MapReduce:
Simplified Data Processing on Large Clusters

CS 230
Bruce Donald
MapReduce:
Simplified Data Processing on Large Clusters

CS 230

Acknowledgement: many of these slides are from Dan Weld’s class at U. Washington (who in turn made his slides based on those by Jeff Dean, Sanjay Ghemawat, Google, Inc.)
Motivation

- **Large-Scale Data Processing**
  - Want to use 1000s of CPUs
    - But don’t want hassle of *managing* things

- **MapReduce provides**
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates
Map/Reduce

- Map/Reduce
  - Programming model from Scheme
- Many problems can be phrased this way
- Easy to distribute across nodes
- Nice retry/failure semantics
(define map
  (lambda (f lst)
    (if (null? lst)
        lst
        (cons (f (first lst)) (map f (rest list))))))

(define accumulate
  (lambda (op base lst)
    (cond ((null? lst) base)
          (else (op (first lst) (accumulate op base (rest lst)))))))

(accumulate + 0 '(1 2 3 4 5 6))
(define map
  (lambda (f lst)
    (if (null? lst)
        lst
        (cons (f (first lst)) (map f (rest list))))))

(define reduce
  (lambda (op base lst)
    (cond ((null? lst) base)
          (else (op (first lst) (reduce op base (rest lst)))))))

(reduce + 0 '(1 2 3 4 5 6))
(define map
  (lambda (f lst)
    (if (null? lst)
        lst
        (cons (f (first lst)) (map f (rest list))))))

(define reduce
  (lambda (op base lst)
    (cond ((null? lst) base)
          (else (op (first lst) (reduce op base (rest lst)))))))

(reduce + 0 '(1 2 3 4 5 6))

Idea:
- Write our program using Map and Reduce as much as possible.
- The serial versions of Map and Reduce will be automatically replaced by parallel implementations on a huge cluster.
- The system will handle all the details, you just write the functional program.
- Map and Reduce become “Parallel Operators”
(define map
    (lambda (f lst)
        (if (null? lst)
            lst
            (cons (f (first lst)) (map f (rest list))))))

(define reduce
    (lambda (op base lst)
        (cond ((null? lst) base)
              ((else (op (first lst) (reduce op base (rest lst))))))))

(reduce + 0 '(1 2 3 4 5 6))

Idea:

- Write our program using Map and Reduce as much as possible.

- The serial versions of Map and Reduce will be automatically replaced by parallel implementations on a huge cluster.

- The system will handle all the details, you just write the functional program.

- Map and Reduce become “Parallel Operators”: “Paralleize This!”
Map in Lisp (Scheme)

- `(map f list [list2 list3 ...])`
- `(map square '(1 2 3 4))
  - `(1 4 9 16)`
- `(reduce + 0 '(1 4 9 16))
  - `→ 30`
- `(reduce + 0 (map square (map - l1 l2))))`
A major roadblock in the development of an HIV vaccine is the need to develop vaccine regimens that will induce antibodies that bind to conserved regions of the HIV envelope and neutralize many different virus quasispecies. One such envelope target is at the region closest to the membrane, the glycoprotein (gp) 41 membrane proximal external region (MPER). Previous work has demonstrated that antibodies that target this region bind both to the gp41 polypeptide and to the adjacent viral membrane. However, what has been missing is a view of what the MPER-neutralizing epitopes may look like in the context of a trimeric orientation with lipids. We have constructed an MPER trimer associated with lipids and solved the trimer structure by NMR spectroscopy.

