CS420 – Lecture 23

Marc Snir

Fall 2018

MapReduce

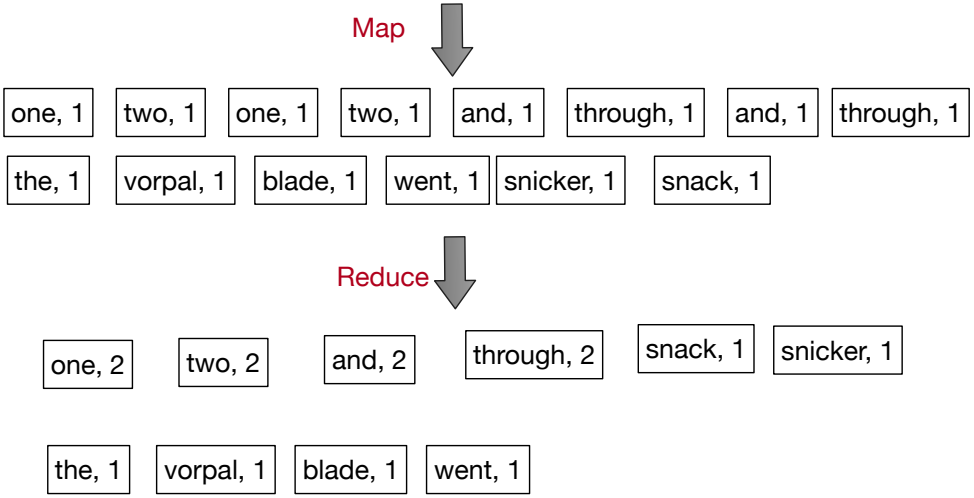
- Data is a set of pairs
- A computation is defined by a *map* function and a *reduce* function

  map $map(k, v) \rightarrow < k', v' >^*$ All elements are scanned one by one and new <key,value> pairs are generated from each element

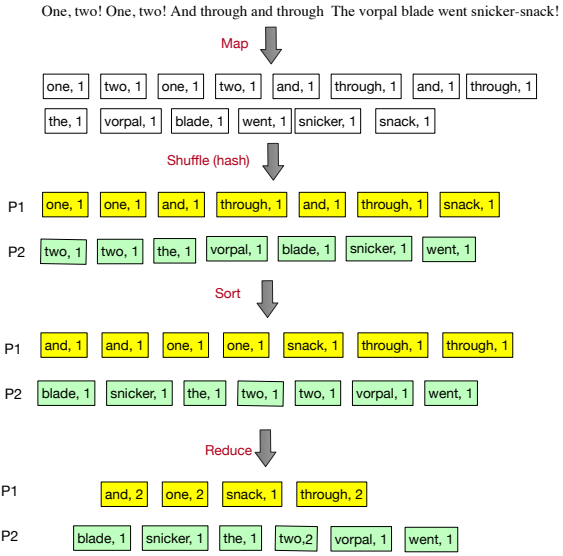
  reduce $reduce(k', < v' >^*) \rightarrow < k', v'' >$ All the elements with the same key values are gathered and a reduction operation is applied to their values.

"Canonical Example": count number of occurences of each word in a text

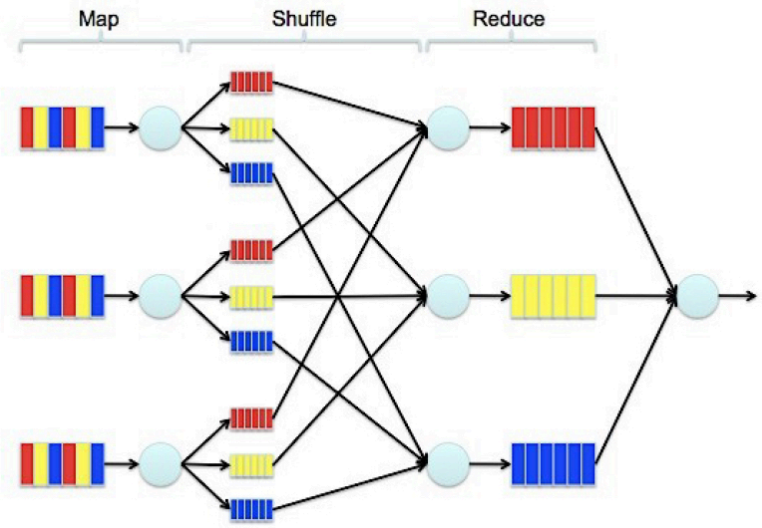
One, two! One, two! And through and through The vorpal blade went snicker-snack!



