Custom applications with Spark's RDD

Tejas Patil

Facebook

Agenda

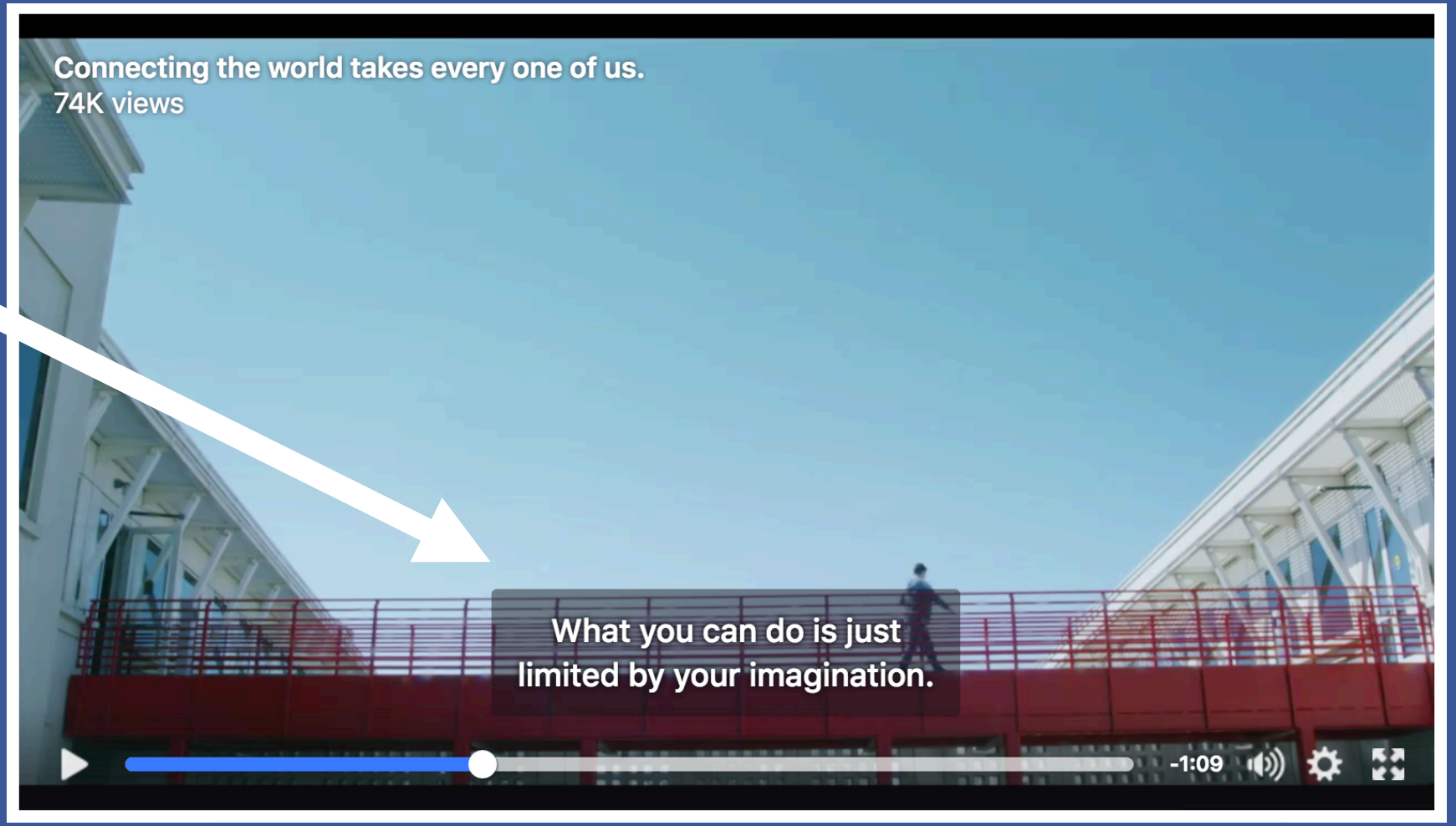
- Use case
- Real world applications
- Previous solution
- Spark version
- Data skew
- Performance evaluation