Abstract from Reardon et al. *PNAS* (2014)
Map/Reduce a lá Google

- They say “emit” for “return”
- Maybe this makes sense for a cloud of small processors....
- The term “unique key” tells you we have to make things unique...
- ...we must remove-duplicates, filter, or hash.
Map/Reduce a lá Google

- **map(key, val)** is run on each item in set
  - emits new-key / new-val pairs
    (map (lambda (word) (list word 1)) doc)
- **reduce(key, vals)** is run for each unique key emitted by map()
  - emits final output
    (reduce ...
      (map (lambda (word) (list word 1))
           document) →
      ((A 1) (major 1) (roadblock 1) (in 1) (the 1) (development 1) (of 1) (an 1) (HIV 1) (vaccine 1) (is 1) (the 1) (need 1) (to 1) (develop 1) (vaccine 1) (regimens 1) (that 1) (will 1) (induce 1) ...))
(define add-wc-pairs
  (lambda (wcp1 wcp2)
    (list (first wcp1) ;; must have: (equal? (first wcp1) (first wcp2))
      (+ (second wcp1) (second wcp2)))))

;; count a particular word in a document
(define count-word
  (lambda (entry doc)
    (reduce add-wc-pairs
      (list entry 0)
      (filter (lambda (WC-pair)
        (equal? (first WC-pair) entry))
      (map (lambda (word) (list word 1)) doc) ))))
(define add-wc-pairs
  (lambda (wcp1 wcp2)
    (list (first wcp1) ;; must have: (equal? (first wcp1) (first wcp2))
      (+ (second wcp1) (second wcp2))))))

;; count a particular word in a document
(define count-word
  (lambda (entry doc)
    (reduce add-wc-pairs
      (list entry 0)
      (filter (lambda (WC-pair)
        (equal? (first WC-pair) entry))
        (map (lambda (word) (list word 1)) doc) ))))

(count-word 'vaccine document)  \( \rightarrow \) (vaccine 2)
(count-word 'HIV document) \( \rightarrow \) (HIV 2)
(count-word 'the document) \( \rightarrow \) (the 10)

(map (lambda (entry) (count-word entry document))
  '(vaccine HIV the)) \( \rightarrow \) ((vaccine 2) (HIV 2) (the 10))
(define add-wc-pairs
 (lambda (wcp1 wcp2)
   (list (first wcp1) ;; must have: (equal? (first wcp1) (first wcp2))
     (+ (second wcp1) (second wcp2))))))

;; count a particular word in a document
(define count-word
 (lambda (entry doc)
   (reduce add-wc-pairs
     (list entry 0)
     (filter (lambda (WC-pair) ;; How, using Map/Reduce?
       (equal? (first WC-pair) entry))
     (map (lambda (word) (list word 1)) doc))))))

(count-word 'vaccine document) \rightarrow (vaccine 2)
(count-word 'HIV document) \rightarrow (HIV 2)
(count-word 'the document) \rightarrow (the 10)

(map (lambda (entry) (count-word entry document))
  '(vaccine HIV the)) \rightarrow
  ((vaccine 2) (HIV 2) (the 10))
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L)
               (cond ((pred? x) (cons x L))
                     (else L)))
            '()
            lst)))

(filter odd? '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16)) ➔
(1 3 5 7 9 11 13 15)
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L)
               (cond ((pred? x) (cons x L))
                     (else L)))
            ()
            lst)))

(filter odd? '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16)) ➞
(1 3 5 7 9 11 13 15)

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L)
                (cond ((member x L) L)
                      (else (cons x L)))
            ()
            lst)))

(remdup '(1 2 3 4 5 6 7 7 7 2 1 3 4 1 5 7 9 11 12 1 2)) ➞
(6 3 4 5 7 9 11 12 1 2)
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L) …

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) …

filter and remdup are wrappers around calls to reduce
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L) …

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) …

;; count a particular word in a document
(define count-word
  (lambda (entry doc)
    (reduce add-wc-pairs
      (list entry 0)
      (filter (lambda (WC-pair)
                (equal? (first WC-pair) entry))
        (map (lambda (word) (list word 1)) doc) ))))

filter and remdup are wrappers around calls to reduce
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L)  …

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) …

;; count a particular word in a document
(define count-word
  (lambda (entry doc)
    (reduce add-wc-pairs
      (list entry 0)
      (filter (lambda (WC-pair)
        (equal? (first WC-pair) entry))
      (map (lambda (word) (list word 1)) doc) ))))

