Weekly meeting

Week 5

Le Duc Tung

PRL group, NII, Tokyo

Nov 21, 2012

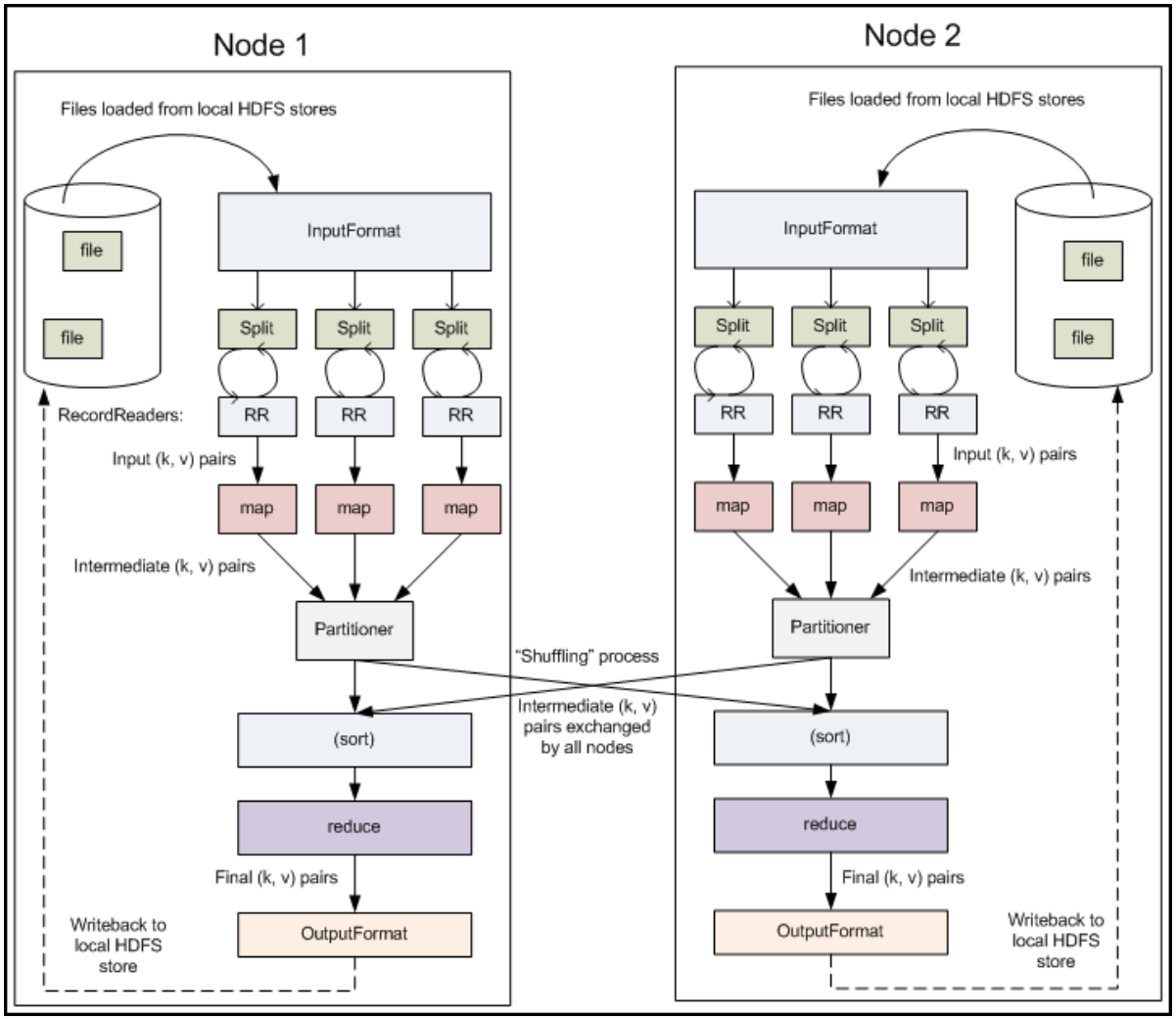
Last week

- Described Quicksort using MapReduce, however,
  - Required many MapReduce tasks: ($log\ n$) to n steps.