N-gram language model training

5-gram

Real world applications

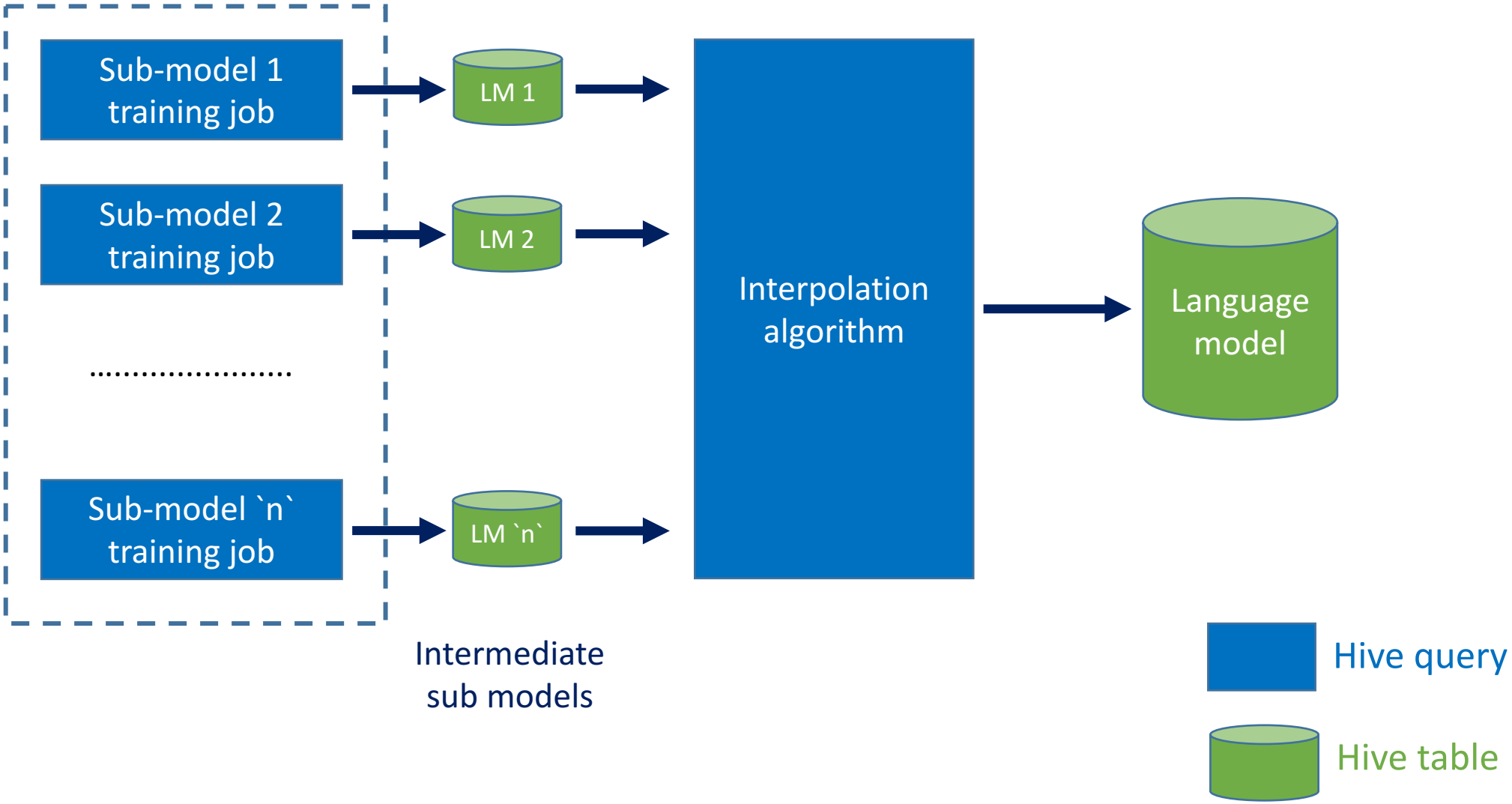
Auto-subtitling for Page videos

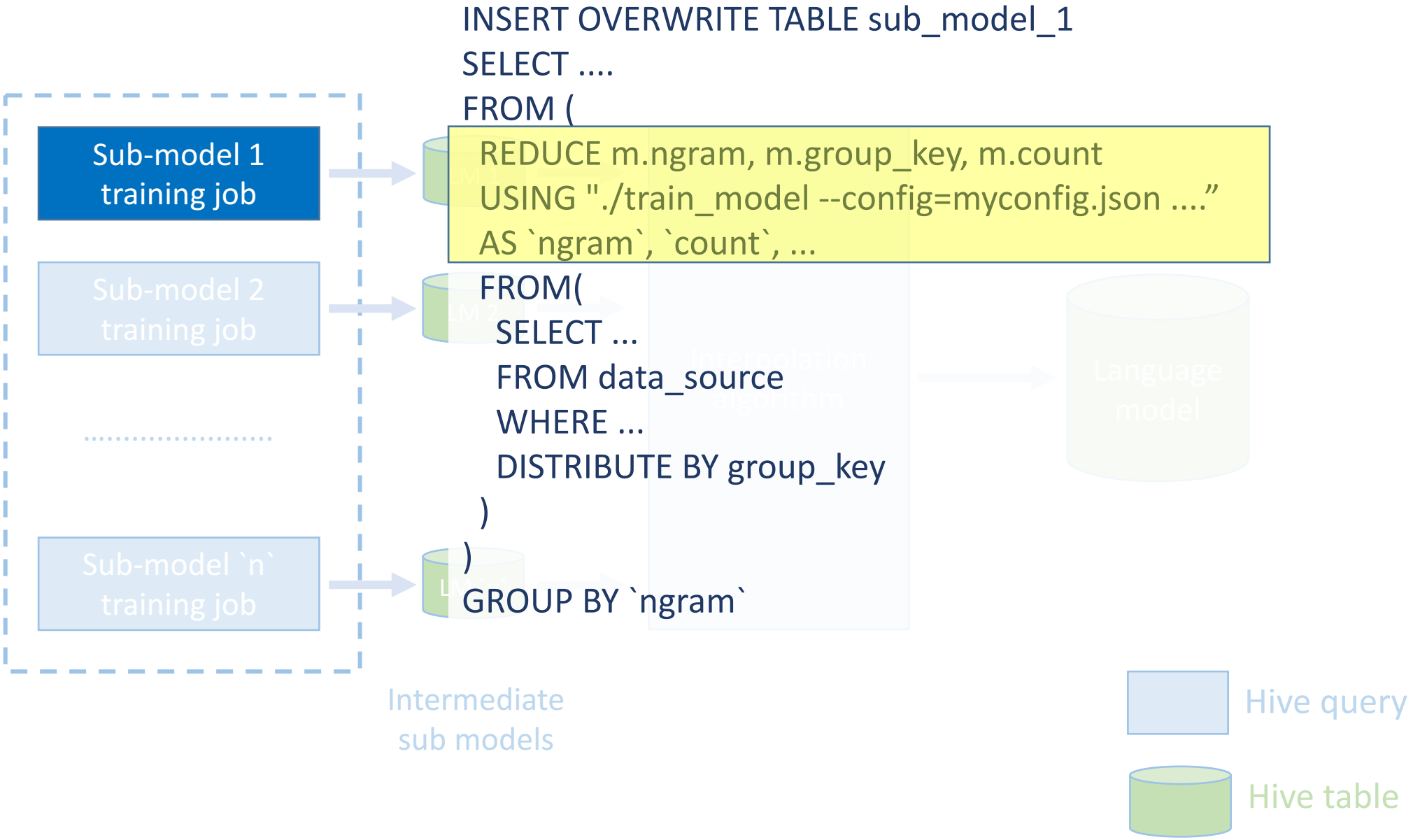


Detecting low quality places

- Non-public places
  - My home
  - Home sweet home
- Non-real places
  - Apt #00, Fake lane, Foo City, CA
  - Mordor, Westeros !!
- Non-suitable for watch
  - Anything containing nudity, intense sexuality, profanity or disturbing content

Previous solution


Sub-model 1 training job
Sub-model 2 training job
.....
Sub-model `n` training job
LM 1
LM 2
LM `n`
Interpolation algorithm
Language model
Intermediate sub models
Hive query
Hive table


Sub-model 1 training job
Sub-model 2 training job
.....
Sub-model `n` training job
INSERT OVERWRITE TABLE sub_model_1
SELECT ....
FROM (
REDUCE m.ngram, m.group_key, m.count
USING "./train_model --config=myconfig.json ...."
AS `ngram`, `count`, ...
FROM(
SELECT ...
FROM data_source
WHERE ...
DISTRIBUTE BY group_key
)
)
GROUP BY `ngram`
Intermediate sub models
Interpolation algorithm
Language model
Hive query
Hive table

Lessons learned

- SQL not good choice for building such applications
  - Duplication
  - Poor readability
  - Brittle, no testing
  - Alternatives
    - Map-reduce
    - Query templating
- Latency while training with large data

Spark solution

Spark solution

- Same high level architecture
  - Hive tables as final inputs and outputs
  - Same binaries used in Hive TRANSFORM
- RDD not Datasets
- `*pipe()*` operator
- Modular, readable, maintainable

Configuration

PipelineConfiguration

- where is the input data ?
- where to store final output ?
- spark specific configs:
  "spark.dynamicAllocation.maxExecutors”
  "spark.executor.memory”
  "spark.memory.storageFraction”
  ……
- list of **ComponentConfiguration**

