Matrix Multiplciation: 2 Two Steps MapReduce

- **Drawbacks of 1.1 One Step MapReduce**
  - If the dimensions of two matrices are unknown? Then, this method does not work.
  - Network communication is heavy: dense emits of individual elements of matrix A_{ik} and B_{kj}.

Solution: Version 2!

- **Two steps MapReduce:**
  - Intermediate products are calculated in 1st reducers
  - Sum these products up at 2nd reducers

Matrix by Matrix Multiplication

- **The operation is done as follows: using index notation:**

$$C_{jk} = \sum_{l=1}^{n} A_{jl} B_{lk}$$

- **for example: C_{ij}**

$\sum_{l=1}^{n} A_{jl} x_l$ $\quad$ $\sum_{l=1}^{n} A_{jl} x_l$

$A_{ij}x_l$ $\quad$ $A_{ij}x_l$ $\quad$ $A_{ij}x_l$ $\quad$ $A_{ij}x_l$

$$AB = \begin{bmatrix} 4 & 3 \\ 7 & 2 \\ 9 & 0 \end{bmatrix} \begin{bmatrix} 2 & 5 \\ 1 & 6 \end{bmatrix} = \begin{bmatrix} 4*2+3*1 & 4*5+3*6 \\ 7*2+2*1 & 7*5+2*6 \\ 9*2+0*1 & 9*5+0*6 \end{bmatrix}$$

$$C = \begin{bmatrix} C11 & C12 \\ C21 & C22 \\ C31 & C32 \end{bmatrix} = \begin{bmatrix} 11 & 38 \\ 16 & 47 \\ 18 & 45 \end{bmatrix}$$

Divide and conquer strategy
Each target value C[j, k] can calculated in parallel
How? get all component data synced: same reducer
Multi step process (i.e., multiple jobs)

Job 1 – [MAPPER] Read input matrices from text files
- Funnel all component terms from A and from B (A[j,l] B[l,$k]_l$)
- [REDUCER] Together and multiply (now got cols in B)

Job 2 [REDUCER] group 1 all $A_{jl}x_l$ and sum

2 Two Steps MapReduce

- **1st map phase:**
  - For each element (*i*, *k*) of A, emit

    Key = *k*, **Value**=(*A, i*, A_{ik}) for *j* in 1…*J*
  - For each element (*k, j*) of B, emit

    Key = *k*, **Value** = (*B*, *j*, B_{kj}) for *i* in 1…*I*

A_{ik} *needs to sync with* B_{k*}

Comes from Matrix A : (A, *, *)
Comes from Matrix B : (B, *, *)

- **1st reduce phase, emit** **#(in memory sync around components of C_{ij})**
  - **Key** = (*i*, *j*)
  - **Value** = $A_{ik} \cdot B_{kj}$

There are *K* reducer's output

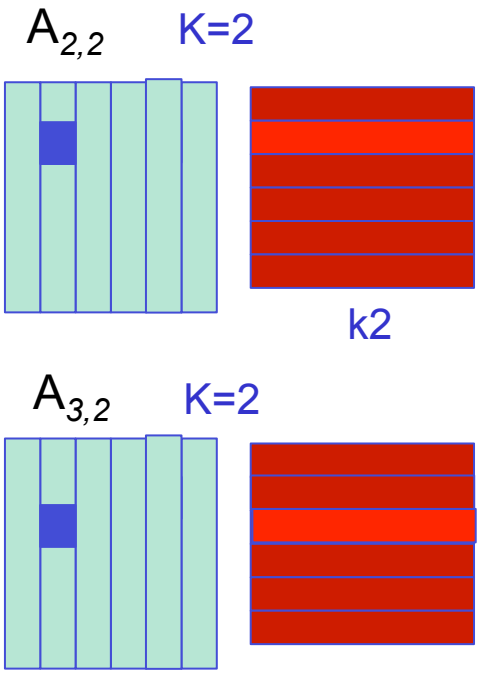
- **2nd map phase, emit**
  - **Key** = (*i*, *j*)
  - **Value** = $A_{ik} \cdot B_{kj}$