;; Q. How do we do this for all words in a document?

filter and remdup are wrappers around calls to reduce
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L) …

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) …

;; count a particular word in a document
(define count-word
  (lambda (entry doc)
    (reduce add-wc-pairs
      (list entry 0)
      (filter (lambda (WC-pair)
        (equal? (first WC-pair) entry))
        (map (lambda (word) (list word 1)) doc) ))))

;; Q. How do we do this for all words in a document?
;; A. Use Map, of course!

filter and remdup are wrappers around calls to reduce
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L) …

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) …

;; count a particular word in a document
(define count-word
  (lambda (entry doc)
    (reduce add-wc-pairs
      (list entry 0)
      (filter (lambda (WC-pair)
        (equal? (first WC-pair) entry))
      (map (lambda (word) (list word 1)) doc) )

(map (lambda (entry) (count-word entry document)))
 'vaccine HIV the) →
  ((vaccine 2) (HIV 2) (the 10)))

filter and remdup are wrappers around calls to reduce
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L)  …

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) …

; count all the words in a document:
(define WC-doc
  (lambda (doc)
    (map (lambda (entry)
       (count-word entry doc) )
    (remdup doc)))))

filter and remdup are wrappers around calls to reduce
A major roadblock in the development of an HIV vaccine is the need to develop vaccine regimens that will induce antibodies that bind to conserved regions of the HIV envelope and neutralize many different virus quasispecies. One such envelope target is at the region closest to the membrane, the glycoprotein (gp) 41 membrane proximal external region (MPER). Previous work has demonstrated that antibodies that target this region bind both to the gp41 polypeptide and to the adjacent viral membrane. However, what has been missing is a view of what the MPER-neutralizing epitopes may look like in the context of a trimeric orientation with lipids. We have constructed an MPER trimer associated with lipids and solved the trimer structure by NMR spectroscopy.

(wc-doc document) ➔

(thesize 10) (to 5) (thesize 5) (that 4) (of 4) (thesize 4) (region 3) (thesize 3) (membrane 3) (is 3) (and 3) (vaccine 2) (HIV 2) (thesize 2) (envelope 2) (thesize 2) (antibodies 2) (thesize 2) (target 2) (thesize 2) (bind 2) (thesize 2) (has 2) (what 2) (thesize 2) (in 2) (a 2) (thesize 2) (an 2) ...
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L) …

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) …

; count all the words in a document:
(define WC-doc
  (lambda (doc)
    (map (lambda (entry)
      (count-word entry doc) )
    (remdup doc)))))

;; Naming count-word is nice, but let’s see the parallel pipeline
;; of the functional program

filter and remdup are wrappers around calls to reduce
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L) ... )

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) ... )

;; count a particular word in a document
(define count-word
  (lambda (entry doc)
    (reduce add-wc-pairs
      (list entry 0)
      (filter (lambda (WC-pair)
        (equal? (first WC-pair) entry))
      (map (lambda (word) (list word 1)) doc) )))

;; Q. How do we do this for all words in a document?
;; A. Use Map, of course!

(filter and remdup are wrappers around calls to reduce)
(define filter
  (lambda (pred? lst)
    (reduce (lambda (x L) ...)
          (pred? lst)
    )
  )
)

(define remdup ;; remove duplicates
  (lambda (lst)
    (reduce (lambda (x L) ...)
            (lst)
    )
  )
)

; count all the words in a document:
(define WC-doc
  (lambda (doc)
    (map (lambda (entry)
          (reduce add-wc-pairs
                   (list entry 0)
          )
          (filter (lambda (WC-pair)
                   (equal? (first WC-pair) entry))
          )
          (map (lambda (word) (list word 1)) doc))
          (remdup doc)))
)

filter and remdup are wrappers around calls to reduce
\( \lambda \)-calculus
Scheme
Substitution Model
Functional Programming
Higher-Order Functions
Induction
Map/Reduce a lá Google