……

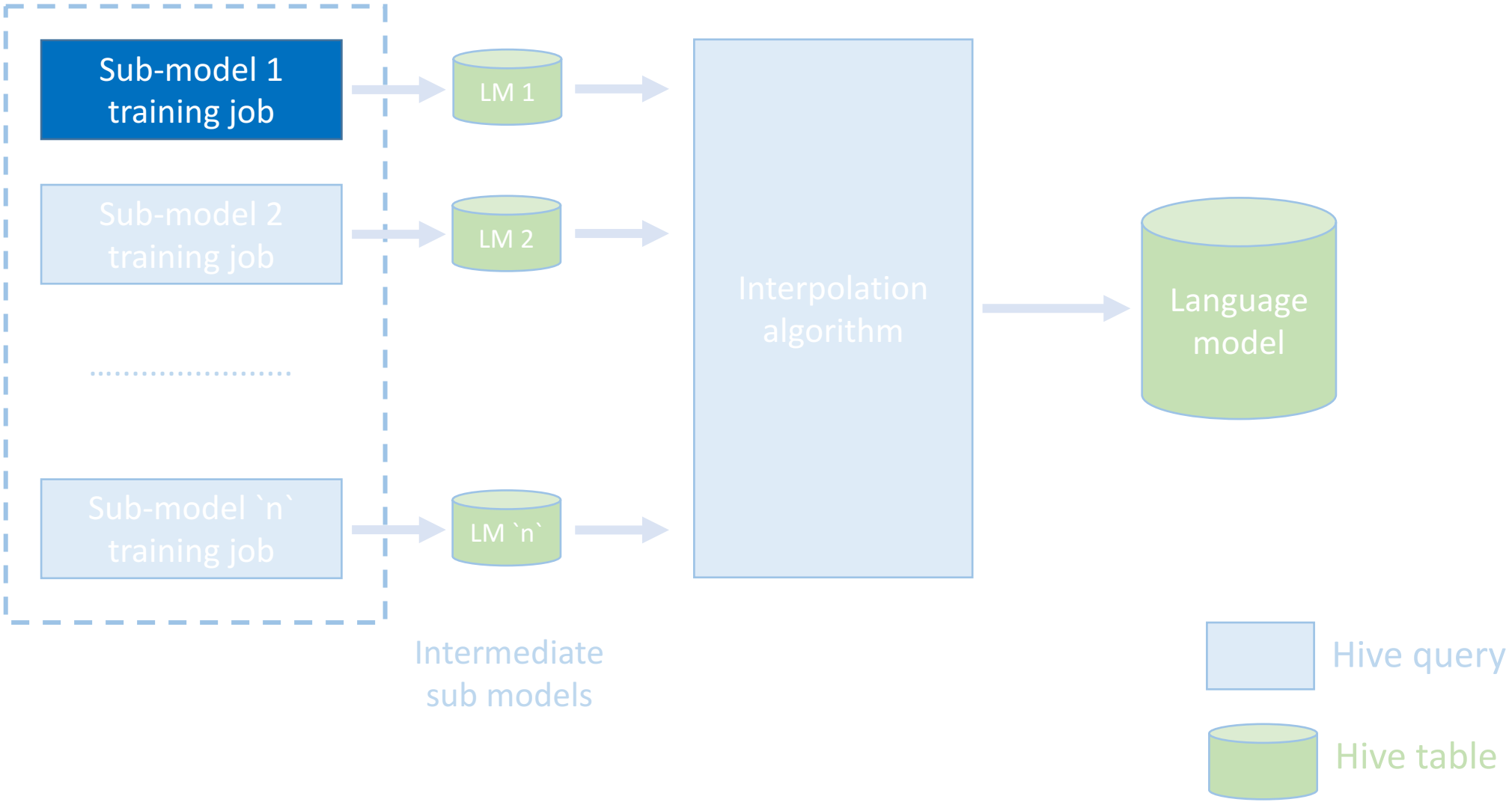
Scalability challenges

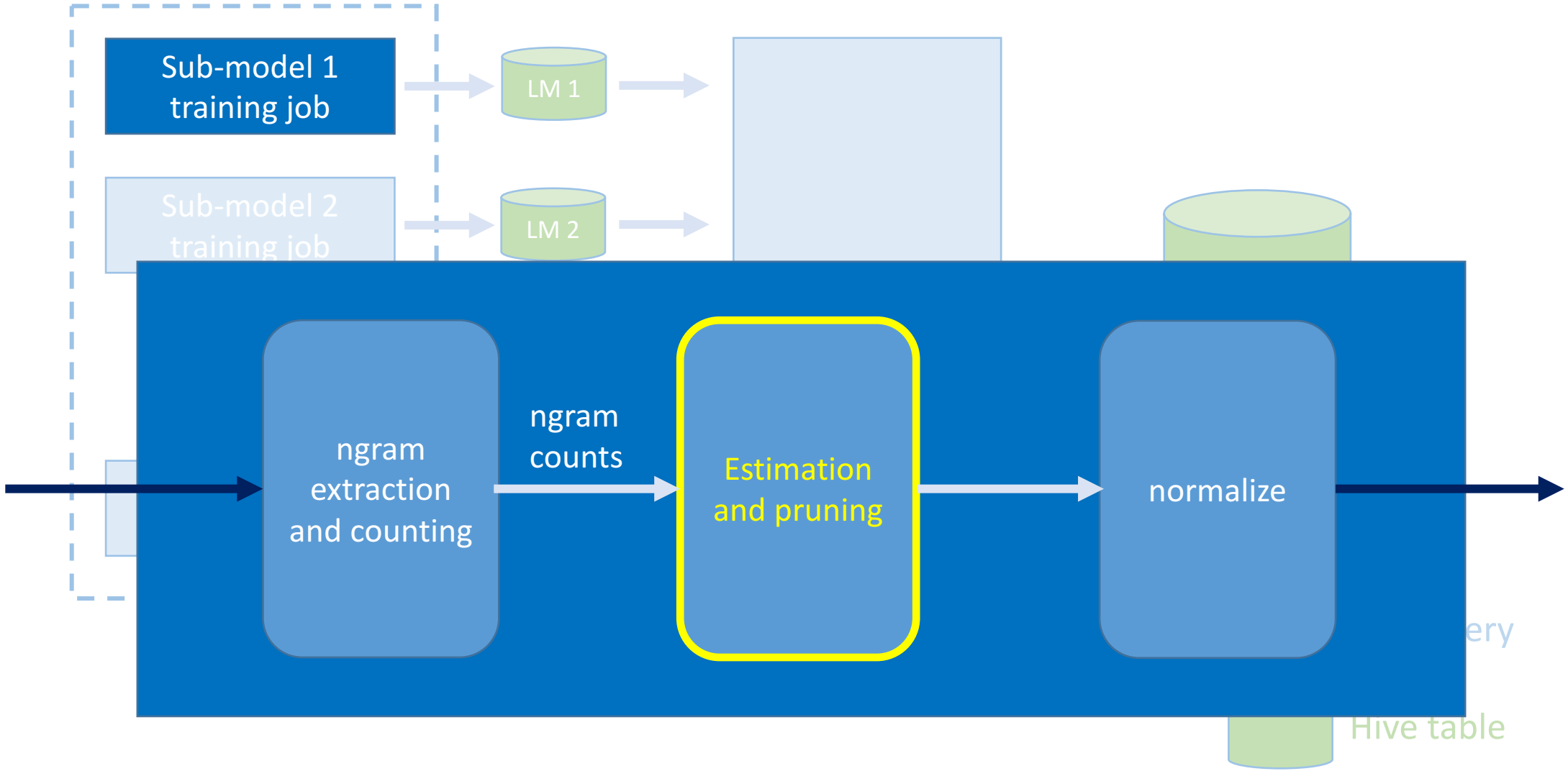
- Executors lost as unable to heartbeat
- Shuffle service OOM
- Frequent executor GC
- Executor OOM
- 2GB limit in Spark for blocks
- Exceptions while reading output stream of pipe process

