Python - P30

弹性分布式数据集 (resilient distributed dataset, RDD)
- abstraction representing state of set of distributed data
  - specify the type of data represented by the dataset
  - constructed either from storage / memory / transformation
- resiliency : failure recovery capability
  - RDDs are computed as needed
  - if a compute worker fails,
    - another worker can be restarted to perform that part of computation
  - if a machine holding a cached portion of a persistent RDD fails
    - can recompute the RDD

转换 (transformation), perform operation on data elements of RDD, produce new RDD
- transformation do not modify the source RDD
  - the return type is some form of RDD
- lazy evaluation : transformation is not evaluated when defined
  - resulting RDD exists as abstraction, not complete dataset
- lineage graph : chain or graph of RDD,
- defined by Spark application, perform only when action invoked on RDD
  - propagated backwards along lineage graph
    - initiating forward execution of transformation

single RDD transformation
- filter : contain only item from source RDD which pass a Boolean test
- map : apply a function to each item of source RDD generating single item
- flatMap : apply a function to each item of source RDD generating multiple items
- reduceByKey : combine all items with the same key from source RDD into single item
- sample : statistical sample of the source RDD data
- distinct : only single copy of any duplicates as the source RDD

multiple RDD transformation
- union : contain all item from multiple source RDDs
- intersection : contain items occuring in all source RDDs
- subtract : remove items in 2nd RDD from 1st RPD
- cartesian : generate Cartesian product from 1st RDD and 2nd RDD

Python - P31

流水线 (pipelining), computation of chained RDDs done in pipelined manner
- normally transformation computed on one data item at one time
- result passed the next RDD transformation when possible
- transformation may be done in parallel on multiple machines
- parallelism achieved within one RDD, over multiple RDDs in pipeline
- if intermediate RDD is to be reused
  - can be marked as persistent (永续的)
  - and cached in memory and/or disk (no default)

distributed attribute of RDD : effectively apply to space and time

行动 (action), compute a result and return it to caller or write to storage

saveAsText	: write RDD to storage
collect	: return entire source RDD in memory
reduce	: combine all the element of source RDD into single item
fold	: combine all elements with identity element specified
aggregate	: two stage merge operation, possibly different types
count	: count the elements in source RDD
take	: return specified number of elements from source RDD
takeOrdered	: 与take类似, with order specified
takeSample	: return a random sample of source RDD
forEach	: apply function to each element of source RDD

compute cluster overview : two main pieces in typical compute cluster
- distributed file system : HDFS for hadoop
  - file may be accessed from any node in the cluster
    - need special procedure to access, not accessible from normal Linux interface
  - not a shared file system like NFS
- job scheduler : yarn for hadoop
  - spark-submit --master yarn --deploy-mode cluster <parameters>
  - job running on cluster may not have access to Linux FS file
  - all data used by job running under yarn should be stored in HDFS
  - all output generated by job should be generated in HDFS

Python - P32

pair

transformation : flatMapValues(func)
keys()
values()
sortByKey()

pair-pair transformation : subtractByKey
join
rightOuterJoin
leftOuterJoin
cogroup

action : countByKey()
collectAsMap()
lookup(key)

partition

logical grouping of data within an RDD in Spark
each partition reside in single worker node
can be distributed across multiple workers
may be created implicitly or explicitly

RDD loaded from file partitioned by file
and by data block in file (128MB of data by default)
caller can override number of partition
Splittable : file can generate multiple partitions
record boundary found from random access into middle of file
text file with one record per line
Hadoop Sequence file
Spark Object file

can provide number of partitions when executing parallelize
sc.parallelize(object, num)
default number for system if not specified

rdd.getNumPartitions() : number of partitions in RDD
rdd.partitioner() : current partitioner of RDD
rdd.glom() : determine elements in each partition

Python - P33

partitionBy() explicitly partition pair RDD by key
specify the number of partitions in the new RDD

specify existing partitioner / define own partitioner
hash partitioner : hash the key to select a partition (default)
range partitioner : use ordering of key to select partition
custom partitioner : can do anything

function process items with same key necessarily partition by key
reduceByKey / groupByKey / combineByKey / join, etc.
if input RDD has a key partitioning, use the same partitioning
otherwise, implicit partitioning by hash function performed

map() for non-pair RDD keep same partition
map() for pair RDD generate new key in the same partition
RDD no longer considered partitioned by key
mapValues() map on the value only, generate new value for the same key
result RDD retain the original partitioning if source partitioned by key

sometime need to perform high-overhead operation during mapping
[redacted] could be reused for multiple items
database access (can often batch request)
random number generator
network connection

partition mapping function call the mapper once per partition
input : iterator over all the items
output : iterator for the [redacted] map output

most normal mapping type available within partition
mapPartitions : used instead of map / flatMap
mapping function : FlatMapFunction returning<Iterator>