Terasort using Hadoop

- Written by Owen O'Malley and Arun Murthy at Yahoo Inc.
- Won the annual general purpose terabyte sort benchmark in 2008 and 2009.
  - In 2008, 910 nodes x (4 dual-core processors, 4 disks, 8 GB memory). Sorting 1TB data in 3.48 minutes.
  - In 2009, 3452 nodes x (2 Quadcore Xeons, 8 GB memory, 4 SATA). Sorting 100TB data in 173 minutes.
- Terasort includes 3 MapReduce applications:
  - Teragen: generates the data.
  - Terasort: samples the input data and uses them with MapReduce to sort the data.
  - Teravalidate: validates the output is sorted.

A closer look at MapReduce's implementation model



"source: http://developer.yahoo.com/hadoop/tutorial/module4.html"

Teragen

- Input: The number of rows and the output directory.
- Output format:

```
.t^#\|v$2\      0AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHH
75@~?'WdUF      1IIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNNNNOOOOOOOOOOPPPPPPPP
w[o||:N&H,      2QQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXX
^Eu)<n#kdP      3YYYYYYYYYYZZZZZZZZZZAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFF
+l-$$0E/ZH      4GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNN
```

MapReduce for Teragen

- Preparation of data for Map task.
  - The number of rows $\overset{InputSplit}{\rightarrow}$ splits of (startid, num of rows).
  - (startid, num of rows) $\overset{RecordReader}{\rightarrow}$ key-value pairs of (rowid, null)
- Example:
  - We need generate data of 100 rows
  - We use 2 mappers
  - We generate 2 splits for 2 mappers.
    - Split 0: consists of two values: startid = 0, number of rows = 50.
    - Split 1: consists of two values: startid = 50, number of rows = 50.
  - Key-value pairs generated from split 0:
    - (0, null), (1, null), ..., (49, null)
  - Key-value pairs generated from split 1:
    - (50, null), (51, null), ..., (99, null)

Map and Reduce task

- Map
  - Input is a pair of (rowid, null)
  - Output is a pair of (key, value), in which
    - Step 1: Create a random generator based on "rowid".
    - Step 2: Create a key that is a text of 10 random bytes.
    - Step 3: The value is a text of: "rowid + 7 blocks of 10 characters + 1 block of 8 characters"
    - Characters are got from ['A' .. 'Z']
- Reduce task
  - Does nothing.

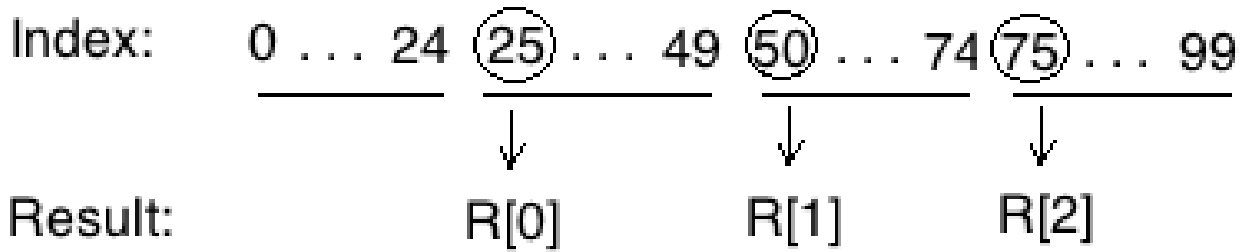
```
.t^#\|v$2\      0AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHH
75@~?'WdUF      1IIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNNNNOOOOOOOOOOPPPPPPPP
w[o||:N&H,      2QQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXX
^Eu)<n#kdP      3YYYYYYYYYYZZZZZZZZZZAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFF
+l-$$0E/ZH      4GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNN
```

Terasort

- Step 1: (Before submitting job) Generate the sample keys by sampling the input. It is a sorted list of sampled keys.
- Step 2: (Using Partitioner) Distribute keys to the correspondent reducers using sample keys.
- Example: If we have N reducers.
  - Generate (N-1) sample keys.
  - All keys such that $\mathbf{sample[i-1]} \leq \mathbf{key} \leq \mathbf{sample[i]}$ are sent to reduce i
  - The above guarantees that the output of reduce i are all less than the output of reduce $(i+1)$