Identity output

- **2nd phase, emit**
  - Key = (*i, j*)
  - Value = $sum_k(A_{ik} \cdot B_{kj})$

Sum up the products

First mapper needs to emit



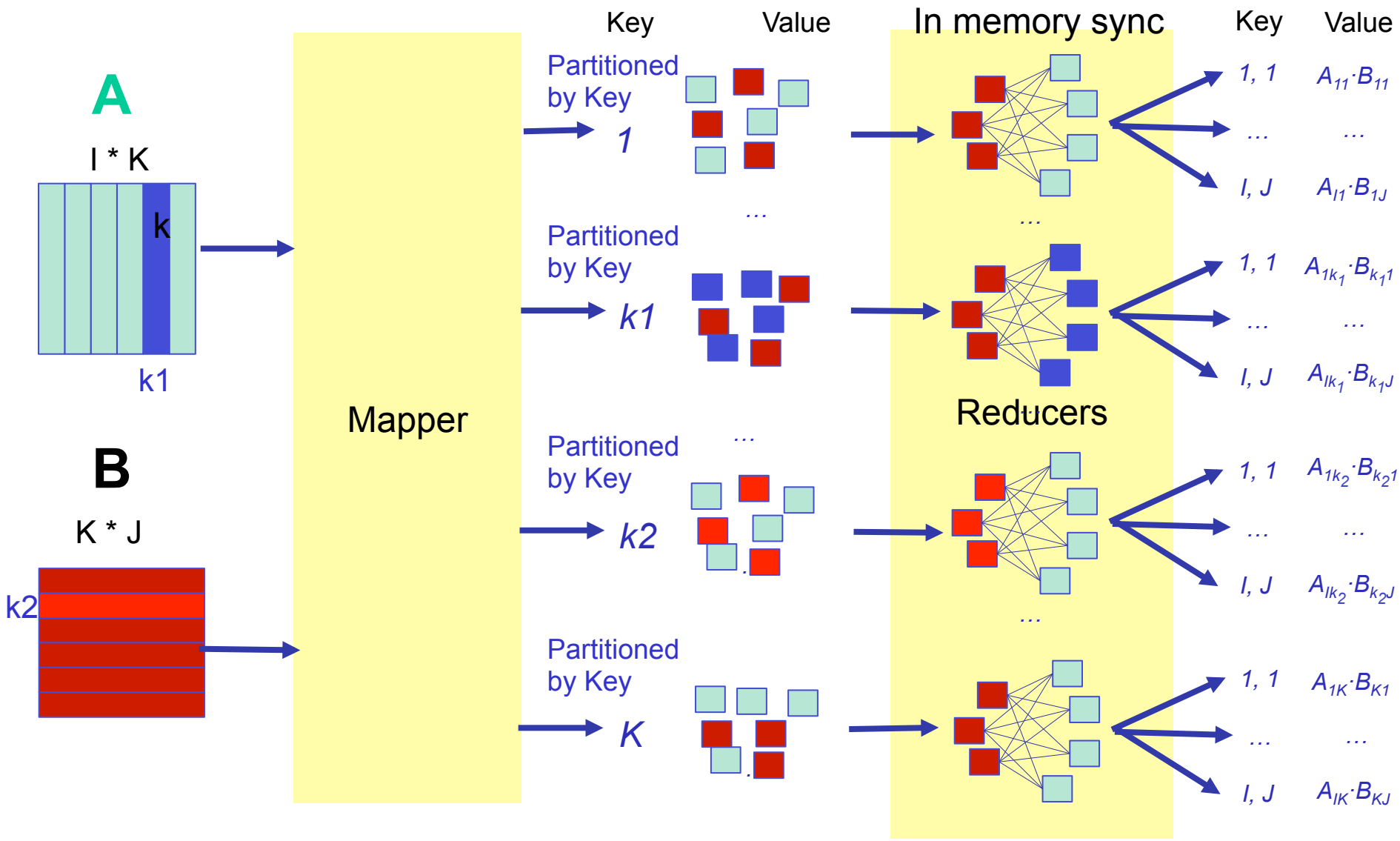
A_{ik} *needs to sync with* B_{k*} ➔ C_{i*}

k, (A, i, A_{ik})