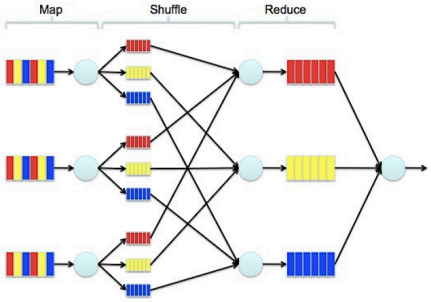
Under the Cover



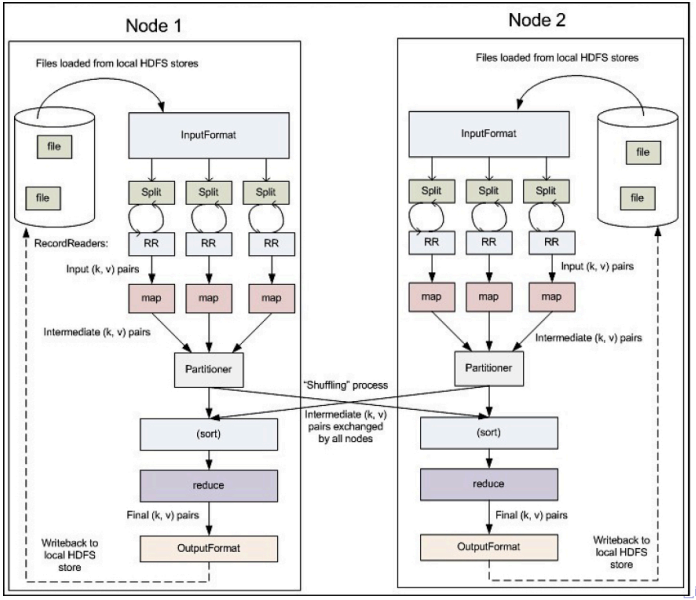
Under the Cover



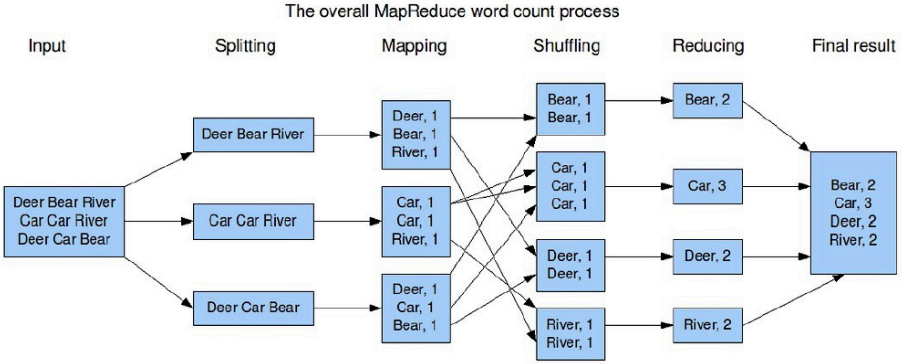
- Map execution partitioned – executed where data is (HDFS blocks)
- Data is sorted locally, by keys
- tuples are sent to reducer nodes, using (default) partition mapping
- Each reducer executes locally its reduction
- Outputs are combined
- Computation is orchestrated by a central manager
- Manager may resubmit task if it failed or is slow



