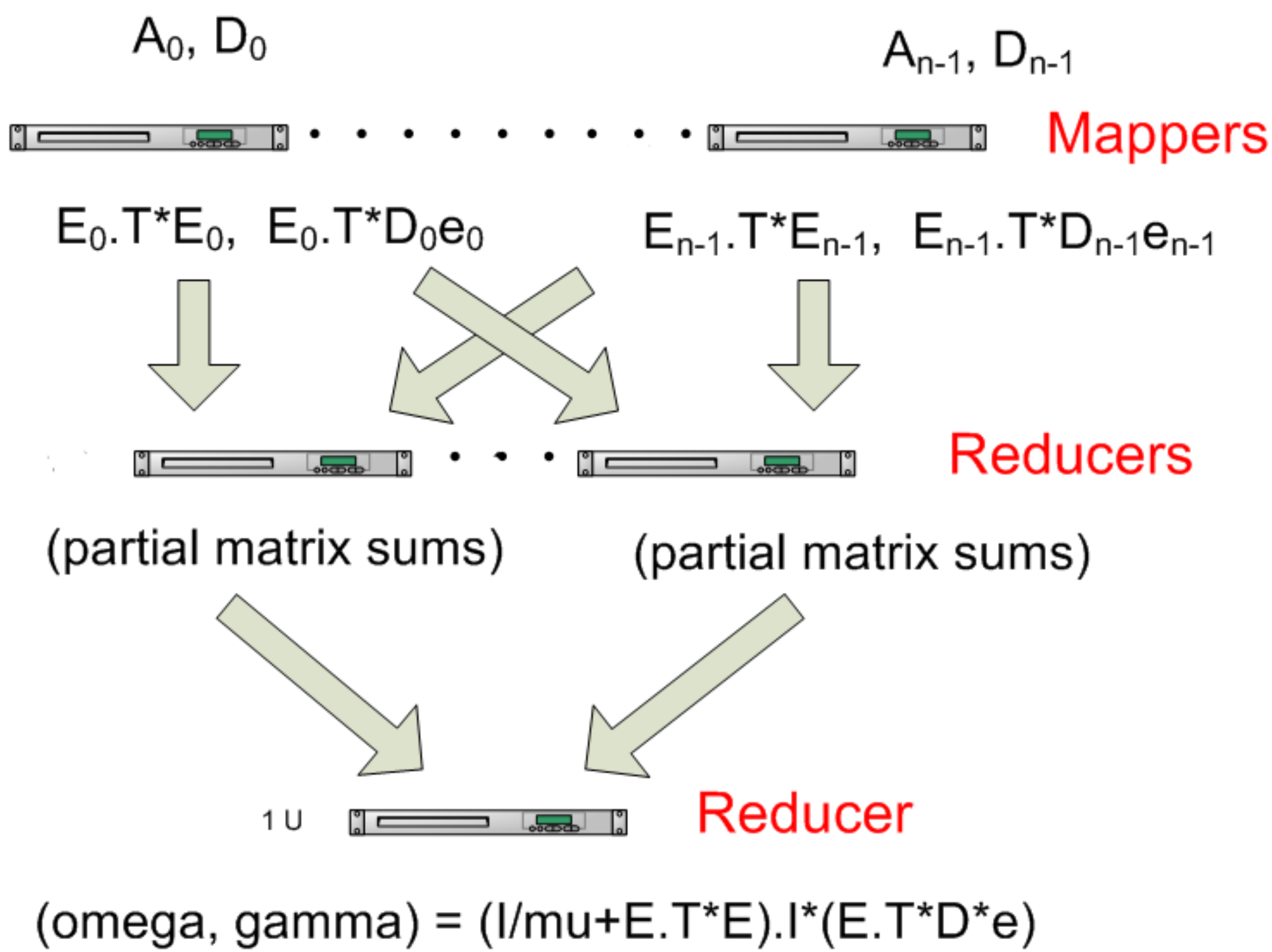
E.T*E and E.T*D*e given to reducer are independent of number of training data(!)
==> put as much training data in each E.T*E and E.T*D*e package

Example:

- Assume 1000 mappers with 1 billion training examples each (web scale :) and 100 features per training example
- If putting all billion examples into one E.T*E and E.T*D*e package
  - reducer needs to summarize 1000 101x101 matrices (not bad)
- Or if sending 1 example per E.T*E and E.T*D*e package
  - reducer needs to summarize 1 trillion (10^12) 101x101 matrices (intractable)

Adv. Mapreduce Example VIII

Avoid stressing the reducer



Part 4

landing

Mapreduce Patterns

Map() and Reduce() methods typically follow patterns, a recommended way of grouping code with such patterns is:

extracting and generalize fingerprints based on:

- **loops**: e.g "do-while", "while", "for", "repeat-until" => "loop"
- **conditions**: e.g. "if" and "switch" => "condition"
- **emits**
- **emit data types** (if available)

the map() method for both word count and index would be:

```
map(key, value)
  loop
      emit(key:string, value:string)
```

How to start with mapreduce?

atbrox

Either download hadoop (comes with mapreduce):

- http://hadoop.apache.org/common/releases.html
- http://www.cloudera.com/downloads/
- http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html

Or use Amazon Elastic Mapreduce (cloud service):

- http://aws.amazon.com/elasticmapreduce/
- http://docs.amazonwebservices.com/ElasticMapReduce/latest/GettingStartedGuide/

Summary

atbrox

- Have given a brief introduction to:
  - mapreduce
  - mapreduce in search
  - in-depth mapreduce example (classification)

- **Any questions, Comments?**
- **Further reading:**
  - http://mapreducepatterns.org
    - algorithms with mapreduce
  - http://atbrox.com/about/
    - misc. mapreduce blog posts (by me)
- **Wanting to work with mapreduce?**
  - amund@atbrox.com