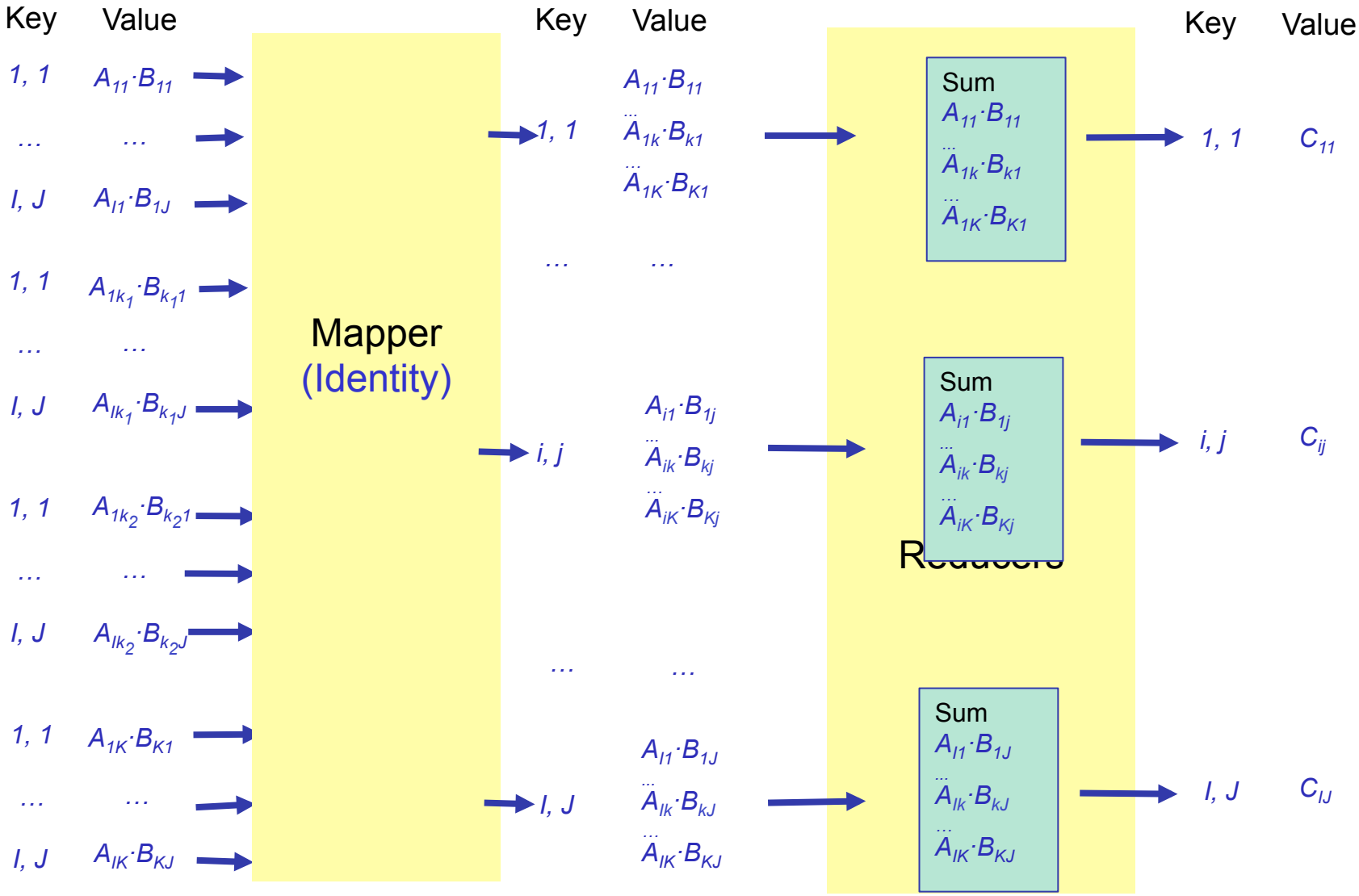
k, (B, j, B_{kj})

k, (B, j, B_{kj})