Another view



Word Count, again



"Split" is nominal: The HDFS file is already split

Pseudo-code

```
Map(String input_key, String input_value) {
// Input key: document name
// Input value: document content
for each word w in input_values
EmitIntermediate(w, ``1'');
}

Reduce(String key, Iterator intermediate_values) {
int result=0;
for each v in intermediate_values
result += ParseInt(v);
Emit (AsString(result));
}
```

Simple real code (Python): mapper.py

```
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
# remove leading and trailing whitespace
line = line.strip()
# split the line into words
words = line.split()

for word in words:
# write the results to STDOUT (standard output);
#each line contains tab-delimited key value
# will be input to reducer.py
print '%s\t%s' % (word, 1)
```

reducer.py

```
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
# remove leading and trailing whitespace
line = line.strip()

# parse the input we got from mapper.py
word, count = line.split('\t', 1)
```

reducer.py

```
# convert count (currently a string) to int
try:
count = int(count)
except ValueError:
# count was not a number, so silently
# ignore/discard this line
continue
```

```
# this IF-switch only works because Hadoop sorts map output
# by key (here: word) before it is passed to the reducer
if current_word == word:
current_count += count
else:
if current_word:
# write result to STDOUT
print '%s\t%s' % (current_word, current_count)
current_count = count
current_word = word

# do not forget to output the last word if needed!
if current_word == word:
print '%s\t%s' % (current_word, current_count)
```

Simple test (sequential execution)

```
echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py
foo     1
foo     1
quux    1
labs    1
foo     1
bar     1
quux    1
```

```
echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py | sort -k1,1 | \
/home/hduser/reducer.py
bar     1
foo     3
labs    1
quux    2
```

Execution using the Hadoop framework

Code run by passing to Hadoop the mapper and reducer functions; Hadoop Streaming API is needed as mapper and reducer communicate via STDOUT and STDIN

```
bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar \
-file /home/hduser/mapper.py    -mapper /home/hduser/mapper.py \
-file /home/hduser/reducer.py   -reducer /home/hduser/reducer.py \
-input /snir/text-dir/* -output /snir/counts-dir/*
```

Will get as output one file per reducer.
Number of reducers is decided by Hadoop, but user can request some number

```
bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar -D \
mapred.reduce.tasks=16 ...
```

Previous example used asci file for communication between mapper and reducer. Not most efficient – in particular because need to generate entire file before reducers start working. Instead can leverage Python's iterators and generators.

Iterator: produces a sequence of values

Generator: an iterator that yields control (with a *yield* call). It is invoked again after previously produced values were consumed. This allows the execution of values' consumer to be interleaved with the execution of the producer.

Pseudocode, again

.

```
Map(String input_key, String input_value) {
for each word w in input_values
EmitIntermediate(w, ``1'');
}

Reduce(String key, Iterator intermediate_values) {
int result=0;
for each v in intermediate_values
result += ParseInt(v);
Emit (AsString(result));
}
```

The *reduce* function is passed an *iterator* that can be used to obtain successive intermediate values

Haddop run time can decide how to schedule value producers (mapper) and value consumers (reducer)

New mapper.py

```
#!/usr/bin/env python

import sys

def read_input(file):
for line in file:
# split the line into words
yield line.split()

def main(separator='\t'):
# input comes from STDIN
data = read_input(sys.stdin)
```

New mapper.py

```
for words in data:
# write the results to STDOUT;
# pairs are tab-delimited;
for word in words:
print '%s%s%d' % (word, separator, 1)

if __name__ == "__main__":
main()
```

new reducer.py

```
#!/usr/bin/env python
from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file, separator='\t'):
for line in file:
yield line.rstrip().split(separator, 1)

def main(separator='\t'):
# input comes from STDIN (standard input)
data = read_mapper_output(sys.stdin, separator=separator)
```

