Parallel and Distributed Computing - P12

fault tolerance by checkpoint / restore (GFS/HDFS)
master periodically request all workers to save state to persistent storage
on failure of any worker, master instructs all workers
to restore from latest checkpoint
may be several supersteps back
message logging also used for confined recovery
can recover single worker by rerunning messages
require deterministic algorithm
can be disabled for truly random
as opposed to pseudo-random, computations

worker implementation, worker keep partion state in memory
State of each vertex: vertex data (may be structured data)
list of outgoing edges (each edge has target and value)
incoming message queues for current and next superstep
vertex active flag
on superstep, worker call compute method for each vertex

message processing, each worker buffers outgoing messages for other workers
multiple messages to the same worker are bundled
send as a single transmission
combiner applied to outgoing messages for the same vertex
local messages placed directly in receiving vertex's incoming message queue
for the next super step

master implementation, master keep list of all workers
master maintain barrier between supersteps
initiate superstep by message to all workers
wait for all workers to indicate superstep completion
request worker to write checkpoint at periodic interval
collect and combine aggregator value (tree structured collection)
master maintain statistic about the computation
progress may be viewed on web page served by the master

Parallel and Distributed Computing - P13

Graph X

provide graph representation using spark RDD

Graph : specialized class contains VertexRDD and EdgeRDD class

vertex : consist of ID :: long and vertex data :: user-defined serializable data abbreviated <VD> in documentation

edge : specified by pair of vertex ID, and edge data :: user-defined serializable data abbreviated <ED> in documentation class

message : contain data :: user-defined serializable class
data abbreviated <A> in documentation

RDD is immutable, each step of graph algorithm must create new RDD
GraphX optimizes by reusing unchanged data from one RDD to the next

GraphX written by Scala
unlike core Spark code, GraphX don't have specialized class and method for Java
can be used from Java, but require use of specific Scala class

GraphLoader.edgeListFile : one line per edge, no vertex/edge value specified
<VD>, <ED> default to <Integer> with default value

Graph.fromEdges : input RDD<Edge<ED>>
pair of vertex ID and each value for each edge
allow specification of default vertex type and value

Graph.fromEdgeTuples : input RDD<scala.Tuple2<object, object>>
allow specification of default vertex type and value
Edge<ED> default to <Integer>

operation available on Graph to generate new Graph

mapEdges : run transformation function on each edge, create new value for edge
useful to initialize edge data after using GraphLoader

mapVertices : run transformation function on each vertex, create new value for vertex
useful to initialize vertex data after using GraphLoader

mapTriplets : run transformation function on each edge and its vertices
create new value for each edge

subgraph : select subgraph edges and vertices
based on predicate evaluated on each EdgeTriplet

Parallel and Distributed Computing - P14

GraphOps
get from Graph.ops() , contain useful algorithm
filter , collect algorithm , connected component
page rank , statistics gathering , pregel

pregel in GraphX , GraphX implement simplified version of Pregel
other operations used to set up Graph before running Pregel
or to extract result after running Pregel
on initialization , same initial message sent to all vertices
on each superstep :
all incomming messages to one vertex are merged
vertex message reader called for each vertex
send-message routine called for each edge
has active source vertex , active destination vertex , either or both
depending on caller specification
on exit , resulting Graph object has final vertex and edge RDDs

Breadth-First Search spanning tree algorithm
can be easily implemented in parallel using system like Pregel
each vertex value is <boolean>
true : part of the spanning tree
all vertex values except the root initialized to false
undirected graph represented by directed edge pairs
message content : vertex ID
receiving message indicate the directed edge to the vertex should be deleted
at termination, spanning tree constructed of directed edge

initialization : at superstep 0
root vertex : set value to ture
send message with vertex ID to all connected vertices
vote to halt
other vertices : set value to false
vote to halt

Parallel and Distributed Computing - P15

BFS spanning tree, at superstep >0, for every vertex:
- vertex value : false (not yet in spanning tree)
  - if any incoming message (connecting to spanning tree)
    - set vertex value to true
    - for each message, delete edge back to vertex ID in message
      - for all but first message
        - send vertex ID back to originating vertex
    - for each remaining edge, send vertex ID to edge target
  - else do nothing
- vertex value : true (already in spanning tree)
  - for each incoming message for which have edge back
    - send vertex ID back to originating vertex
    - delete edge back to vertex ID
- vote to halt (always)

Pregel implementation in GraphX
- only allow mutation to vertex data, not to edge data
- original Pregel algorithm require modifying tree structure by deleting edge
  - GraphX Pregel not allow change to edge or edge value
- original Pregel send message by vertex
  - GraphX Pregel send message be edge

BFS in GraphX
- each vertex value type <Long>
  - set to VERTEX_UNVISITED if not yet connected to tree
  - otherwise set to vertex ID on edge connecting to tree
  - root vertex set to own vertex ID
- undirected graph represented : EdgeDirection.Either()
- message content : vertex ID
  - receiving message in which vertex ID connect to the tree
- at termination, (vertex ID, data) define edges of spanning tree

on initialization : root vertex : set value to own vertex ID
other vertex : set to unique value VERTEX_UNVISITED (Long.MIN_VALUE)

Parallel and Distributed Computing - P16

BFS in GraphX, at each call sendMsg, for edge
- one vertex in the tree (value != VERTEX_UNVISITED)
- one vertex not in the tree (value == VERTEX_UNVISITED)
- send message to unvisited vertex with tree vertex ID
- ignore edge both vertices in tree or both not in tree

mergeMsg routine simply pick vertex ID to connect to the tree
- pick the smaller ID for reproducibility

vprog (receive message) routine simply update vertex value with ID

at termination, write out pair of (vertex ID, vertex value)
- each pair define one edge in spanning tree
- ignore pair from root to itself

difference between running Spark locally / in the cluster (on Yarn)
- can cause program run fine locally, fail when running on Yarn