2 Two Steps MapReduce: diagram (1st MapReduce)



2 Two Steps MapReduce: diagram (2nd MapReduce)



Example: 1st Mapper

Traditional array format RowIndex, ColumnIndex, Value

– Matrix A:

$$\begin{pmatrix} 5 & 0 \\ 3 & 8 \\ 0 & 6 \end{pmatrix}$$

(0,0), 5
(1,0), 3
(1,1), 8
(2,1), 6

– Matrix B:

$$\begin{pmatrix} 6 & 3 \\ 2 & 0 \end{pmatrix}$$

(0,0), 6
(0,1), 3
(1,0), 2

1st Mapper

A

(0,0),5
(1,1),8
(1,0),3
(2,1),6

Key k	Value (A, i, A_{ik})
0	A, 0, 5
1	A, 1, 8
0	A, 1, 3
1	A, 2, 6

B

(0,0),6
(0,1),3
(1,0),2

Key k	Value (B, J, B_{kJ})
0	B, 0, 6
0	B, 1, 3
1	B, 0, 2

Example: 1ST Reducer

In memory sync of values associated with k
Basically all data shipped to reducer k needs to be loaded in memory…
Need in memory hashing

Key	Value (flag,index, value)
0	(A, 0, 5), (A, 1, 3) (B, 0, 6), (B, 1, 3)
1	(A, 1, 8),(A, 2, 6) (B, 0, 2)

Cross Product of A and B

Key (i,j)	Value C(*i*,*j*)
0, 0	30
0, 1	15
1, 0	18
1, 1	9
1, 0	16
2, 0	12

Send them to 2nd MapReduce

Example: 2nd MapReduce

Aggregate value by keys (*i,j*)

Key (i,j)	Value C(*i,j*)
0, 0	30
0, 1	15
1, 0	18
1, 1	9
1, 0	16
2, 0	12

Key (i,j)	Value C(*i,j*)
0, 0	30
0, 1	15
1, 0	34
1, 1	9
2, 0	12
2, 1	0

TRACE of 2Job M by M Multiplication

- **Trace of the two map-reduce job version of the matrix by matrix multiplication**

Matrix-vector product

$$\mathbf{Ax} = \mathbf{y}$$

$$y_i = \sum_k A_{ik} x_k$$

Follow along!
mapreduce-matrix-tutorial
/codes/smatvec.py

Sparse Matrix

Row number [column number, value]
E.g., value for [0,0] is encoded as
0 0 0.12537732342

6 lines (5 sloc) | 0.127 kb

```
1  0 0 0.12527732342 3 1.02407852061 4 0.121151207685
2  1 0 0.597062100484
3  2 2 1.24708888756
4  3 4 -1.45057798535
5  4 2 0.0618772663296
```

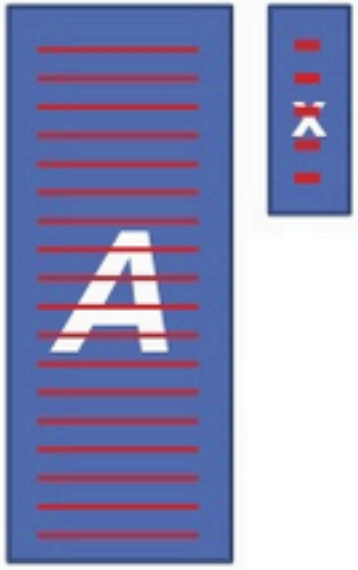
https://github.com/dgleich/matrix-hadoop-tutorial/blob/master/samples/smat_5_5.txt

Matrix-vector product

$$\mathbf{Ax} = \mathbf{y}$$

$$y_i = \sum_k A_{ik} x_k$$

Follow along!
mapreduce-matrix-tutorial
/codes/smatvec.py



A is stored by row

```
$ head samples/smat_5_5.txt
0 0 0.125 3 1.024 4 0.121
1 0 0.597
2 2 1.247
3 4 -1.45
4 2 0.061
```

x is stored entry-wise

```
$ head samples/vec_5.txt
0 0.241
1 -0.98
2 0.237
3 -0.32
4 0.080
```