- **map(key, val)** is run on each item in set
  - emits new-key / new-val pairs
    - \( \text{map } (\text{lambda } (\text{word}) (\text{list } \text{word } 1)) \text{ } \text{doc} \)
- **reduce(key, vals)** is run for each unique key emitted by map()
  - emits final output
    - \( \text{reduce } \ldots \)

\[
\text{(map } (\text{lambda } (\text{word}) (\text{list } \text{word } 1)) \text{ } \text{document}) \Rightarrow \\
((\text{A } 1) (\text{major } 1) (\text{roadblock } 1) (\text{in } 1) (\text{the } 1) (\text{development } 1) (\text{of } 1) (\text{an } 1) (\text{HIV } 1) (\text{vaccine } 1) (\text{is } 1) (\text{the } 1) (\text{need } 1) (\text{to } 1) (\text{develop } 1) (\text{vaccine } 1) (\text{regimens } 1) (\text{that } 1) (\text{will } 1) (\text{induce } 1) \ldots)\]
count words in docs

- Input consists of (url, contents) pairs

- map(key=url, val=contents):
  - For each word $w$ in contents, emit $(w, "1")$

- reduce(key=word, values=uniq_counts):
  - Sum all "1"s in values list
  - Emit result "(word, sum)"
map(key=url, val=contents):
   For each word w in contents, emit (w, “1”)
reduce(key=word, values=uniq_counts):
   Sum all “1”s in values list
   Emit result “(word, sum)”

see bob throw
see spot run

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>see</td>
<td>1</td>
</tr>
<tr>
<td>bob</td>
<td>1</td>
</tr>
<tr>
<td>run</td>
<td>1</td>
</tr>
<tr>
<td>see</td>
<td>1</td>
</tr>
<tr>
<td>spot</td>
<td>1</td>
</tr>
<tr>
<td>throw</td>
<td>1</td>
</tr>
</tbody>
</table>
function map(String name, String document):
    // name: document name
    // document: document contents
    for each word w in document:
        emit (w, 1)

function reduce(String word, Iterator partialCounts):
    // word: a word
    // partialCounts: a list of aggregated partial counts
    sum = 0
    for each pc in partialCounts:
        sum += ParseInt(pc)
    emit (word, sum)

(map (lambda (word) (list word 1)) document) ➞

((A 1) (major 1) (roadblock 1) (in 1) (the 1) (development 1) (of 1) (an 1) (HIV 1) (vaccine 1) (is 1) (the 1) (need 1) (to 1) (develop 1) (vaccine 1) (regimens 1) (that 1) (will 1) (induce 1) ...)
Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, “1”)

- reduce(key=line, values=uniq_counts):
  - Don’t do anything; just emit line
Reverse Web-Link Graph

Find all pages that link to a certain page

- **Map**
  - For each URL linking to target, ...
  - Output $<\text{target, source}>$ pairs
- **Reduce**
  - Concatenate list of all source URLs
  - Outputs: $<\text{target, list (source)}> \text{ pairs}$
Inverted Index

Def. An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents.

The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database.

It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines.
Inverted Index

- Map

- Reduce
Model is Widely Applicable
MapReduce Programs In Google Source Tree

Example uses:
distributed grep
term-vector / host
document clustering

distributed sort
web access log stats
machine learning

web link-graph reversal
inverted index construction
statistical machine translation
Implementation Overview

Typical cluster:

- 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
- Limited bisection bandwidth
- Storage is on local IDE disks
- GFS: distributed file system manages data (SOSP'03)
- Job scheduling system: jobs made up of tasks, scheduler assigns tasks to machines
Execution

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
  2. After all map()s are complete, consolidate all emitted values for each unique emitted key
  3. Now partition space of output map keys, and run reduce() in parallel

- If map() or reduce() fails, reexecute!
1. Client submits "grep" job, indicating code and input files.
2. JobTracker breaks input file into $k$ chunks, (in this case 6). Assigns work to tasktrackers.
3. After map(), tasktrackers exchange map-output to build reduce() keyspace.
4. JobTracker breaks reduce() keyspace into $m$ chunks (in this case 6). Assigns work.
5. reduce() output may go to NDFS.
Execution