New mapper.py

```
# groupby groups multiple word-count pairs by word,
# and creates an iterator that returns consecutive keys and their group:
#   current_word - string containing a word (the key)
#   group - iterator yielding all items
for current_word, group in groupby(data, itemgetter(0)):
try:
total_count = sum(int(count) for current_word, count in group)
print "%s%s%d" % (current_word, separator, total_count)
except ValueError:
# count was not a number, so silently discard this item
pass

if __name__ == "__main__":
main()
```

Same with C++

```
#include <algorithm>
#include <limits>
#include <string>
#include  "stdint.h"
#include "hadoop/Pipes.hh"
#include "hadoop/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"
```

```
using namespace std;

class WordCountMapper : public HadoopPipes::Mapper {
public:
// constructor: does nothing
WordCountMapper( HadoopPipes::TaskContext& context ) {
}

// map function: receives a line, outputs (word,"1")
// to reducer.
void map( HadoopPipes::MapContext& context ) {
//--- get line of text ---
string line = context.getInputValue();
```

New mapper.py

```
//--- split it into words ---
vector< string > words =
HadoopUtils::splitString( line, "␣" );

//--- emit each word tuple (word, "1" ) ---
for ( unsigned int i=0; i < words.size(); i++ ) {
context.emit( words[i], HadoopUtils::toString( 1 ) );
}
}
};
```

```
class WordCountReducer : public HadoopPipes::Reducer {
public:
// constructor: does nothing
WordCountReducer(HadoopPipes::TaskContext& context) {
}

// reduce function
void reduce( HadoopPipes::ReduceContext& context ) {
int count = 0;

//--- get all tuples with the same key, and count their numbers ---
while ( context.nextValue() ) {
count += HadoopUtils::toInt( context.getInputValue() );
}

//--- emit (word, count) ---
context.emit(context.getInputKey(), HadoopUtils::toString( count ));
}
};

int main(int argc, char *argv[]) {
return HadoopPipes::runTask(HadoopPipes::TemplateFactory<
```

Matlab and MapReduce

Can used MapReduce on a Hadoop cluster from Matlab, in order to analyze large datasets. Need

- Matlab storage for large datasets
- Matlab interface to MapReduce

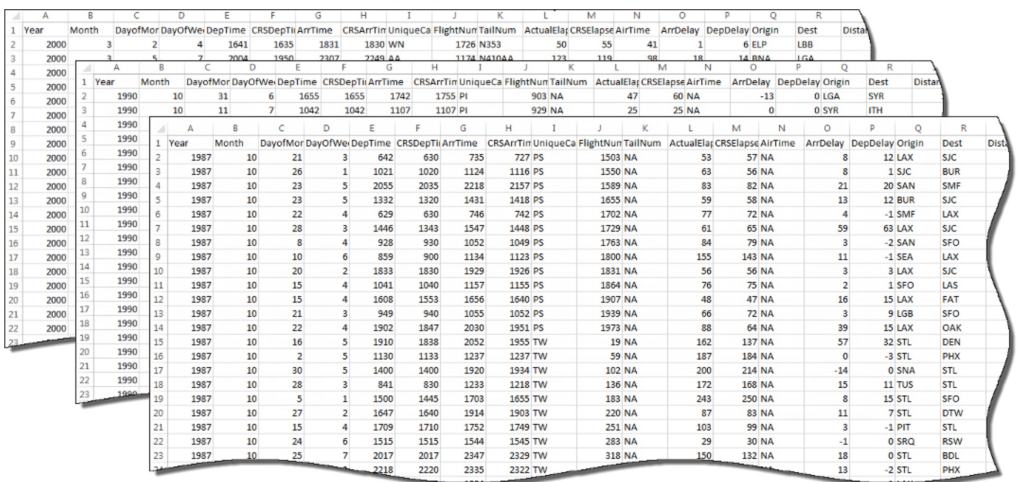
Matlab Datastore

Repository for large data objects (can be built atop HDFS)

Text files containing column-oriented data, including CSV files.	TabularTextDatastore
Image files, including formats that are supported by imread such as JPEG and PNG.	ImageDatastore
Spreadsheet files with a supported Excel® format such as .xlsx.	SpreadsheetDatastore
Key-value pair data that are inputs to or outputs of mapreduce.	KeyValueDatastore
Custom file formats. Requires a provided function for reading data. ...	FileDatastore ...

