Spark: Cluster Computing with Working Sets

wbg231

January 2023

1 abstract

- map reduce is build in an acyclic data flow, which limits the type of programs it can be used for
- spark allows for iterative algorithms while still having fault tolerance
- this is done with a spark RDD which is a new abstraction that is a read only collection of objects across a set of machines, that can be rebuilt if a partition is lost.

Introduction

- map reduce does not allow iterative algorithms because its speed and fault tolerance come from restricting to map and reduce functions
- this frame work does not work for iterative jobs like many ml algorithms
- also not suited for interactive analytics, because hadoop treats each sql query as a map reduce functions and thus must read data from disk at each call.
- spark supports applications with working sets that is re-used across multiple parallel operations
- the main abstraction in spark is the RDD which represents a read only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost
- users can explicitly cache an rdd in memory and reuse it in multiple operations
- RDD get fault tolerance through lineage that is stored information within each partitioned to understand how that partition was derived from other RDDs
- RDDs are at a sweet spot of expressiveness and scalability

- spark works in Scala
- it is the first general purpose programming language that can be user interactively to work with large datasets over a Cluster

programming model

- to use spark developers write a driver program that implements the high level control flow of their applications and launch operations in parallel

RDD

- RDDs are a read only collection of objects partitioned across a set of machines
- ellements of an rdd dont have to exist in physical storage, instead a handle to an RDD has enough information to compute the RDD starting from data in reliable storage
- this means rdd can be reconstructed if a node fails
- RDDs can be built in 4 ways. from an object, by parallelizing a scala object, by transforming an existing rdd, by changing the persistance of an RDD
- RDDs are by default lazy and emphemeral that is partitions of a dataset of computed when they need to be used for parallel computation
- this can be changed wih the cahce and save methods
- the cahce method keeps the rdd lazy, but says it should stay in memory after computed because it will be re-used
- the save method evaluates the dataset and writes it out to HDFS

parallel operations

- spark has the following parallel operations
    1. reduce combined dataset elements using a reduce function
    2. collect send all elements of the dataset to the driver in program
    3. foreach pass each element through a user provided function
- spark does not support a grouped reduce operation like map reduce (that is group by)

shared variables

- there are two types of variables that are shared between nodes
- there are broadcast variables which are large read only objects like RDD or look up tables
- there are also accumulators which are variables that can only be added to, generally used as counters

examples

text search

- suppose we want to count the errors in a large dataset into an rdd
- we read in teh file filter for those rows with error
- map rows with error to one
- then reduce with a sum
- the middle two lines are transformations not actions, so they can be done lazily

logistic regression

- read the file in as an rdd and cache it since we will be using it a lost
- then works more or less normally in parallel just usingan accumulator for the gradient

implementation

- spark is built on top of mesos a cluser operating system this is helpful because it lets spark run alongside other existing computing frameworks
- the core of the spark implementation is RDD's
- all rdds have liniage which track how they derived from there parent data source
- each rdd has a get partitions operation as well as getiterator adn get preferd lcoaiton
- when a parllel operation is invoked spark creates a task for each partion and sends these task to worker nodes

- try to send tasks as close to there data source as possible to achive locality
- linage structure makes it easy to trace back where something fialed and thus have fault tolerence if you need to reconstruct it

shared variables

- they talk about how shared variables are serialized which is important but not super revelnt for this

results

- the rest of this paper is just talking about results and related works.