Input

Intermediate

Group by Key

Grouped

Output
Parallel Execution

Map Task 1

Map Task 2

Map Task 3

Partitioning Function

Sort and Group

Reduce Task 1

Reduce Task 2
Task Granularity & Pipelining

- **Fine granularity tasks**: map tasks >> machines
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing
- Often use 200,000 map & 5000 reduce tasks
- Running on 2000 machines

<table>
<thead>
<tr>
<th>Process</th>
<th>Time ------------------- &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Program</td>
<td>MapReduce() ... wait ...</td>
</tr>
<tr>
<td>Master</td>
<td>Assign tasks to worker machines...</td>
</tr>
<tr>
<td>Worker 1</td>
<td>Map 1, Map 3</td>
</tr>
<tr>
<td>Worker 2</td>
<td>Map 2</td>
</tr>
<tr>
<td>Worker 3</td>
<td>Read 1.1, Read 1.3, Read 1.2, Reduce 1</td>
</tr>
<tr>
<td>Worker 4</td>
<td>Read 2.1, Read 2.2, Read 2.3, Reduce 2</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>0</td>
<td>323</td>
<td>878934.6</td>
<td>1314.4</td>
<td>717.0</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>323</td>
<td>717.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>72.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>145825686</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>506631</td>
</tr>
</tbody>
</table>
| dups-in-index-
merge | 0      |
| mr-operator-
calls | 508192 |
| mr-operator-
outputs | 506631 |
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>1857</td>
<td>1707</td>
<td>878934.6</td>
<td>191995.8</td>
<td>113936.6</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>113936.6</td>
<td>57113.7</td>
<td>57113.7</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>57113.7</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>699.1</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>349.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5004411944</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17290135</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17331371</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17290135</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>5354</td>
<td>1707</td>
<td>878934.6</td>
<td>406020.1</td>
<td>241058.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>241058.2</td>
<td>196362.5</td>
<td>196362.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>196362.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>704.4</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>371.9</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5000364228</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17300709</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17342493</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17300709</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>8841</td>
<td>1707</td>
<td>878934.6</td>
<td>621608.5</td>
<td>369459.8</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>369459.8</td>
<td>326986.8</td>
<td>326986.8</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>326986.8</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>706.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>419.2</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>4982870667</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17229926</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17272056</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17229926</td>
</tr>
</tbody>
</table>
**MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03**

**Started:** Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>195</td>
<td>305</td>
<td>523499.2</td>
<td>523389.6</td>
<td>523389.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>195</td>
<td>523389.6</td>
<td>2685.2</td>
<td>2742.6</td>
</tr>
</tbody>
</table>

**Counters**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.3</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>45.7</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>2313178</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>7936</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mrr-merge-calls</td>
<td>1954105</td>
</tr>
<tr>
<td>mrr-merge-outputs</td>
<td>1954105</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>133837.8</td>
<td>136929.6</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.1</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1238.8</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0.10</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51738599</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51738599</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>263283.3</td>
<td>269351.2</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1225.1</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-indexed</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51842100</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51842100</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>390447.6</td>
<td>399457.2</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1222.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51640600</td>
</tr>
<tr>
<td>mr-merge-outsuts</td>
<td>51640600</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>520468.6</td>
<td>520468.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>406</td>
<td>94</td>
<td>520468.6</td>
<td>512265.2</td>
<td>514373.3</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>849.5</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0 100</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-indexed</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>35083350</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>35083350</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>519781.8</td>
<td>519781.8</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>498</td>
<td>2</td>
<td>519781.8</td>
<td>519394.7</td>
<td>519440.7</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>9.4</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-indexmerge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>394792</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>394792</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>519774.3</td>
<td>519774.3</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>499</td>
<td>1</td>
<td>519774.3</td>
<td>519735.2</td>
<td>519764.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1.9</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0 1050</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>73442</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>73442</td>
</tr>
</tbody>
</table>
Fault Tolerance / Workers

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why????
- Re-execute in progress reduce tasks
- Task completion committed through master

Robust: lost 1600/1800 machines once → finished ok

Semantics in presence of failures: see paper
Master Failure

- Could handle, ...?
- But don't yet
  - (master failure unlikely)
Refinement: Redundant Execution

Slow workers significantly delay completion time

- Other jobs consuming resources on machine
- Bad disks w/ soft errors transfer data slowly
- Weird things: processor caches disabled (!!)