Key value

Sparse matrix matrix product 2-step

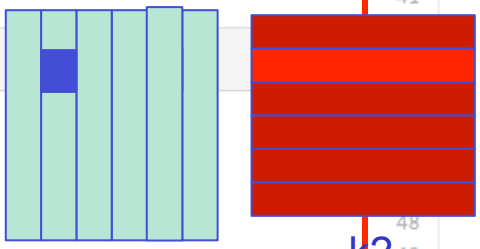
dgleich / matrix-hadoop-tutorial

branch: master ▾ matrix-hadoop-tutorial / codes / matmat.py

dgleich on Apr 8, 2013 Working versions of codes

1 contributor

74 lines (60 sloc) | 2.381 kb




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

from mrjob.job import MRJob
from mrjob.compat import get_jobconf_value
import itertools

class SparseMatMatMult(MRJob):
    """ First, join the two matrices based on columns, then output
    the cartesian product of the columns, which forms the input for
    the product. """

    def configure_options(self):
        super(SparseMatMatMult,self).configure_options()
        self.add_passthrough_option('--A-matrix',default='A',
            dest='Amatname')

    def parsemat(self):
        """ Return 1 if this is the A matrix, otherwise return 2"""
        fn = get_jobconf_value('map.input.file')
        if self.options.Amatname in fn:
            return 1
        else:
            return 2

    def joinmap(self, key, line):
        mtype = self.parsemat()
        vals = [float(v) for v in  line.split()]
        row = int(vals[0])
        rowvals = [(int(vals[i]),vals[i+1]) for i in xrange(1,len(vals),2)]
        if mtype==1:
            # rowvals are the entries in the row
            # we output the entire row for each column
            for val in rowvals:
                # reorganize data by columns
                yield (val[0], (row, val[1]))
        else:
            yield (row, (rowvals,))

    def joinred(self, key, vals):
        # each key is a column of the matrix.
        # and there are two types of values:
        #  len == 2 (1, row, A_row,key) # a column of A
        #  len == 1 rowvals # a row of B

        # load the data into memory
        brow = []
        acol = []
        for val in vals:
            if len(val) == 1:
                brow.extend(val[0])
            else:
                acol.append(val)

        for (bcol,bval) in brow:
            for (arow,aval) in acol:
                yield ((arow,bcol), aval*bval)

    def sumred(self, key, vals):
        yield (key, sum(vals))

    def rowgroupmap(self, key, val):
        yield key[0], (key[1], val)

    def appendred(self, key, vals):
        yield key, list(itertools.chain.from_iterable(vals))

    def steps(self):
        return [self.mr(mapper=self.joinmap, reducer=self.joinred),
            self.mr(mapper=None, reducer=self.sumred),
            self.mr(mapper=self.rowgroupmap, reducer=self.appendred)]