Matlab also provides *tall arrays* for tabular information – Can be directly accessed by Matlab stats functions.

Analyzing airline delays



Year	Month	DayofMon	DayOfWee	DepTime	CRSDepTi	ArrTime	CRSArrTim	UniqueCa	FlightNum	TailNum	ActualEla	CRSElapse	AirTime	ArrDelay	DepDelay	Origin	Dest	Dista
2000	3	2	4	1641	1635	1831	1830	WN	1726	N353	50	55	41	1	6	ELP	LBB	
2000	3	5	7	2004	1950	2307	2249	AA	1174	N410AA	123	119	98	18	14	BNA	LGA	

Year	Month	DayofMon	DayOfWee	DepTime	CRSDepTi	ArrTime	CRSArrTim	UniqueCa	FlightNum	TailNum	ActualEla	CRSElapse	AirTime	ArrDelay	DepDelay	Origin	Dest	Dista
1990	10	31	6	1655	1655	1742	1755	PI	903	NA	47	60	NA	-13	0	LGA	SYR	
1990	10	11	7	1042	1042	1107	1107	PI	929	NA	25	25	NA	0	0	SYR	ITH	

Year	Month	DayofMon	DayOfWee	DepTime	CRSDepTi	ArrTime	CRSArrTim	UniqueCa	FlightNum	TailNum	ActualEla	CRSElapse	AirTime	ArrDelay	DepDelay	Origin	Dest	Dist
1987	10	21	3	642	630	735	727	PS	1503	NA	53	57	NA	8	12	LAX	SJC	
1987	10	26	1	1021	1020	1124	1116	PS	1550	NA	63	56	NA	8	1	SJC	BUR	
1987	10	23	5	2055	2035	2218	2157	PS	1589	NA	83	82	NA	21	20	SAN	SMF	
1987	10	23	5	1332	1320	1431	1418	PS	1655	NA	59	58	NA	13	12	BUR	SJC	
1987	10	22	4	629	630	746	742	PS	1702	NA	77	72	NA	4	-1	SMF	LAX	
1987	10	28	3	1446	1343	1547	1448	PS	1729	NA	61	65	NA	59	63	LAX	SJC	
1987	10	8	4	928	930	1052	1049	PS	1763	NA	84	79	NA	3	-2	SAN	SFO	
1987	10	10	6	859	900	1134	1123	PS	1800	NA	155	143	NA	11	-1	SEA	LAX	
1987	10	20	2	1833	1830	1929	1926	PS	1831	NA	56	56	NA	3	3	LAX	SJC	
1987	10	15	4	1041	1040	1157	1155	PS	1864	NA	76	75	NA	2	1	SFO	LAS	
1987	10	15	4	1608	1553	1656	1640	PS	1907	NA	48	47	NA	16	15	LAX	FAT	
1987	10	21	3	949	940	1055	1052	PS	1939	NA	66	72	NA	3	9	LGB	SFO	
1987	10	22	4	1902	1847	2030	1951	PS	1973	NA	88	64	NA	39	15	LAX	OAK	
1987	10	16	5	1910	1838	2052	1955	TW	19	NA	162	137	NA	57	32	STL	DEN	
1987	10	2	5	1130	1133	1237	1237	TW	59	NA	187	184	NA	0	-3	STL	PHX	
1987	10	30	5	1400	1400	1920	1934	TW	102	NA	200	214	NA	-14	0	SNA	STL	
1987	10	28	3	841	830	1233	1218	TW	136	NA	172	168	NA	15	11	TUS	STL	
1987	10	5	1	1500	1445	1703	1655	TW	183	NA	243	250	NA	8	15	STL	SFO	
1987	10	27	2	1647	1640	1914	1903	TW	220	NA	87	83	NA	11	7	STL	DTW	
1987	10	15	4	1709	1710	1752	1749	TW	251	NA	103	99	NA	3	-1	PIT	STL	
1987	10	24	6	1515	1515	1544	1545	TW	283	NA	29	30	NA	-1	0	SRQ	RSW	
1987	10	25	7	2017	2017	2347	2329	TW	318	NA	150	132	NA	18	0	STL	BDL	
				2218	2220	2335	2322	TW						13	-2	STL	PHX	