Scalability challenges

- Executors lost as unable to heartbeat
- Shuffle service OOM
- Frequent executor GC
- Executor OOM
- 2GB limit in Spark for blocks
- Exceptions while reading output stream of pipe process

Data skew


Sub-model 1 training job
Sub-model 2 training job
.....
Sub-model `n` training job
LM 1
LM 2
LM `n`
Interpolation algorithm
Language model
Intermediate sub models
Hive query
Hive table


Sub-model 1
training job
Sub-model 2
training job
LM 1
LM 2
ngram
extraction
and counting
ngram
counts
Estimation
and pruning
normalize
ery
Hive table

Training dataset

How are you
How are they
Its raining
How are we going
When are we going
You are awesome
They are working

.....

.....

Word count

<How are we going> : 1

....

<How are you> : 1
<How are they> : 1

....

<How are> : 4
<You are> : 1
<Its raining> : 1

....

<are> : 6
<you> : 1
<How> : 4

.....

Word count

<How are we going> : 1

<are we going> : 2

<we going> : 2

<going> : 1

<When are we going> : 1

<Its raining> : 1

<You are awesome> : 1

.....

.....

Partition based on 2-word suffix

Word count

<How are we going> : 1
<are we going> : 2
<we going> : 2
<going> : 1
<When are we going> : 1
<Its raining> : 1
<You are awesome> : 1
.....
.....

<How are we going> : 1
<are we going> : 2
<we going> : 2
<When are we going> : 1
.....

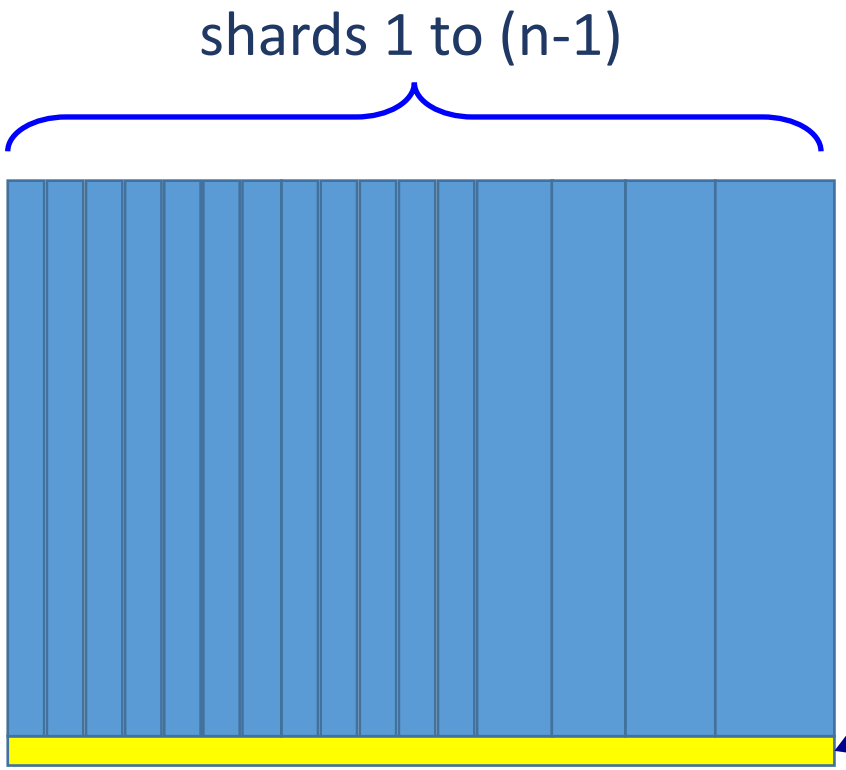
<Its raining> : 1
<You are awesome> : 1
.....

.....
.....

<are> : 6
<How> : 4
<you> : 1
<doing> : 1
<going> : 1
<awesome> : 1
<working> : 1
.....
.....

Frequency of every word:
0’th shard

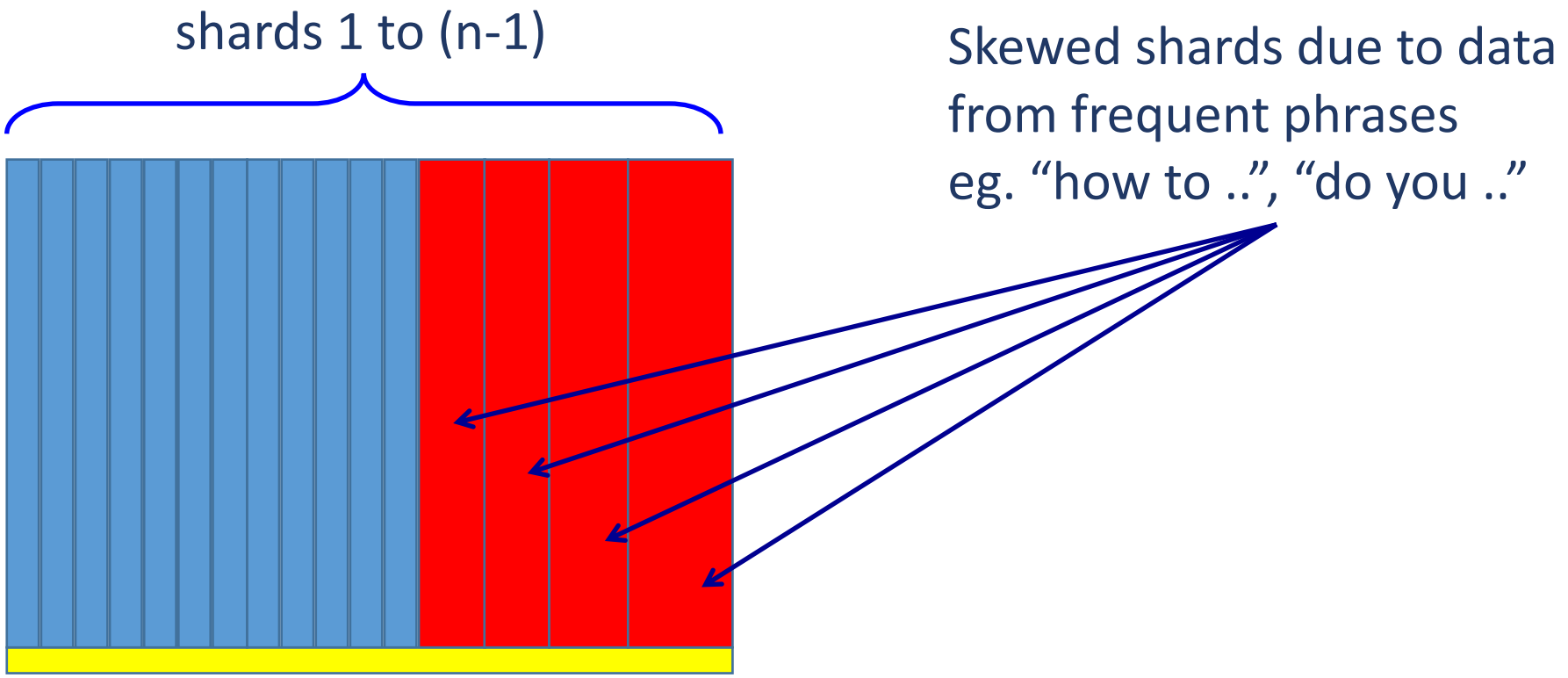
Distribution of shards (1-word sharding)