Generate sample keys

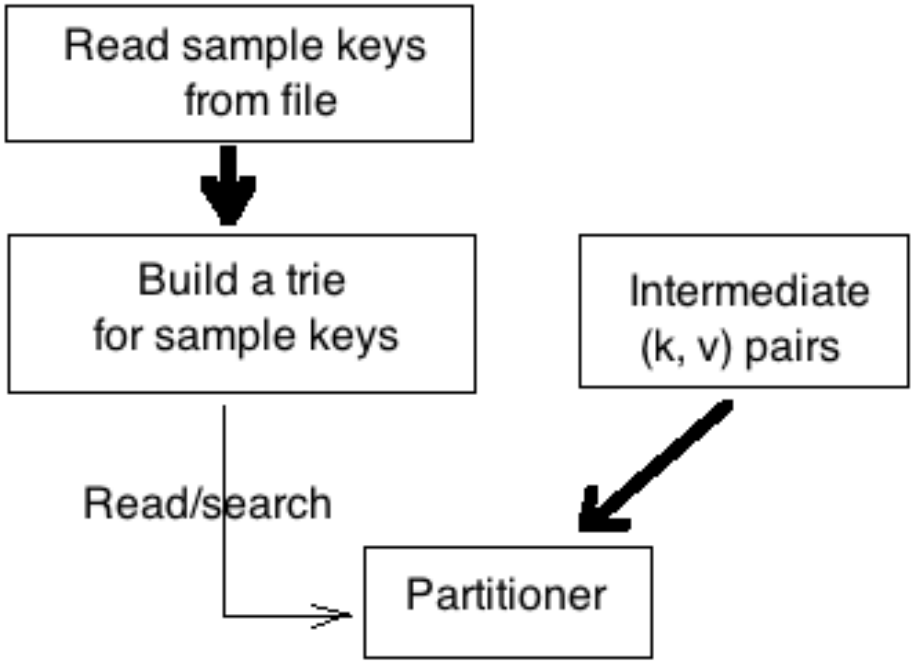
- From the input, create an array R containing all keys or the maximum of 100.000 keys (1MB). However, we can specify how many elements the array has.
- Using Quicksort to sort keys in the array.
- Create an array of (N-1) sample keys from the array R, in which N is the number of reducers, such that,
  - $(i-1)$-th element in the new array is $(i * stepSize)$-th element in the original array.
  - stepSize is of (No. of elements of R)/(No. of reducers)
- List of sampled keys is written into HDFS.



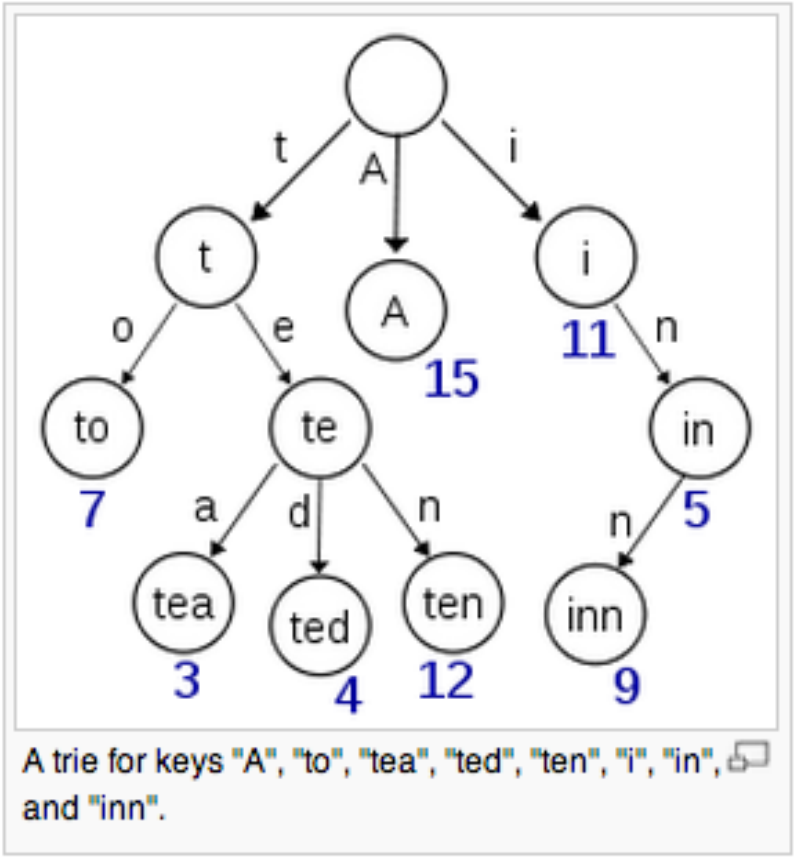
Partitioner

```
public interface Partitioner<K2, V2> extends JobConfigurable {
  /**
   * Get the partition number for a given key (hence record)
   * given the total number of partitions
   * i.e. number of reduce-tasks for the job.
   *
   * <p>Typically a hash function on a all or a subset of
   * the key.</p>
   *
   * @param key the key to be partitioned.
   * @param value the entry value.
   * @param numPartitions the total number of partitions.
   * @return the partition number for the <code>key</code>.
   */
  int getPartition(K2 key, V2 value, int numPartitions);
}
```

Partitioner and Trie tree



- getPartition() function will return the position of k in the trie tree.

