Extreme Computing

Map-Reduce 2

Stratis Viglas

School of Informatics
University of Edinburgh
sviglas@inf.ed.ac.uk

Programming Model

Examples

Efficiency

Programming Model

MR offers one restricted version of parallel programming:

- Coarse-grained.
- No inter-process communication.
- Communication is (generally) through files.

Programming Model

Mapping:

- The input data is divided into *shards*.
- The *Map* operation works over each shard and *emits key-value* pairs.
- Each mapper works in parallel.

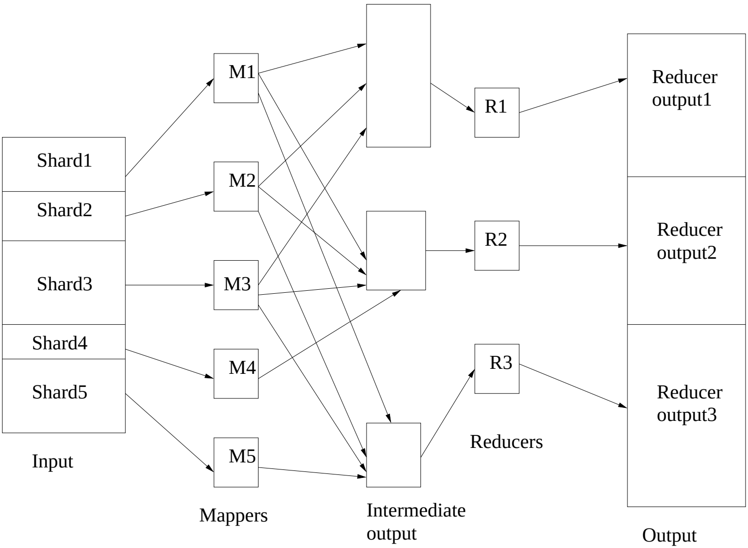
Keys and values can be anything which can be represented as a string.

Programming Model

Reducing:

- After mapping, each key-value pair is hashed on the key.
- Hashing sends that key-value pair to a given *reducer*.
  - All keys that hash to the same value are sent to the same reducer.
- The input to a reducer is sorted on the key.
  - Sorted input means that related key-value pairs are locally grouped together.

Programming Model



Programming Model

Note:

- Each Mapper and Reducer runs in parallel.
- There is no state sharing between tasks.
  - Task communication is achieved using either external resources or at start-time
- There need not be the same number of Mappers as Reducers.
  - It is possible to have *no* Reducers.

Programming Model

Note:

- Tasks read their input sequentially.
  - Sequential disk reading is far more efficient than random access
- Reducing starts once Mapping ends.
  - Sorting and merging etc can be interleaved.

Example: System Maintenance

Example

Check the health of a set of machines. Report on any that are bad.

- The input data will be a list of machine names
  - borg1.com, borg2,com, . . . borg12123.com
- The output will be reports
- There is no need to run a reducing stage

Example: System Maintenance

Mapper:

- Read each line, giving a machine name,
- Log into that machine and collect statistics.
- Emit statistics

Example: Word Counting

Example

Count the number of words in a collection of documents

- Our Mapper counts words in each shard.
- The Reducer gathers together partial counts for a given word and sums them

Example: Word Counting

Mapper:

- For each sentence, emit *word, 1* pair.
  - The key is the *word*
  - The value is the number 1

Example: Word Counting

Reducer:

- Each Reducer will see all instances of a given word.
- Sequential reads of the reducer input give partial counts of a word.
- Partial counts can be summed to give the total count.

Example: Word Counting

Input sentences:

- *the cat*
- *the dog*

Key	Value	
the	1	
cat	1	Mapper output
the	1	
dog	1	

Example: Word Counting

Reducer 1 input	Reducer 2 input
the, 1	*cat, 1*
the, 1	
dog, 1	

Reducer 1 output	Reducer 2 output
the, 2	*cat, 1*
dog, 1	

Map Reduce Efficiency

MR algorithms involve a lot of disk and network traffic:

- We typically start with Big Data
- Mappers can produce intermediate results that are *bigger* than the input data.
- Task input may not be on the same machine as that task.
  - This implies network traffic
- Per-reducer input needs to be sorted.

Map Reduce Efficiency

Sharding might not produce a balanced set of inputs for each Reducer:

- Often, the data is heavily skewed
  - Eg all function words might go to one Reducer
- Having an imbalanced set of inputs turns a parallel algorithm into a sequential one

Map Reduce Efficiency

Selecting the right number of Mappers and Reducers can improve speed

- More tasks mean each task might fit in memory / require less network access
- More tasks mean that failures are quicker to recover from.
- Fewer tasks have less of an over-head.

This is a matter of guess-work

Map Reduce Efficiency

Algorithmically, we can:

- Emit fewer key-value pairs
  - Each task can locally aggregate results and periodically emit them.
  - (This is called *combining*)
- Change the key
  - Key selection implies we partition the output. Some other selection might partition it more evenly

Summary

- Introduced the MR programming model
- Sample MR applications
- Looked at efficiency