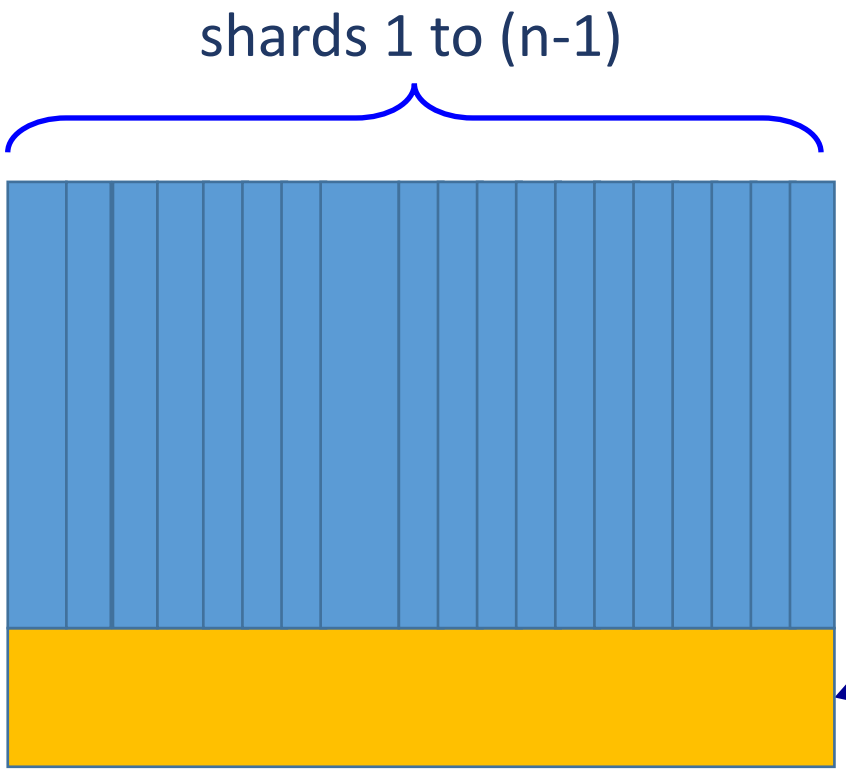
N-grams with same 2-word suffix will fall in the same shard

0-shard (has frequency of every word) and is shipped to all the nodes

Distribution of shards (1-word sharding)



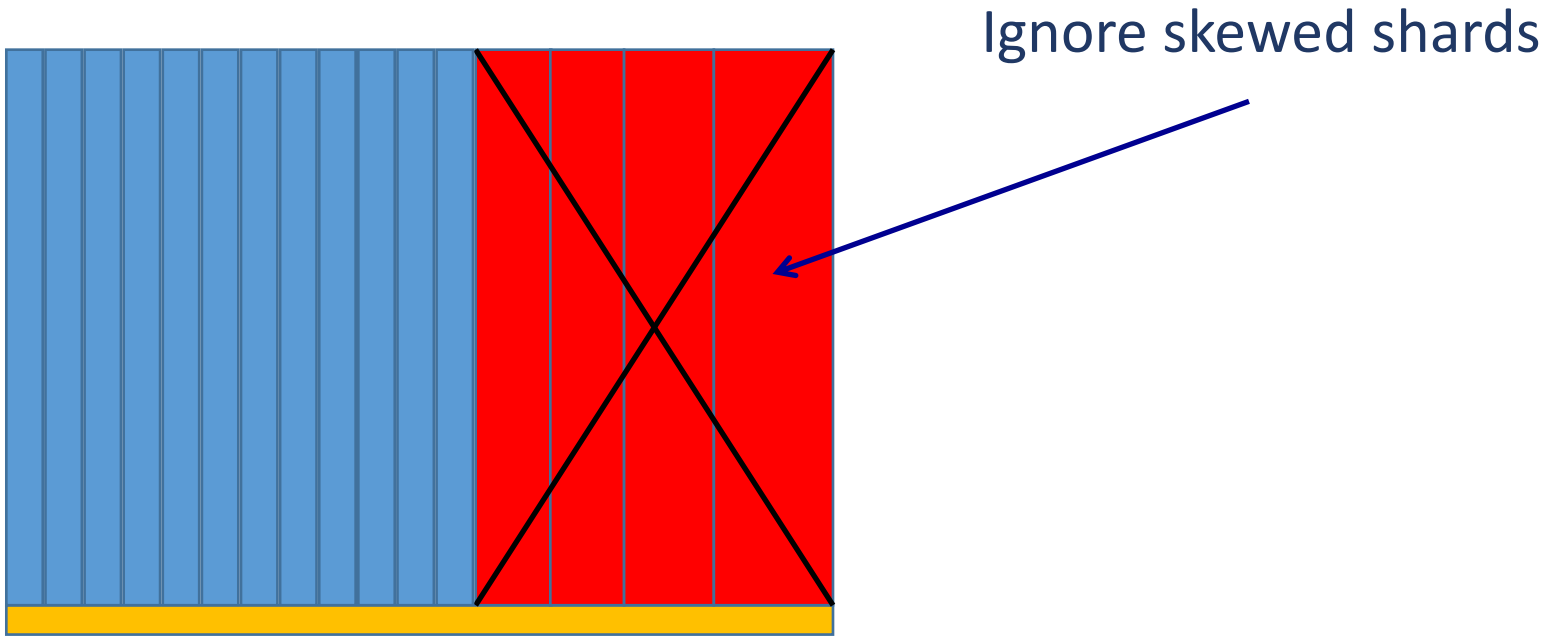
Distribution of shards (2-word sharding)