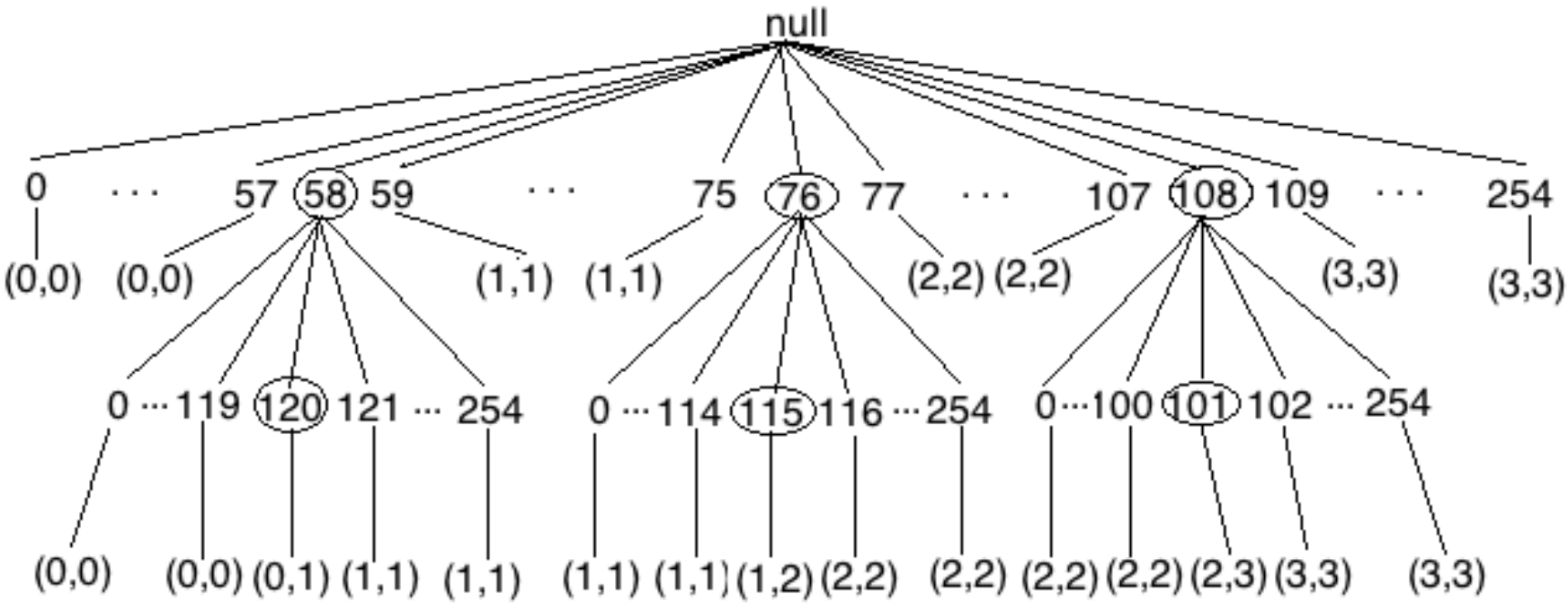


A trie for keys "A", "to", "tea", "ted", "ten", "i", "in", and "inn".

" source: http://en.wikipedia.org/wiki/Trie"

Trie for sample keys

- Assume that we have three sample keys:
  - **:x=P[Q%D¡**, in which, ASCII code of : and x is 58 and 120 respectively.
  - **LsS8)—.ZLD**, in which, ASCII code of L and s is 76 and 115 respectively.
  - **le5awB.$sm**, in which, ASCII code of l and e is 108 and 101 respectively.
- Using the 2 first bytes of keys for building a 2-level trie tree.



Default sort

- By default, data will be sorted before passing to reducers.
- Reduce task does nothing, it just outputs the (K, V) pairs to output files.