Solution: Near end of phase, spawn backup tasks

- Whichever one finishes first "wins"

Dramatically shortens job completion time
Refinement: Locality Optimization

- **Master scheduling policy:**
  - Asks GFS for locations of replicas of input file blocks
  - Map tasks typically split into 64MB (GFS block size)
  - Map tasks scheduled so GFS input block replica are on same machine or same rack

- **Effect**
  - Thousands of machines read input at local disk speed
    - Without this, rack switches limit read rate
Refinement
Skipping Bad Records

- Map/Reduce functions sometimes fail for particular inputs
  - Best solution is to debug & fix
    - Not always possible ~ third-party source libraries
  - On segmentation fault:
    - Send UDP packet to master from signal handler
    - Include sequence number of record being processed
  - If master sees two failures for same record:
    - Next worker is told to skip the record
Other Refinements

- **Sorting guarantees**
  - within each reduce partition
- **Compression of intermediate data**
- **Combiner**
  - Useful for saving network bandwidth
- **Local execution for debugging/testing**
- **User-defined counters**
Performance

Tests run on cluster of 1800 machines:

- 4 GB of memory
- Dual-processor 2 GHz Xeons with Hyperthreading
- Dual 160 GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100 Gbps

Two benchmarks:

- **MR_GrepScan**: 1010 100-byte records to extract records matching a rare pattern (92K matching records)
- **MR_SortSort**: 1010 100-byte records (modeled after TeraSort benchmark)
Locality optimization helps:

- 1800 machines read 1 TB at peak ~31 GB/s
- W/out this, rack switches would limit to 10 GB/s

Startup overhead is significant for short jobs
- Backup tasks reduce job completion time a lot!
- System deals well with failures
Experience

Rewrote Google’s production indexing System using MapReduce

- Set of 10, 14, 17, 21, 24 MapReduce operations
- New code is simpler, easier to understand
  - 3800 lines of code → 700
- MapReduce handles failures, slow machines
- Easy to make indexing faster
  - add more machines
# Usage in Aug 2004

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>29,423</td>
</tr>
<tr>
<td>Average job completion time</td>
<td>634 secs</td>
</tr>
<tr>
<td>Machine days used</td>
<td>79,186 days</td>
</tr>
<tr>
<td>Input data read</td>
<td>3,288 TB</td>
</tr>
<tr>
<td>Intermediate data produced</td>
<td>758 TB</td>
</tr>
<tr>
<td>Output data written</td>
<td>193 TB</td>
</tr>
<tr>
<td>Average worker machines per job</td>
<td>157</td>
</tr>
<tr>
<td>Average worker deaths per job</td>
<td>1.2</td>
</tr>
<tr>
<td>Average map tasks per job</td>
<td>3,351</td>
</tr>
<tr>
<td>Average reduce tasks per job</td>
<td>55</td>
</tr>
<tr>
<td>Unique map implementations</td>
<td>395</td>
</tr>
<tr>
<td>Unique reduce implementations</td>
<td>269</td>
</tr>
<tr>
<td>Unique map/reduce combinations</td>
<td>426</td>
</tr>
</tbody>
</table>
Conclusions

- MapReduce proven to be useful abstraction
- Greatly simplifies large-scale computations
- Based on Scheme Functions with a well-defined semantics.
- Substitution model holds (functional program)
- Fun to use:
  - focus on problem,
  - let library deal w/ messy details