0-shard has single word frequencies and 2-word frequencies as well

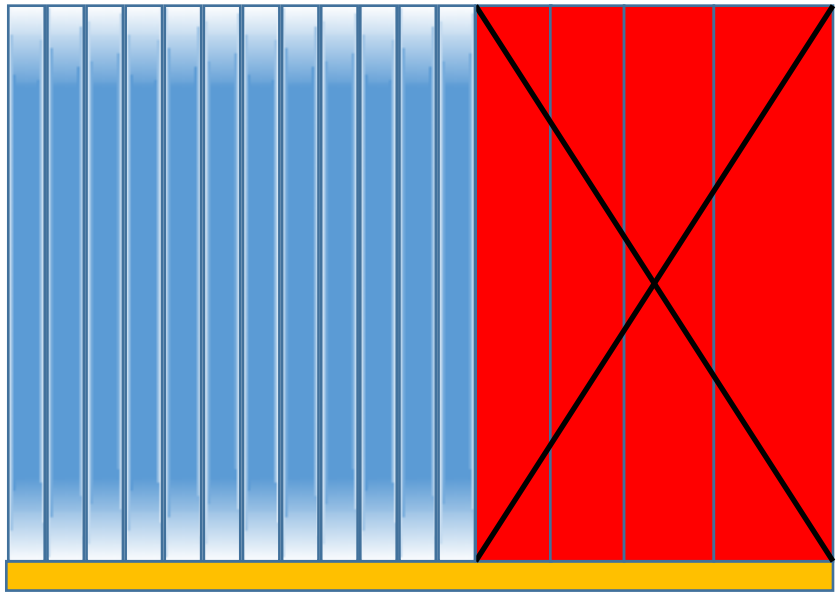
Solution: Progressive sharding

First iteration



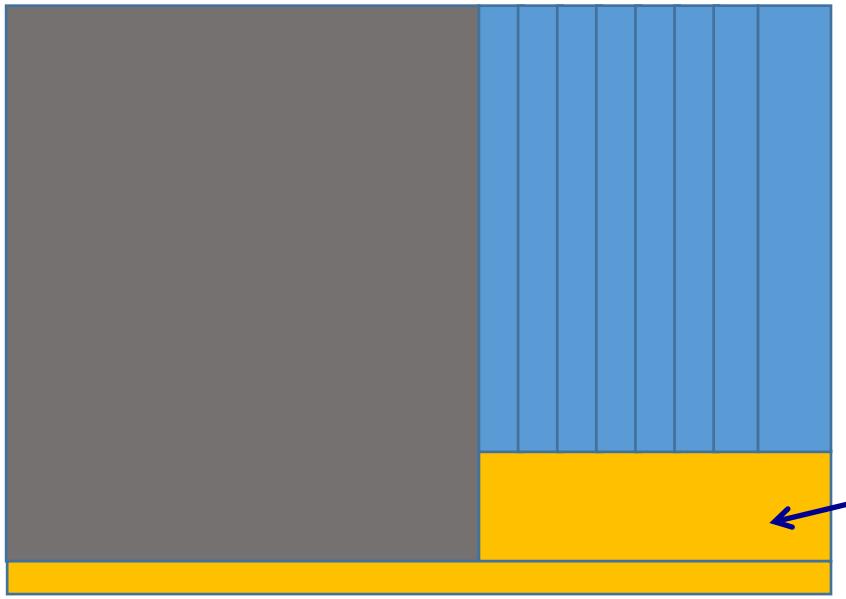
```
def findLargeShardIds(sc: SparkContext, threshold: Long, .....): Set[Int] = {
    val shardSizesRDD = sc.textFile(shardCountsFile)
        .map {
          case line =>
              val Array(indexStr, countStr) = line.split('\t')
             (indexStr.toInt, countStr.toLong)
        }
    val largeShardIds = shardSizesRDD.filter {
          case (index, count) => count > threshold
    }.map(_._1)
    .collect().toSet

    return largeShardIds
}
```