if __name__=='__main__':
    SparseMatMatMult.run()
```

Map Job1

Reduce Job1

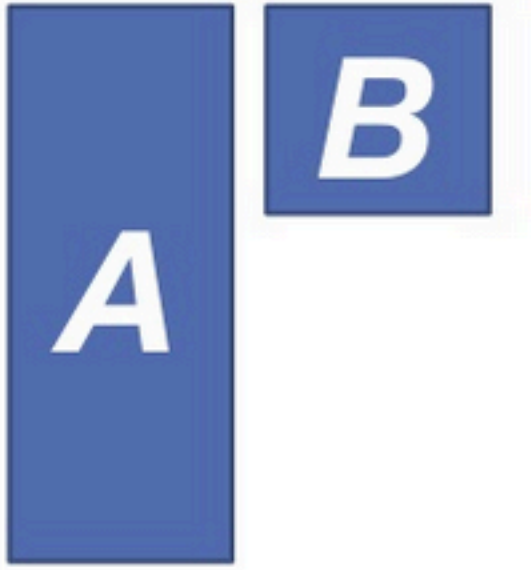
More coordination around the B matrix
In reducer; and also have a mapper in job 2

Reduce Job2

Map-Reduce Job3
Prettify output matrix

- **Matrix-matrix product**

  Follow along!
  `mapreduce-matrix-tutorial/codes/matmat.py`

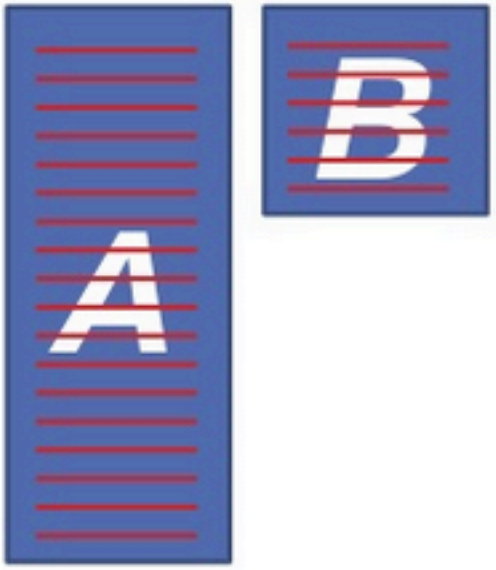


$$AB = C$$
$$C_{ij} = \sum_{k} A_{ik} B_{kj}$$

Matrix-matrix product

$$AB = C$$
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Follow along!
mapreduce-matrix-tutorial
/codes/matmat.py



A is stored by row

```
$ head samples/smat_10_5_A.txt
0 0 0.599 4 -1.53
1
2 2 0.260
3
4 0 0.267 1 0.839
```

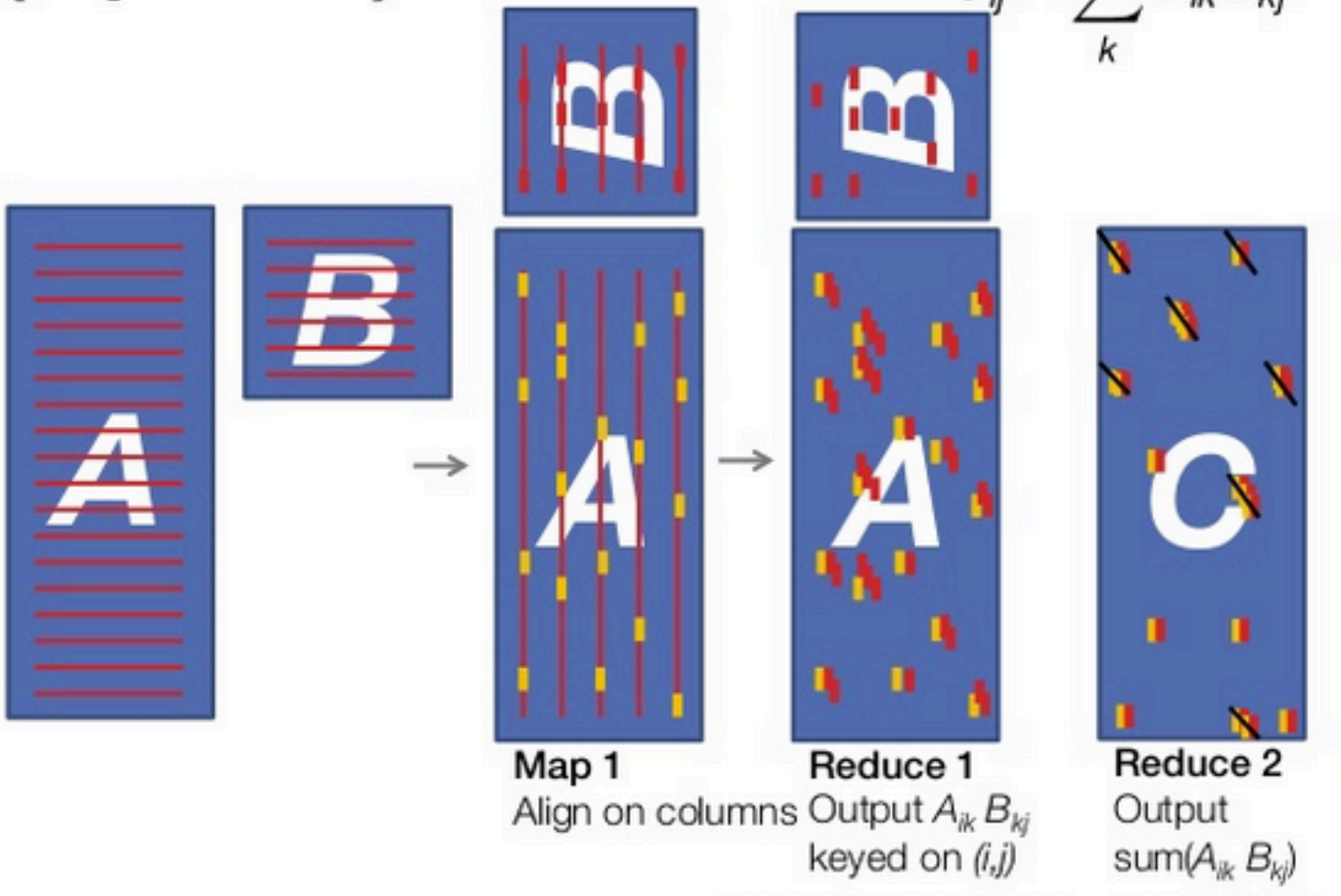
B is stored by row