Ingest CSV files into datastore

```
ds = datastore('flights.csv','TreatAsMissing','NA');
```

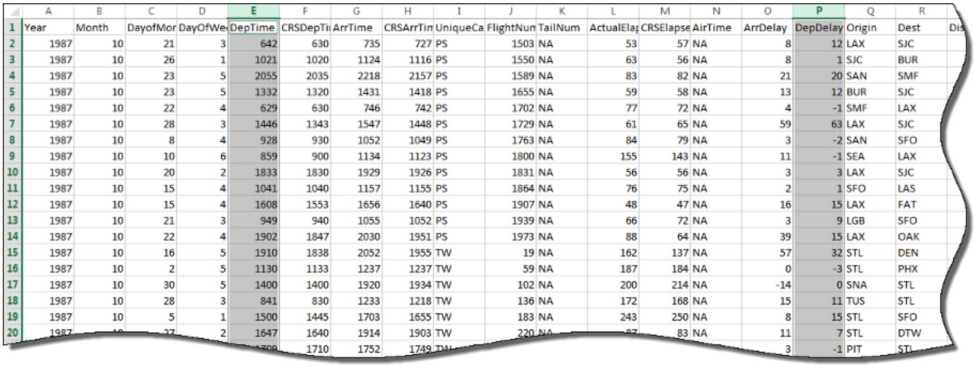
Creates tabular datastore with fields as named in the title row; a field containing `'NA'` is assumed to represent a missing value. Matlab makes 'best guess' on the type of cell entries.

```
ds.MissingValue = 0;
```

Replace missing values with 0 (by default, if field is numeric, Matlab would replace with NaN). Can then select and preview part of the datastore

```
ds.SelectedVariableNames = {'DepTime','DepDelay'};
preview(ds)
```

preview of read only these variables:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Year	Month	DayofMor	DayOfWe	DepTime	CRSDepTi	ArrTime	CRSArrTin	UniqueCa	FlightNun	TailNum	ActualEla	CRSElapse	AirTime	ArrDelay	DepDelay	Origin	Dest
2	1987	10	21	3	642	630	735	727	PS	1503	NA	53	57	NA	8	12	LAX	SJC
3	1987	10	26	1	1021	1020	1124	1116	PS	1550	NA	63	56	NA	8	1	SJC	BUR
4	1987	10	23	5	2055	2035	2218	2157	PS	1589	NA	83	82	NA	21	20	SAN	SMF
5	1987	10	23	5	1332	1320	1431	1418	PS	1655	NA	59	58	NA	13	12	BUR	SJC
6	1987	10	22	4	629	630	746	742	PS	1702	NA	77	72	NA	4	-1	SMF	LAX
7	1987	10	28	3	1446	1343	1547	1448	PS	1729	NA	61	65	NA	59	63	LAX	SJC
8	1987	10	8	4	928	930	1052	1049	PS	1763	NA	84	79	NA	3	-2	SAN	SFO
9	1987	10	10	6	859	900	1134	1123	PS	1800	NA	155	143	NA	11	-1	SEA	LAX
10	1987	10	20	2	1833	1830	1929	1926	PS	1831	NA	56	56	NA	3	3	LAX	SJC
11	1987	10	15	4	1041	1040	1157	1155	PS	1864	NA	76	75	NA	2	1	SFO	LAS
12	1987	10	15	4	1608	1553	1656	1640	PS	1907	NA	48	47	NA	16	15	LAX	FAT
13	1987	10	21	3	949	940	1055	1052	PS	1939	NA	66	72	NA	3	9	LGB	SFO
14	1987	10	22	4	1902	1847	2030	1951	PS	1973	NA	88	64	NA	39	15	LAX	OAK
15	1987	10	16	5	1910	1838	2052	1955	TW	19	NA	162	137	NA	57	32	STL	DEN
16	1987	10	2	5	1130	1133	1237	1237	TW	59	NA	187	184	NA	0	-3	STL	PHX
17	1987	10	30	5	1400	1400	1920	1934	TW	102	NA	200	214	NA	-14	0	SNA	STL
18	1987	10	28	3	841	830	1233	1218	TW	136	NA	172	168	NA	15	11	TUS	STL
19	1987	10	5	1	1500	1445	1703	1655	TW	183	NA	243	250	NA	8	15	STL	SFO
20	1987	10	27	2	1647	1640	1914	1903	TW	220	NA	87	83	NA	11	7	STL	DTW
					1708		1710	1752	1749	TW					3	-1	PIT	STL