First iteration



Process all the non-skewed shards

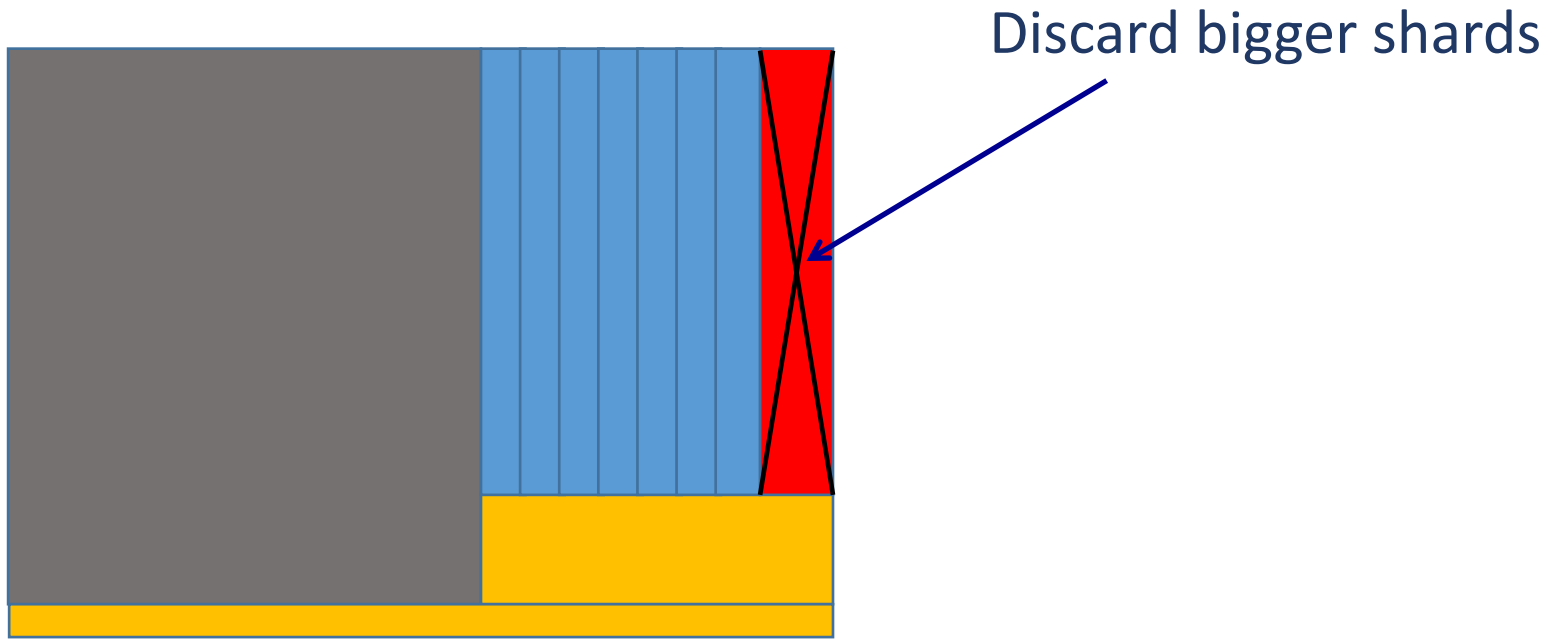
Second iteration



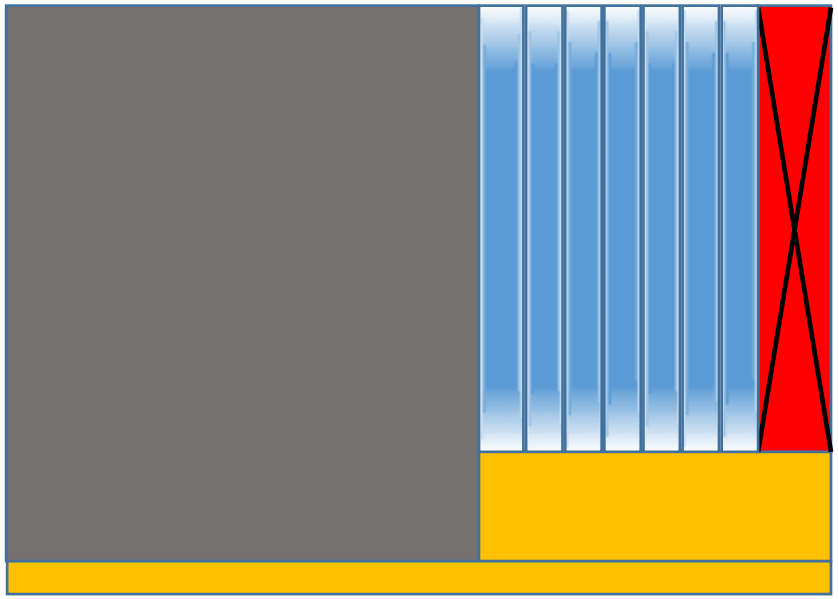
Effective 0-shard is small

Re-shard left over with 2-words history

Second iteration



Second iteration

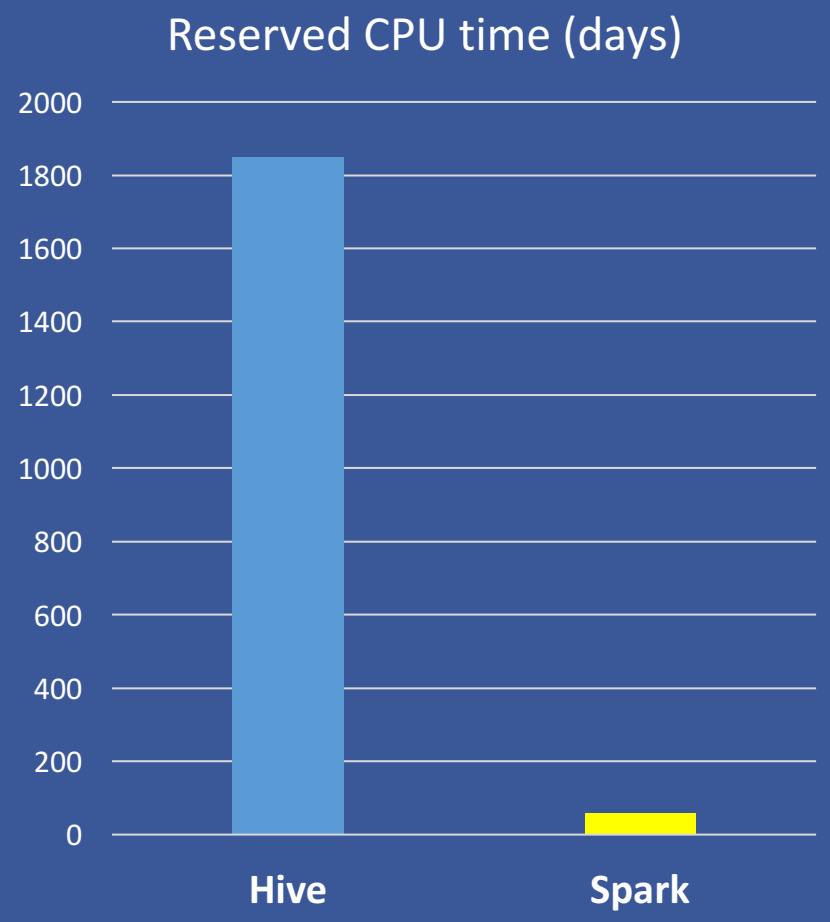


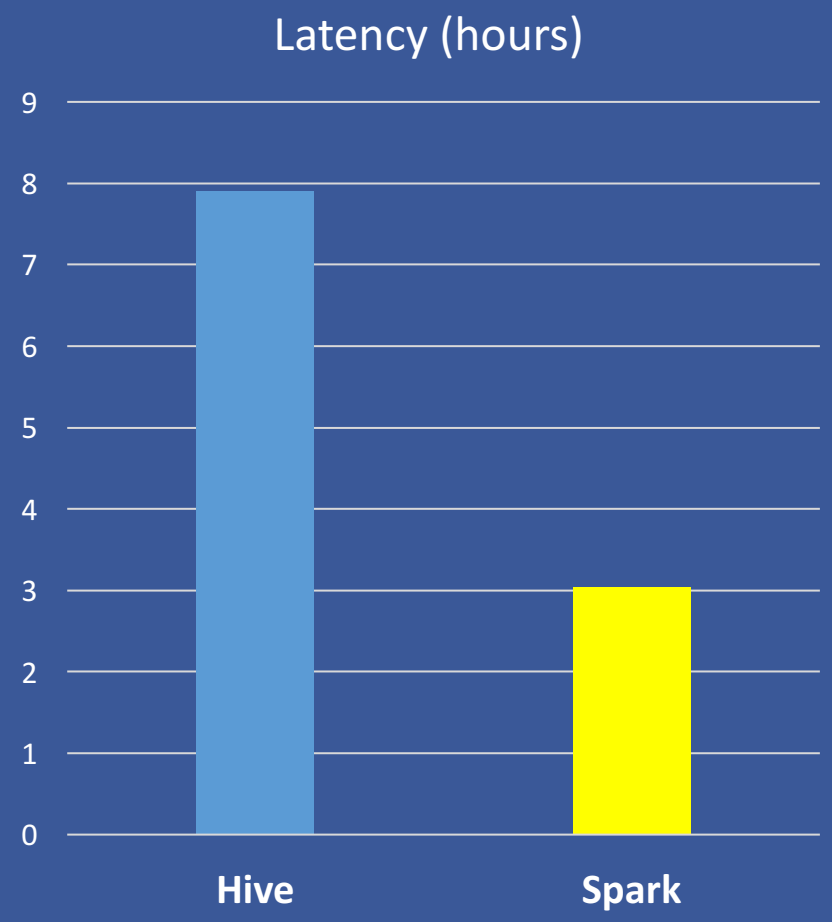
Process all the non-skewed shards

Continue with further iterations ….