Python - P34

mapPartionsWithIndex : like mapPartitions
also pass partition number to mapper

mapPartionsToPair : used instead of mapToPair/flatMapToPair
mapping function : PairFlatMapFunction returning <Iterator>
注意 mapToPair 在 Python Spark 中不存在

forEachPartition : pass Iterator of partition to function
function may perform any operation on items using <Iterator>
function not return any value

accumulator , allow any worker to add to value
write-only : only driver program can read
not update until one action occur
useful for : debugging / measuring input quality / error counting
part of transformation may be re-run in case of failure
increment to accumulator may be performed more times than expected
usually correctly, few edge cases may cause over-accumulation

broadcast sometimes necessary to send constant value to all functions on all workers
Spark provide wrapper for shared, unchanging data
variable can be any serializable / pickled type
broadcast variable object : handle reference the value
pass broadcast variable instead of actual data
only one copy of data content shipped to each worker
referencing method .value() (.value) to access the single copy
treated as read-only in worker

SparkConf conf.set(property, value) allow setting property
can set through spark-submit command
use --conf property = value
through property config file --properties-file
shortcut available for some properties

Python - P35

common Spark configuration properties

property name	default	description
spark.executor.memory	512MB	memory for each executor process
spark.executor.cores	1	number of CPU cores per executor
spark.speculation	false	true to enable speculative execution
spark.executor.extraJavaOptions	[empty]	JVM options
spark.executor.extraClassPath	[empty]	add extra jars to CLASSPATH
spark.serializer	[Java]	specify faster serializer

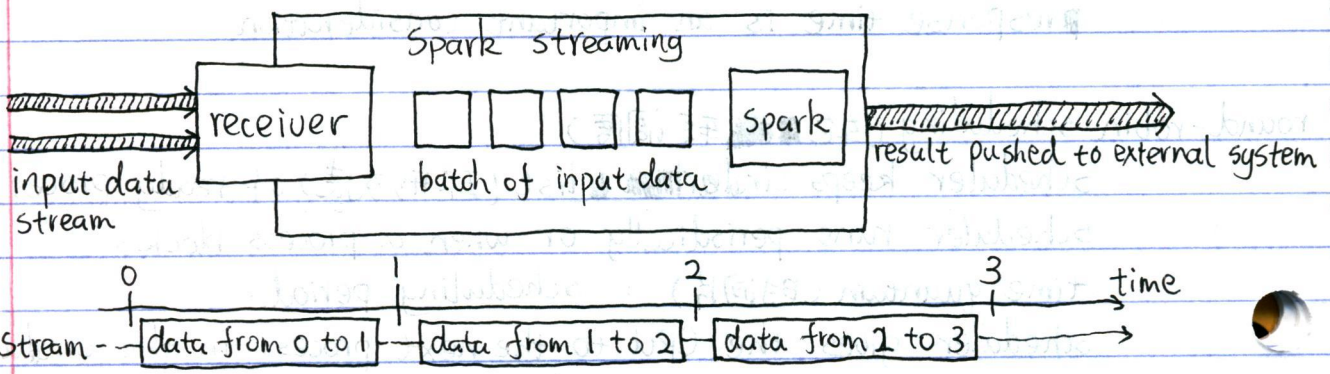
execution component : job : compute all needed RDDs
stage : grouped parallel waves of computation
computation of one or more RDD transformation
generally bounded by action/transformation requiring shuffle
task : compute segment of one or more RDDs within stage

controlling level of parallelism
factors : number of executors : specified on the spark-submit command
number of partitions : can leave to Spark to choose
most operations that can repartition
allow caller to optionally specify number of ouput partitions
coalesce and repartition just change the partitioning
coalesce more efficient, not shuffle, only reduce partition
combining/filtering : leave partitions too small
flatMap : make partitions too large