```
$ head samples/smat_5_5.txt
0 0 0.125 3 1.024 4 0.121
1 0 0.597
2 2 1.247
```

Matrix-matrix product (in pictures)

$$AB = C$$

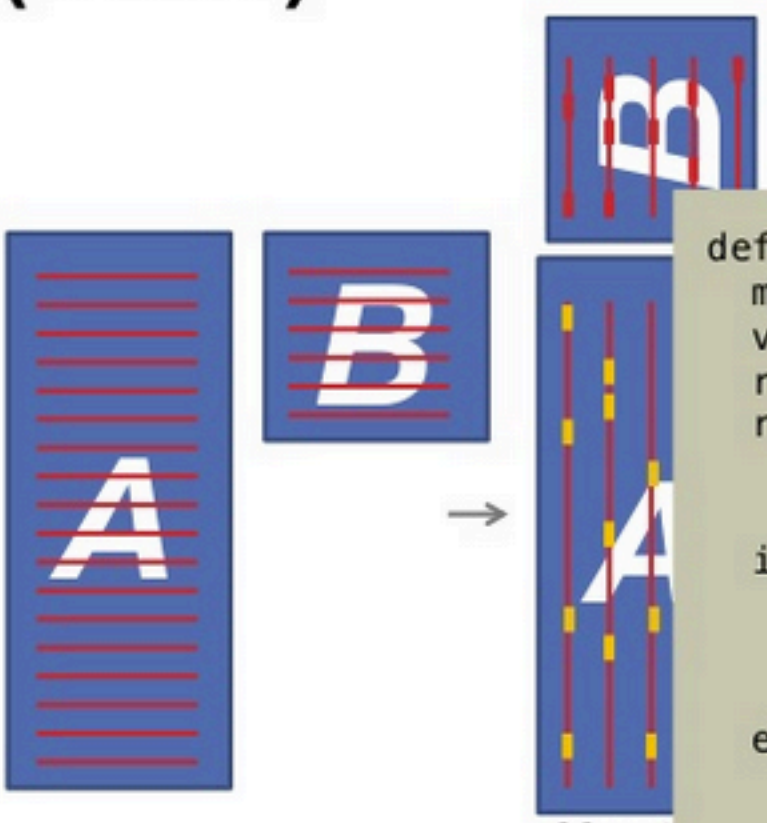
$$C_{ij} = \sum_k A_{ik} B_{kj}$$



Matrix-matrix product (in code)

$$AB = C$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

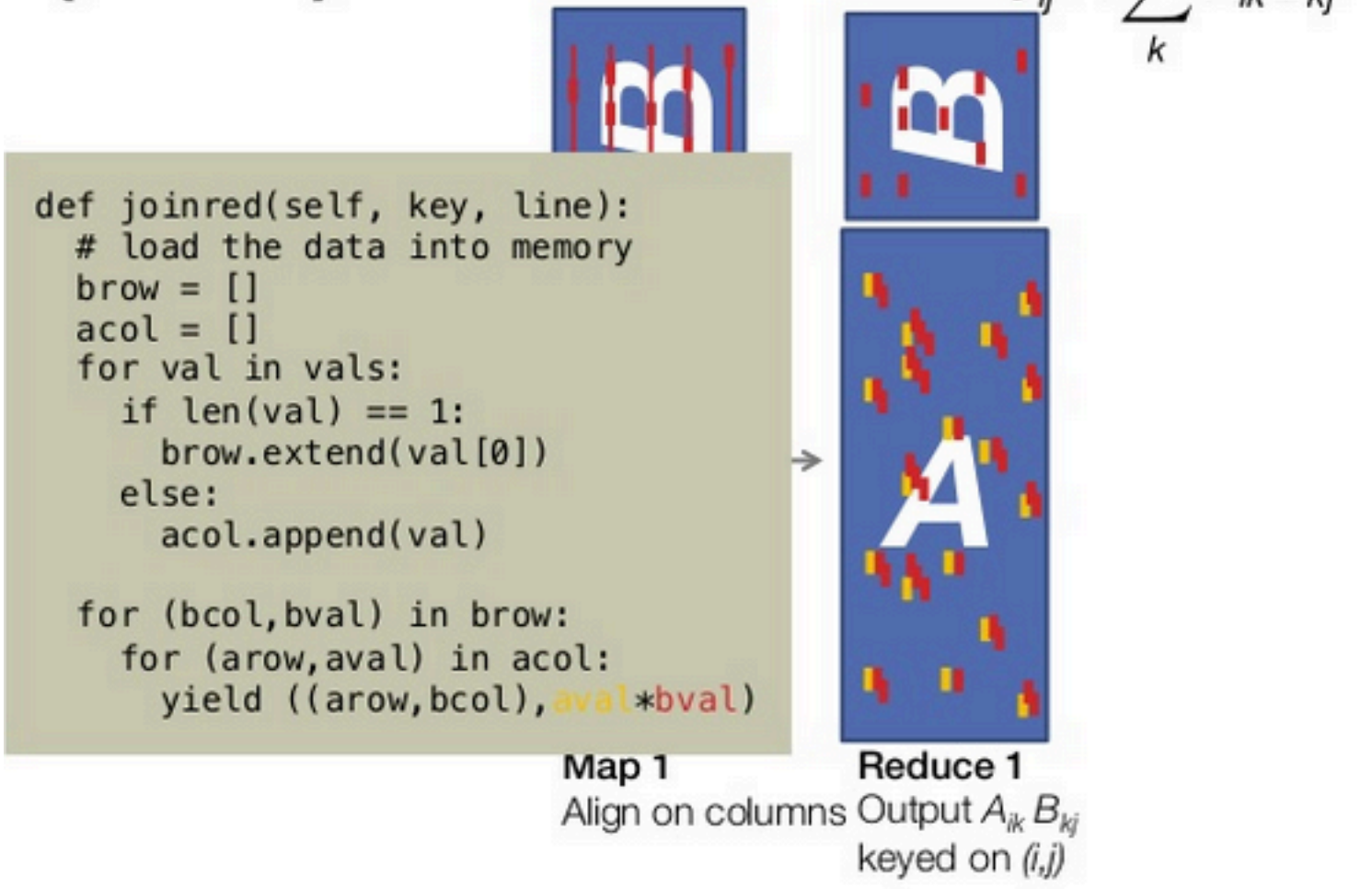



```
def joinmap(self, key, line):
  mtype = self.parsemat()
  vals = line.split()
  row = vals[0]
  rowvals = \
   [(vals[i],float(vals[i+1]))
    for i in xrange(1,len(vals),2)]
  if mtype==1:
    # matrix A, output by col
    for val in rowvals:
      yield (val[0], (row, val[1]))
  else:
    yield (row, (rowvals,))
```

Matrix-matrix product (in code)

$$AB = C$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

- 

Matrix-matrix product (in pictures)

$$AB = C$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$