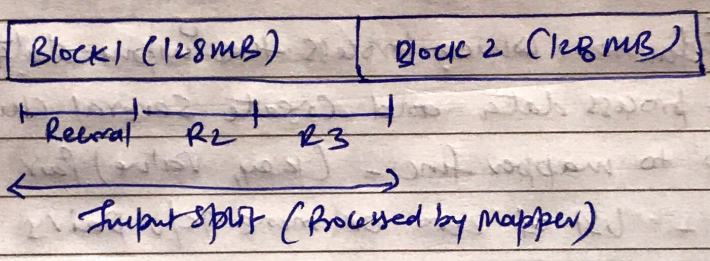
InputSplit

* logical representation of data
* unit of work that contain a single map task.
* Represent data which is processed by an individual mapper.
* Split —divide→ records —Process→ Mapper
  (key, value)
* Doesn't Contain the data, reference to data
* Inputformat create Input Split and divide into record
  - ↳ file input format — break into 128 MB Chunks (default)
  - ↳ mapred.min.Split.Size (mapred.xml)



* Using Starting record in block + byte offset — get Complete record even if it spans the block boundary

Input Split	HDFS Block
* logical rep of data	* Physical rep of data
* Appx equal to blocksize by default — Can Configure	* Default size — 128 MB
* Block1 | Block2 — file split into 2 blocks	Block 1 | Block2 — Input Split (logical grouping of blocks)
* No of map task (mapper) are equal to no of Input Split	

extramarks

Record Reader

* load data from source and convert into key-value pair suitable for reading by mapper.
* uses start / end byte position - generating k/v pair.
* Communicate with Input Split until file reading complete

works

* iterate over records
* map task uses one record to generate key-value pair - Passed to map function

Mapper

* A function which process the input data
* It process data and create several chunks of data
* I/P to mapper func - (key, value) pair
* O/P - list of zero or more k/v pairs
* May use or completely ignore the input keys

eg - Standard pattern - read one line at a time
- key byte offset, value - Content of line
- key is considered irrelevant
- If mapper write anything out, the output must be in the form of key/value pairs

eg

① Upper Case mapper
map (k,v) = emit (k.toupper, v.toUpper)

② Explode
map (k,v) = foreach char c in v emit (k,c)

extramarks

(3) filter
map(k,v) = if (isprime(v)) then emit(k,v)

(4) Changing keyspace
map(k,v) = emit(v.length(), v)

(5) Identity mapper
map(k,v) = emit(k,v)

Reducer

* Process o/p of the mapper
* After processing produces a new set of o/p.
* Take intermediate key-value pair as I/p and runs reducer function on each of them.
  ↳ eg aggregate, ~~filter~~, combine the data
* Reducers run in parallel - as independent
* User can decide the no of reducer (def - 1)
* HDFS store the o/p data
* Job.setNumReduceTask(int) - Set no of reducer

3 Phases

1) Shuffle - Reducer copies the sorted out put from each mapper using HTTP across the n/w
2) Sort - Framework merge sort reduce input by keys (diff mapper - may have same key)
2) ↳ secondary sort
3) Reduce - reduce method is called on each key in the sorted input
the o/p of reduce task to RecordWriter

extramarks * o/p is not sorted

eg 1) Sum reducer

```
reduce(K, vals) = Sum = 0;
    foreach int i in vals:
        Sum += i
    emit(K, Sum)
```

2) Average reducer

```
reduce(K, val) = Sum = 0; Counter = 0;
    foreach(int i in Val:
        Sum += i; Counter += 1;
    emit(K, Sum/Counter)
```

3) Identity reducer

```
reduce(K, val) = foreach v in val:
        emit(K, v)
```

Speculative Execution

* MR breaks job → task → run in parallel to reduce overall exe time
* Sensitive to slow task - Slowdown the exe of job
* Reason - H/w degradation - Soft misconfiguration
* Hadoop doesn't try to fix, it detects and run
* It launches another parallel / backup task for each task that is performing slow than expected on faster nodes
* these are called Speculative task.

extramarks

Assignment - Benefit/Limitation Case of S.E

Date: Page: 15

* Task Complete - Inform Job tracker.
* Whichever copy finishes first - definitive copy
* Kill other task (Slow)
* Enabled by default
* - mapred.map.task.speculative.execution
  - mapred.reduce.task.speculative.execution

] Config