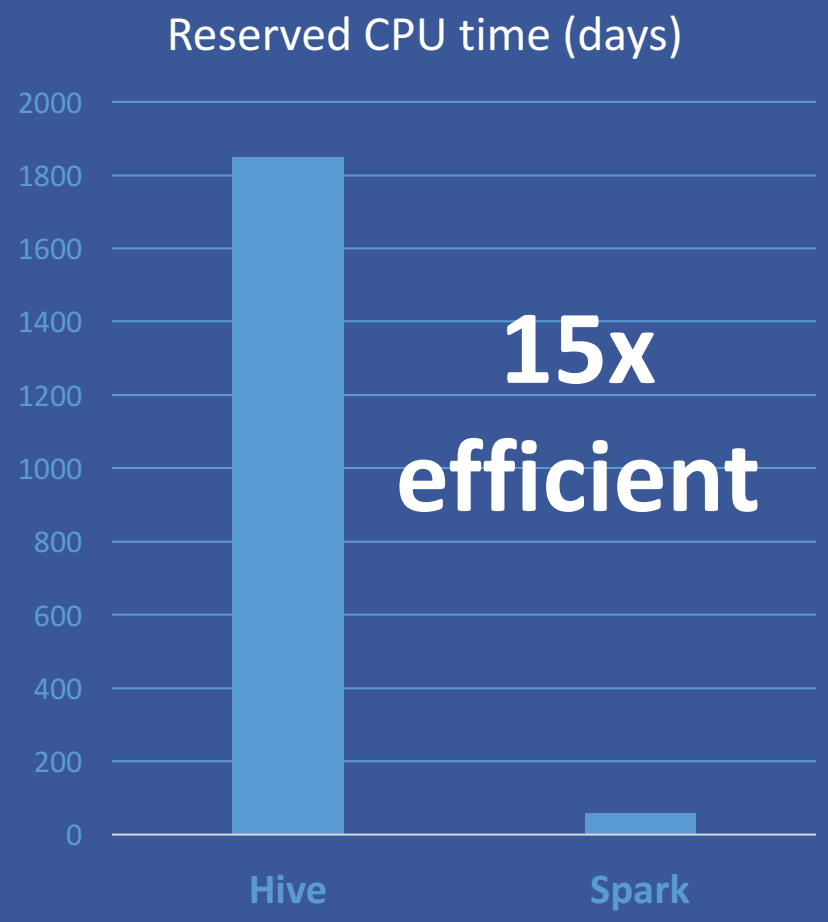
```
var iterationId = 0
do {
  val currentCounts: RDD[(String, Long)] = allCounts(iterationId - 1)
  val partitioner = new PartitionerForNgram(numShards, iterationId)

  val shardCountsFile = s"${shard_sizes}_$iterationId"
  currentCounts
    .map(ngram => (partitioner.getPartition(ngram._1), 1L))
    .reduceByKey(_ + _)
    .saveAsTextFile(shardCountsFile)

  largeShardIds = findLargeShardIds(sc, config.largeShardThreshold, shardCountsFile)
  trainer.trainedModel (currentCounts, component, largeShardIds)
        .saveAsObjectFile(s"${component.order}_$iterationId")
  iterationId + 1
} while (largeShards.nonEmpty)
```

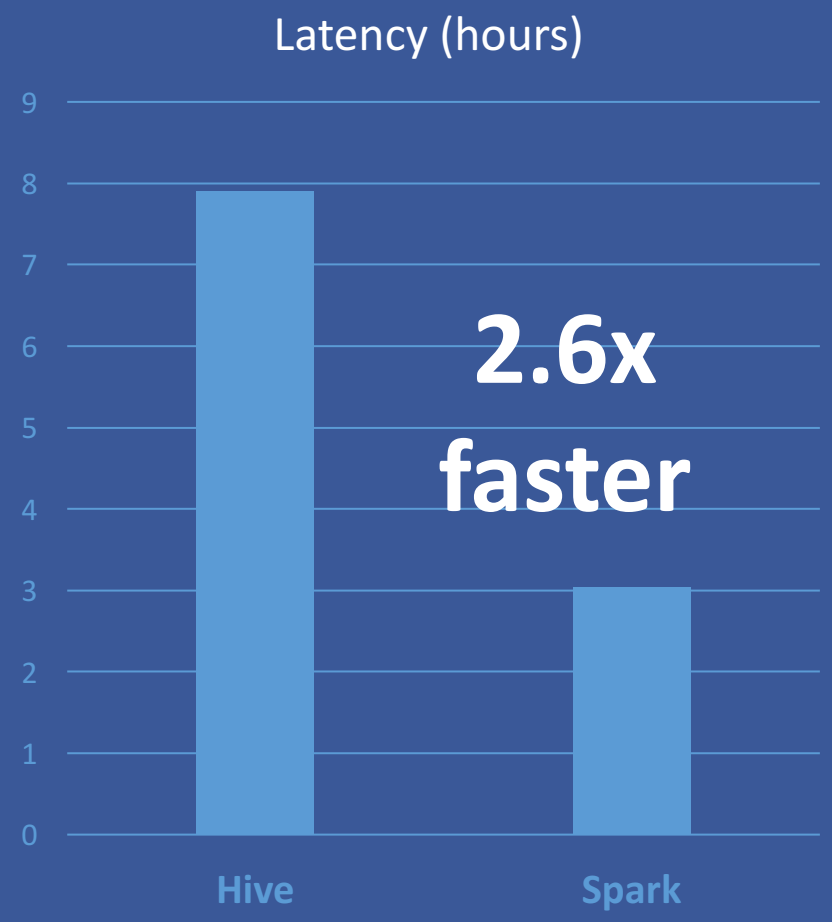
Performance evaluation





Performance evaluation





Upstream contributions to *pipe()*

- [SPARK-13793] PipedRDD doesn't propagate exceptions while reading parent RDD
- [SPARK-15826] PipedRDD to allow configurable char encoding
- [SPARK-14542] PipeRDD should allow configurable buffer size for the stdin writer
- [SPARK-14110] PipedRDD to print the command ran on non zero exit

Questions ?