Map-Reduce Examples

Facebook Friends Finder: (it's just an example)

- FB has a list of friends (note that friends are a bi-directional thing on Facebook. If I'm your friend, you're mine).
- They also have lots of disk space and they serve hundreds of millions of requests everyday. They've decided to pre-compute calculations when they can to reduce the processing time of requests.

FB (Contd.)

- One common processing request is the "You and Joe have 230 friends in common" feature.
- When you visit someone's profile, you see a list of friends that you have in common. This list doesn't change frequently so it'd be wasteful to recalculate it every time you visited the profile .

Problem Statement

- Use mapreduce so that we can calculate everyone's common friends once a day and store those results. Later on it's just a quick lookup. We've got lots of disk, it's cheap.

- Assume the friends are stored as Person->[List of Friends], our friends list is then:
  - A -> B C D
  - B -> A C D E
  - C -> A B D E
  - D -> A B C E
  - E -> B C D
- Each line will be an argument to a mapper. For every friend in the list of friends, the mapper will output a key-value pair. The key will be a friend along with the person. The value will be the list of friends.

- For map(A -> B C D) :
  - (A B) -> B C D
  - (A C) -> B C D
  - (A D) -> B C D
- For map(B -> A C D E) : (Note that A comes before B in the key)
  - (A B) -> A C D E
  - (B C) -> A C D E
  - (B D) -> A C D E
  - (B E) -> A C D E

- For map(C -> A B D E) :
  - (A C) -> A B D E
  - (B C) -> A B D E
  - (C D) -> A B D E
  - (C E) -> A B D E
- For map(D -> A B C E) :
  - (A D) -> A B C E
  - (B D) -> A B C E
  - (C D) -> A B C E
  - (D E) -> A B C E

- And finally for map(E -> B C D):
  - (B E) -> B C D
  - (C E) -> B C D
  - (D E) -> B C D

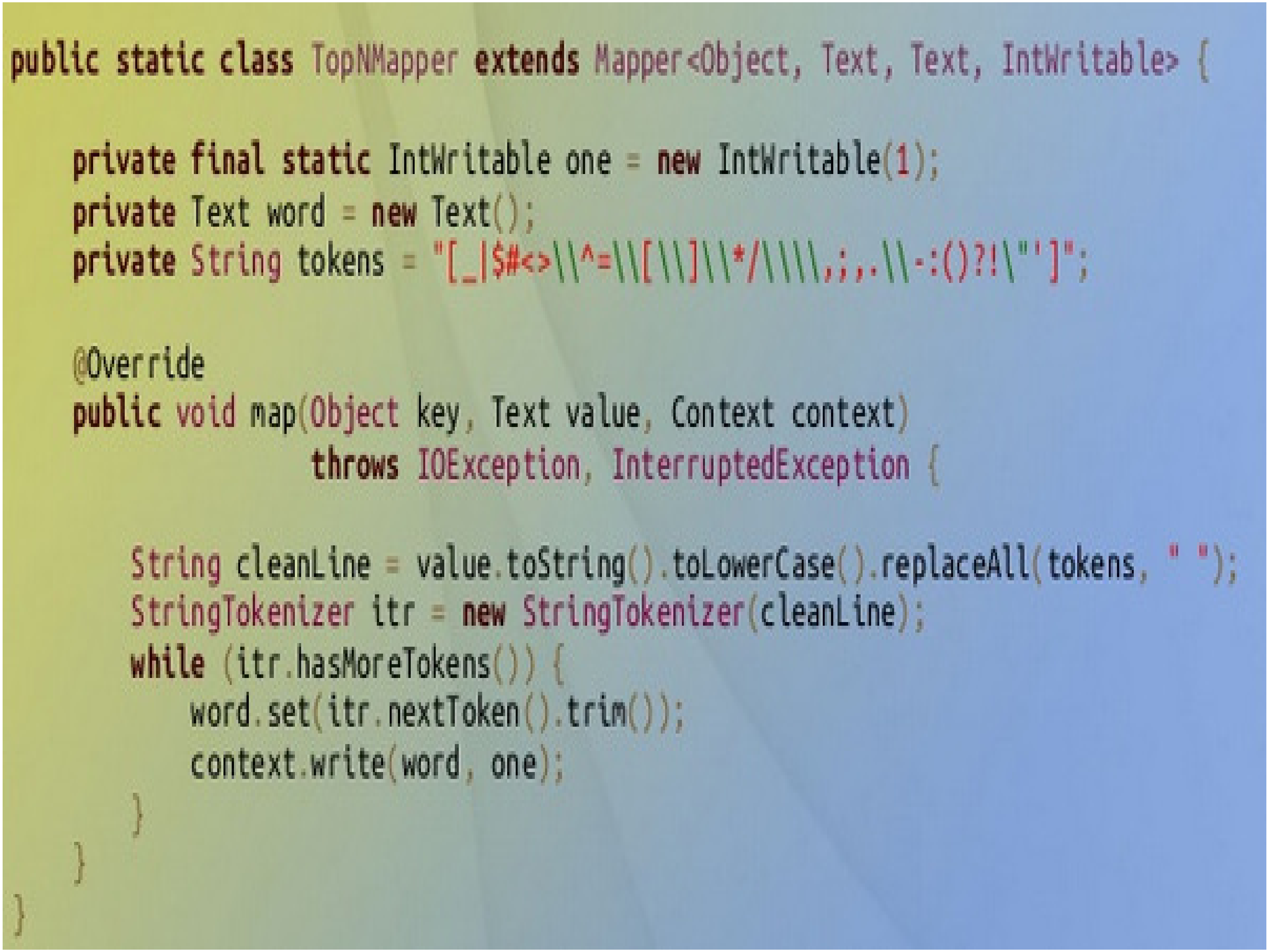
- Before sending these key-value pairs to the reducers, Group them by their keys and get:
  - (A B) -> (A C D E) (B C D)
  - (A C) -> (A B D E) (B C D)
  - (A D) -> (A B C E) (B C D)
  - (B C) -> (A B D E) (A C D E)
  - (B D) -> (A B C E) (A C D E)
  - (B E) -> (A C D E) (B C D)
  - (C D) -> (A B C E) (A B D E)
  - (C E) -> (A B D E) (B C D)
  - (D E) -> (A B C E) (B C D)