Java program may use outer class member variable
- to hold value accessed by any class member function

locally: Spark program run in single process
- any of outer class variable available to all code

on cluster: initialization code run only on Spark master process
- computation code run on worker processes
- function supply to RDD transformation and action
  - run on worker processes
  - function code class serialized by master (from main)
    - sent to worker to execute (not class initialization code
  - outer class variable set by initialization code only available on master
  - data item used in function supplied Spark transformation / action
    - must contained within function class
    - always pass needed data into function class through constructor

Parallel and Distributed Computing - P17

Pipe transformation : allow external program to be used for transform
current RDD written to standard input of command
standard output of command form new JavaRDD<String>
one item per line of output
external program must be available to run on any worker node
SparkContext.addFile(path) : specify file copied to each worker node
SparkFiles.getRootDirectory / SparkFiles.get(filename)
worker find the file

numeric RDD JavaDoubleRDD data type : similar to JavaRDD<double>
provide additional capability of built-in statistics computation
stats() : compute variety of statistics in single pass over RDD
count, min, max, sum, mean, variance
sampleVariance, stdev, sampleStdev.
return StatsCounter object
method return various precomputed statistics

configuration option precedence order
override
- option specified directly in program code
- spark-submit option
- property file option
- default option

Spark streaming, input from socket : can connect to specified host and socket number
基本语法为, JavaDStream<String> scoreStream =
jsc.socketTextStream(streamHost, streamPort);
will read data from the socket
at specified interval batch, data into DStream and initiate processing
will continue running even if successful connection cannot be made

other subsystem may supply custom module to interface with Spark
Flume, Kafka : somewhat limited through Python
custom loader can be written to read in any source
not supported yet in Python

Parallel and Distributed Computing - P18

streaming

input from file : can take input from file moved into directory
usually shared file system like HDFS
can also be directory on local file system in local mode
基本语法为 JavaDStream<String> fileStream =
jsc.textFileStream(dirPath);
file must be created elsewhere and moved into directory
otherwise Spark may start processing file whill still being written
file may have any format recognized by Spark
use fileStream instead of textFileStream

streaming transformation : stateless , processing of each batch not depend on other batch
include common RDD transformation (map / filter / reduceByKey)
stateful , use intermediate data or results from prior batch
sliding window transformation
saved state transformation

stateless transformation : applied independently to each batch
can have multiple DStreams in each batch
DStreams can interact using union / join / etc.
DStream transformation is subset of RDD transformation

stateful transformation : track across time
windowed transform : act over sliding window of time
state by key transform : maintain state value for each key
require checkpointing for fault tolerance

windowed transformation
each time step process data from current and one or more previous step
batch interval : time step
window duration : period to process at each time step (> batch interval)
sliding interval : how often to compute over window (≥ batch interval)
基本语法为 JavaDStream<T> windowedDS =
ds.window(Durations.seconds(int), Durations.seconds(int));

Parallel and Distributed Computing - P19

special windowed reduce operation, can use any normal RDD / DStream operation
- Spark provide incremental reduce operation
  - reduceByWindow / reduceByKeyAndWindow
  - require providing inverse for the reduce operator
  - cache partial result from prior window
    - to avoid recomputing the entire reduce

State by key transformation, keep DStream of <key, state> pair over time
- updateStateByKey : update DStream at any time step
  - apply data from time step DStream data
    - to state with user defined function

streaming output operation ; similar to output operation available for regular RDD
- saveAsTextFiles / saveAsHadoopFiles / etc.
- main difference : each time step create new subdirectory for saved file

RDD access foreachRDD : for other output by DStream
- or to access additional method not provided directly
- execute user-defined code each time step on RDD for that time step

wholeTextFiles, 与textFile相比，生成一个JavaPairRDD<String, String>
- key : file name , value : complete content of file
- should normally be used only with relatively small file
- large file may be too large for memory
- useful for application like indexing

binaryRecords, 与textFile相似，生成一个JavaRDD<byte[]>
- each byte array is single record
- specify fixed-size record read from raw binary file

binaryFiles, 与wholeTextFiles相似，生成一个JavaPairRDD<String, PortableDataStream>
- key: file name , value : stream
- allow reading of binary data as primitive type
- entire file not loaded into memory

Parallel and Distributed Computing - P20

HBase individual row operation : most efficient method for using HBase from Spark

Get used to specify data to be read
- constructor specify row key (byte array)
- default to get everything in specified row
  - addFamily : get all columns from specific family
  - addColumn : get specific column
  - setTimeRange : only retrieve column with specific range of version (timestamp)
  - setTimeStamp : only retrieve column with specific timestamp
  - setMaxVersions : limit number of version of each column
  - setFilter : add a filter

用于函数 Table.get :: Get -> Result / List<Get> -> Result[]

Put used to write individual row, like in Spark
- constructor specify row key (byte array)
- 与 Get 类似，通过 addColumn 添加列的信息

用于函数 Table.put :: Put / List<Put>

Delete used to delete specified data
- constructor argument : row_id (byte array)
- default to delete entire row
- can specify timestamp to delete all version ≤ specified timestamp value
  - addFamily : delete all columns in specified family
  - addColumns : delete all versions of specific column within family
  - addColumn : delete latest version / specified version of specific column within family

用于函数 Table.delete :: Delete / List<Delete>

RowMutations can atomically perform sequence of Put and Delete operation
- constructor argument : row_id (byte array)
- add :: Put/Delete : specify order of Put and Delete operation

用于函数 Table.mutateRow :: RowMutations