Answer

```
ans =
DepTime    DepDelay
-------    --------
642        12
1021         1
2055        20
1332        12
629        -1
1446        63
928        -2
859        -1
```

Datastore organized to be read in big chunks.
By default, a chunk is 20,000 rows; the default can be changed.

```
ds.ReadSize = 15000;
```

User can also control other properties of table: names of columns, type of entries in each column, etc.

Simple processing of tabular datastore

```
reset(ds)
X = [];
while hasdata(ds)
T = read(ds);
X(end+1) = max(T.DepDelay);
end
maxDelay = max(X)
```

While loop iterates over data chunks – possibly over multiple files
Execution is sequential

Compute average delay

```
sums = [];
counts = [];
while hasdata(ds)
T = read(ds);

sums(end+1) = sum(T.ArrDelay);
counts(end+1) = length(T.ArrDelay);
end
avgArrivalDelay = sum(sums)/sum(counts)
```

Invoking MapReduce from Matlab

```
outds = mapreduce(ds,mapfun,reducefun)
```

- outds: Output datastore (`KeyValueDatastore` object)
- ds: Input datastore (of any type)
- mapfun: Mapper
- reducefun: Reducer

```
outds = mapreduce(ds,mapfun,reducefun,mr)
```

can be used to control execution environment

Example: Count # flights by airline

Same problem as word frequency count

```
ds = tabularTextDatastore('flights.csv', 'TreatAsMissing', 'NA');
ds.SelectedVariableNames = 'UniqueCarrier';
ds.SelectedFormats = '%C';
```

Create a tabular datastore from spreadsheet; treat `'NA'` as indication of missing field.
Select column with title `'UniqueCarrier'`.
The cells in that columns contain characters.

Mapper

```
function countMapper(data, info, intermKV)
% Counts unique airline carrier names in each chunk.
a = data.UniqueCarrier;
keys = categories(a);
c = num2cell(countcats(a));
addmulti(intermKV, keys, c)
end
```

categories(): Returns vector containing the categories (distinct entries) in a.

countcats(): Returns number of elements in each category

numtocell(): Converts numerical array to cell array

addmulti(): Adds multiple tuples to key-value store

Reducer

```
function countReducer(key, intermValIter, outKV)
% Combines counts from all chunks to produce final counts.
count = 0;
while hasnext(intermValIter)
data = getnext(intermValIter);
count = count + data;
end
add(outKV, key, count)
end
```

- The reducer function is invoked on a chunk of tuples that have the same key
- `intermValIter` is an iterator that returns next tuple

MapReduce invocation

```
outds = mapreduce(ds, @countMapper, @countReducer);
```

- Mapper is invoked once for each chunk of input data
- Reducer is invoked on a chunk of internediate key-value tuples all having same key
- Reducer uses an iterator to get successive tuples