- Each line will be passed as an argument to a reducer. The reduce function will simply intersect the lists of values and output the same key with the result of the intersection. For example:
- reduce((A B) -> (A C D E) (B C D)) will output (A B) : (C D) and means that friends A and B have C and D as common friends.

- The result after reduction is:
  - (A B) -> (C D)
  - (A C) -> (B D)
  - (A D) -> (B C)
  - (B C) -> (A D E)
  - (B D) -> (A C E)
  - (B E) -> (C D)
  - (C D) -> (A B E)
  - (C E) -> (B D)
  - (D E) -> (B C)
- Now when D visits B's profile, we can quickly look up (B D) and see that they have three friends in common, (A C E).

TopN :
top-n used words of a text file Input

- We want to find the top-n used words of a text file Input Data: The text of the book ”Flatland” By E. Abbott. Source:

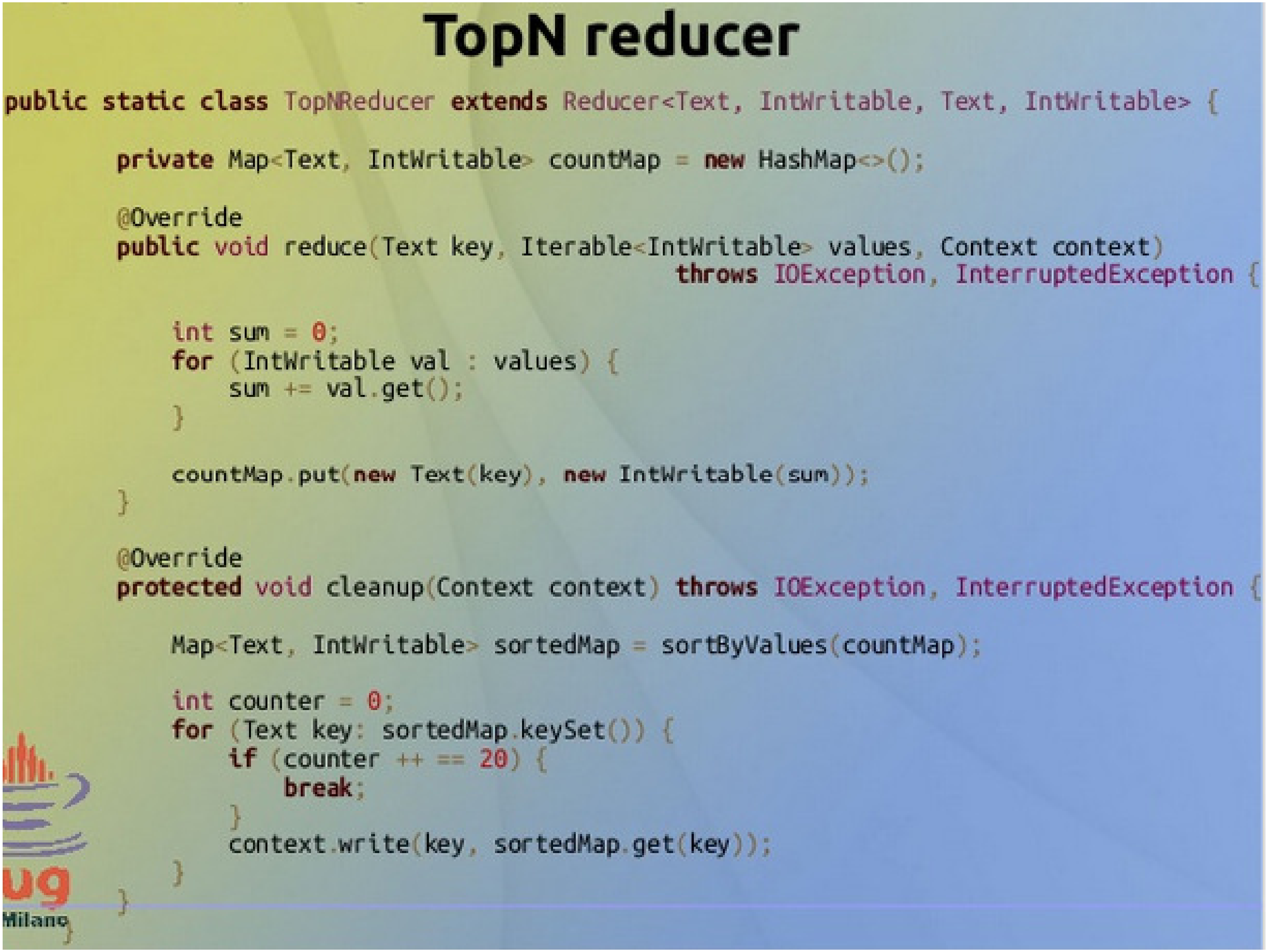


```
public static class TopNMapper extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private String tokens = "[_|$#<>\\^=\\[\\]\\*/\\\\,;,.\\-:()?!\"']";

    @Override
    public void map(Object key, Text value, Context context)
                    throws IOException, InterruptedException {

        String cleanLine = value.toString().toLowerCase().replaceAll(tokens, " ");
        StringTokenizer itr = new StringTokenizer(cleanLine);
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken().trim());
            context.write(word, one);
        }
    }
}
```



TopN reducer

```
public static class TopNReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private Map<Text, IntWritable> countMap = new HashMap<>();

        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                                                throws IOException, InterruptedException {

            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }

            countMap.put(new Text(key), new IntWritable(sum));
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {

            Map<Text, IntWritable> sortedMap = sortByValues(countMap);

            int counter = 0;
            for (Text key: sortedMap.keySet()) {
                if (counter ++ == 20) {
                    break;
                }
                context.write(key, sortedMap.get(key));
            }
        }
    }
```

TopN

Results:

```
the     2286
of      1634
and     1098
to      1088
a       936
i       735
in      713
that    499
is      429
you     419
my      334
it      330
as      322
by      317
not     317
or      299
but     279
with    273
for     267
be      252
...
```

- In the shuffle and sort phase, the partioner will send every single word (the key) with the value ”1” to the reducers. All these network transmissions can be minimized if we reduce locally the data that the mapper will emit. This is obtained by a Combiner.

References

- https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/
- http://www.slideshare.net/andreaiacono/mapreduce-34478449
- http://stevekrenzel.com/finding-friends-with-mapreduce