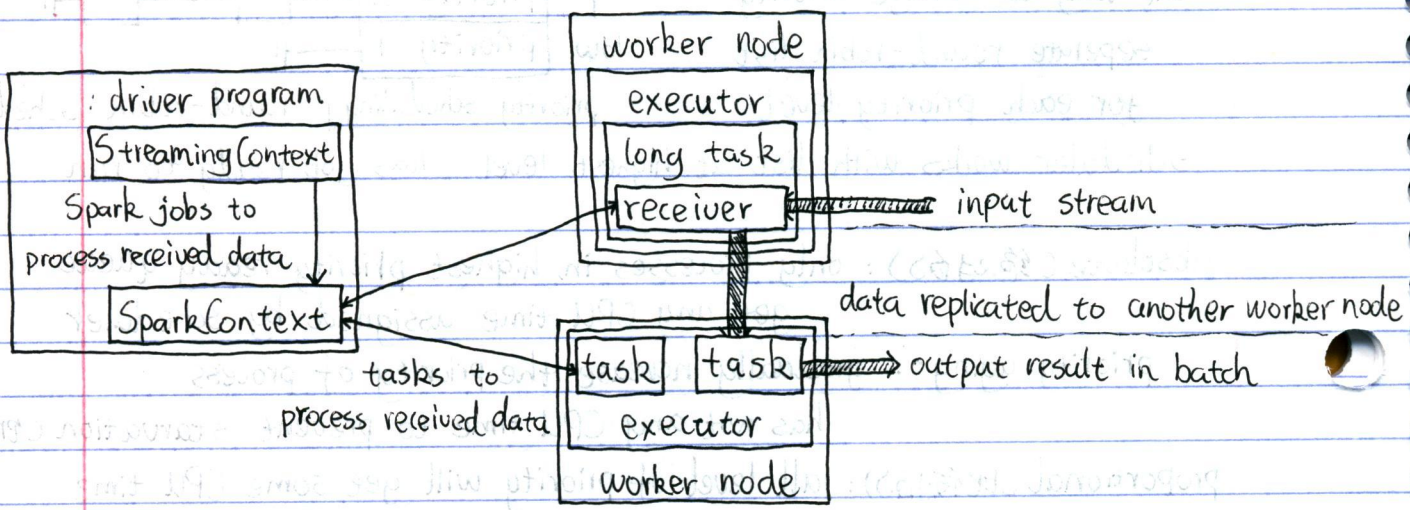
serialization efficiency : object serialized when moved over network or written to disk
needed during shuffle operations
built-in Java serialization inefficient
Kryo serialization : provided by Spark, more efficient
class to be serialized is registered with Kryo
set property : cause Kryo to throw an exception
when encounter non-registered class

Python - P36

spark streaming : allow Spark job to process data incrementally
arrive from : network connection / file / other software subsystem
Spark DStream monitor data source
at specified interval, data arrived since last interval processed
data may be integrated in sliding window
over multiple intervals
DStream effectively create RDD of data arrived during one interval
and perform specified processing as separate job



component
receiver : get data from network stream, file directory
data replicated to another worker node
need to have enough thread to handle extra receiver
transform : DStream input → DStream / RDD output
DStream as continuing series of RDD
output operation : similar to RDD action, but run periodically each time step



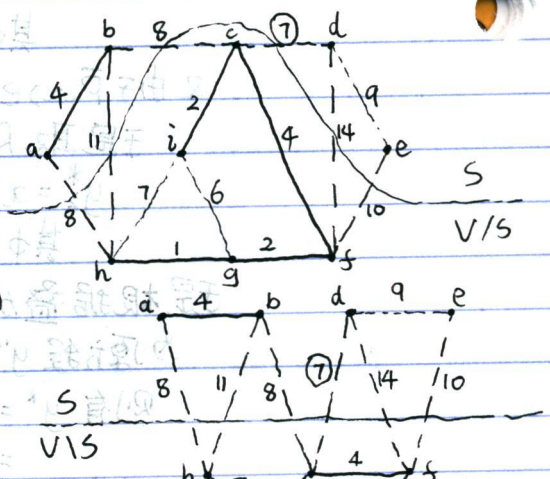
Python - P37

GraphFrame: provide graph representation using Spark DataFrame
Graph: specialized class contain DataFrame of vertex and edge
vertex: consist of DataFrame with column "id" and vertex data
edge: specified DataFrame with column "src" and "dst" in addition to other edge data

can use DataFrame operation on Graph
can get input/output degree for each vertex
can use Motifs to patteran match path

Pregel implementation in GraphFrame
only allow mutation to vertex data, not to edge data
not allow change to edge or edge value
send message by edge
can use different strategy to find spanning tree

最小生成树(minimum weight spanning tree, MST)
GENERIC-MST(G, w)
A = ∅
while A does not form spanning tree
(hard) find one safe edge (u,v) for A
A = A ∪ {(u,v)}
return A



割(cut), 对于图G(V,E), 有对于点集V的划分(S, V\S)
称边跨越(cross)割, 当其端点分别在S和V\S中
light edge: edge cross cut with smallest weight of any edge cross the cut
称割respect边集A, 当A中没有跨越割的边

对于连通无向权重图 G(V,E)
给定边集 A ⊆ E, 并且A是图G的某个MST的子集
令割(S, V\S) respect A
令(u,v)为跨越(S, V\S)的light edge
则有(u,v) is safe edge for A