- The problem can be solved with two lines of Matlab, using the `accumarray()` function (to come)
- But, without using MapReduce or other explicitly parallel construct from the Matlab *Parallel Computing Toolbox*, Matlab executes sequentially

Example: Compute Max delay using MapReduce

Mapper

```
function maxTimeMapper(data, ~, intermKVStore)
maxElaspedTime = max(data{:,:});
add(intermKVStore, 'MaxElaspedTime',maxElaspedTime);
end
```

- No info argument
- Tuples are emitted one by one (`add` vs. `multiadd`)

Reducer

```
function maxTimeReducer(~, intermValsIter, outKVStore)
maxElaspedTime = -inf;
while hasnext(intermValsIter)
maxElaspedTime = max(maxElaspedTime, getnext(intermValsIter));
end

add(outKVStore, 'MaxElaspedTime', maxElaspedTime);
end
```

Count number of flights per day for each airline

The datastore contains flight records from 10/1/1987 to 12/31/2008
The mapper constucts tuples where the key is the airline code and the value is a vector of number of flights per day

Mapper

```
function countFlightsMapper(data, ~, intermKVStore)
 dayNumber = days((datetime(data.Year, data.Month, data.DayofMonth)
  -  datetime(1987,10,1)))+1;
 daysSinceEpoch = days(datetime(2008,12,31) - datetime(1987,10,1))+1;
  [airlineName, ~, airlineIndex] = unique(data.UniqueCarrier,
    'stable');
 for i = 1:numel(airlineName)
  dayTotals = accumarray(dayNumber(airlineIndex==i),
     1, [daysSinceEpoch,1]);
  add(intermKVStore, airlineName{i}, dayTotals);
 end
end
```

Unique

`unique`: Matlab function that returns unique entries from a table

```
A = [9 2 9 5];
[C, ia, ic] = unique(A)

C = 2 5 9
ia = 2 4 1
ic = 3 1 3 2
```

- Unique elements are sorted
- `C = A(ia)` and `A = C(ic)`.

```
A = [9 2 9 5];
[C, ia, ic] = unique(A,'stable')

C = 9 2 5
ia = 1 2 4
ic = 1 2 1 3
```

- Unique elements are in original order
- `C = A(ia)` and `A = C(ic)`.

`accumarray(subs, val)`: Matlab function that returns partial sums of elements in `val`, as indicated by indices in `subs`

```
val = 10 11 12 13 14
subs = 1 3  4  3  4

A = accumarray(subs, val)

A = 10 0 24 26
```

```
subs = 1 2 4 2 4

A = accumarray(subs, 1)

A = 1 2 0 2
```

Result is count of number of occurrences of each value in subs

```
subs = 1 2 4 2 4

A = accumarray(subs, 1, [5,1])

A = 1 2 0 2 0
```

Last argument specifies dimensions of result matrix

```
% create array airlineName with each airline name occurring once,
% and array airlineIndexm providing index of arline name in
%each data row
[airlineName, ~, airlineIndex] = unique(data.UniqueCarrier,
'stable');
for i = 1:numel(airlineName)
% accumulate for each day the number of occurences
% of airline name with index i
 dayTotals = accumarray(dayNumber(airlineIndex==i),
   1, [daysSinceEpoch,1]);
% add tuple consisting of airline name and vecor of counts
% per day to intermediate store
 add(intermKVStore, airlineName{i}, dayTotals);
end
```

Reducer

Reducer sums vectors

```
function countFlightsReducer(intermKeysIn, intermValsIter,
outKVStore)
%add, for each airline, the vectors of counts per day
 daysSinceEpoch = days(datetime(2008,12,31)
   - datetime(1987,10,1))+1;
 dayArray = zeros(daysSinceEpoch, 1);
 while hasnext(intermValsIter)
  dayArray = dayArray + getnext(intermValsIter);
 end
 add(outKVStore, intermKeysIn, dayArray);
end
```