###논문 spark\_hpssd####

I. Introduction

분산컴퓨팅에서 memory bandwidth 및 capacity가 중요한 bottleneck임.

high performence SSD를 사용하고, caching과 shuffling location을 분리함.

-> 에너지 & 비욜 효율성

II. Background & Motivation

data parallelism -> MapReduce & Dryad framework

Hadoop : an open source implementation of MapReduce. input data를 쪼개서 작업해서 parallelism을 가능하게함. cost에만 좋은게 아니라 data locallity에도 좋다. 그러나 explicit data movements가 포함됨. task&data synchronization 때문에 I/O가 bottl이 됨.

spark: 분산컴퓨팅에서 node의 메모리 용량이 충분히 활용되지 않는 문제를 위해 만들어짐? 하둡보다 빠름. 처리 중간 data를 memory에 저장해서 speed up.

core 갯수를 늘려도 이상적인 만큼 결과가 나오지 않음. memory용량 문제~

-> high performance ssd 사용해서 해결하자.

SSD가 DRAM보다 느리지만, 가격당 용량이 매우 크므로 더 이득.

RDD랑 shuffle data를 내부 메모리에 저장하는 것보다 저장장치에 저장하는 것이 이득임. -> CMP(chip multiprocessor)의 메모리에서의 inter-thread 간섭은 피할 수 없지만, 메모리를 execution과 storage 용 같이 사용하는데서 발생하는 간섭은 줄일 수 있다...

SATA HDD, PCIe SSD랑 Tach-RAM(?) 비교!

III. I/O Optimization on Spark

spark RDD(Resilient distributed datasets): RAM을 read-only로 사용하자! -> data coherence를 신경 쓸 필요가 없다. 아마 dirty 얘기인듯.

### miniC ###

1. builder는 어디에서 만드는 건가요…. Function 만들때 넣을라고 하니까 parameter alloca가 불가능하다는 문제가 이씀!

2. add 이런거 할때 fadd, add 구별해야됨??? convert하는것도 다 해야됨??

3. expr 계산하는거 할때마다 %register 새로 만드는건가? return도 값이 아니라 ptr return하는건가? → ir.constant를 return 시켜도 되는건지? 다 symbol table에 추가해야 하는건가?! 그럼 음수처리는 어떻게하나…? -1이랑 \*시키나???

4. terminal node 는 type비교가 불가능한거같은데… 구분…?

‘+’ 이런거는 비교가능하다

Intnum 은 isdigit()으로 본다고 하고 floatnum은 어떻게 구분?!

#### PPT Spark Architecture ###

1. spark motivation

→ 하둡 map reduce에서 직접 프로그래밍 하기 어렵다.

하둡 코딩 61줄이 스파크 코딩 1줄임.(ex. Word count implementation)

→ 딱 맞는 case가 아니여서 생기는 performance 병목현상.

Single map reduce job하는데 HDD에 data를 3~4번 넣어야 됨.

Hive query 한번에 3-5 map reduce job. Map reduce 한번에 3번 이상의 HDD data 넣는 작업. 한번 HDD에 data 넣을때마다 read+대략 hive 한번에 18-30 scan 필요

그러나! spark는 lazy computation해서 execute하기 전에 최적화를 함. In-memory data caching을 해서 HDD는 한번만 scan하고 RAM을 scan함. 효율적인 pi으로 HDD에서 data 충돌을 최소화함.

→ Machine learning같은 반복되는 작업에 더 좋음.

2. spark pillars(핵심)

주요개념 – RDD(Resilient distributed dataset), DAG(directed acyclic graph)

→ RDD

RDD는 partition으로 분할된 data 모음이며 cluster의 worker node에 있는 메모리에 저장됨.

Data transformation을 위한 인터페이스이다.

RDD는 영구 저장소(HDFS, Cassandra, Hbase 등)이나 cache(메모리, 메모리+디스크,디스크 등)이나 다른 RDD에 저장된 data를 말한다.

partition들은 실패하거나 cache에서 제거될 때 다시 계산됨.

interface를 위한 metadata

Partitions - RDD와 관련된 data split set

dependencies – 계산과 관련된 parent RDD목록

compute - dependencies로부터 parent partition이 주어진 RDD의 partition을 계산하는 기능.(뭔소리;)

preferred locations – 이 partition에서 계산을 할 최적의 장소(data locality)

partitioner – data가 partition으로 분할되는 방법

RDD는 Spark의 data 연산(을 위한 유일한 도구.

Lazy computation model → metadata가 바뀔때마다 transformation이 일어나고, 완성되면 action이 한번 일어나서 value를 compute함.

→ DAG

data에 대해 수행된 계산 순서를 나타낸 그래프

node – RDD partition

edge – data 위의 transformation

acyclic – 그래프를 이전 partition으로 되돌릴 수 없음

direct - transformation은 partition 상태를 전환하는 것임.

3. Spark Architecture

spark cluster → driver node하나에 worker node여러개. Driver node 안에는 driver, 그안에 spark context. Worker node 안에는 executor들. Executor 안에 cache랑 task들.

Driver – Spark Shell 의 입구. Spark context가 만들어지는 곳. RDD를 execution graph로 translate. graph를 stage로 분할. task들을 스케쥴링하고 그들의 execution을 조절함. 모든 RDD와 그 partitions의 metadata를 저장하고 있음. Spark WebUI에 job 정보를 가져옴.

Executor – JVM heap의 cache나 HDDs에 data를 저장. 외부 source로부터 data를 읽어옴. 외부 source에 data를 적음. 모든 data processing을 수행함.

Application – 일부 data processing logic을 저장하고 일련의 job을 순차적으로 또는 병렬로 스케줄링할 수 있는 Spark Context의 단일 instance.

Job – driver application에 의해 유발된 action이나 data saving으로 완료되는 RDD transformation set.

Stage - transformation들의 set으로 pipeline되거나 단일 worker로 인해 execute될 수 있다. 일반적으로 read, shuffle, action, save중 하나임. (pipelining된 task set이 포함됨)

Task - 단일 data partition에서의 stage execution. scheduling의 단위.

Persistence level – LRU rule써서 memory를 cache처럼 사용. Persistence level에 “disk”가 포함되면 data가 disk로 쫓겨나짐.

4. Spark Shuffle

shuffle이란? 각 worker에 존재하는 data partition에 대해서 다른 partition으로 이동하는 것을 의미한다.

Hash Shuffle(옛날거) → Partition 하나에서 “map” task 한번 할때마다 output file을 조금씩 채워나가는듯...

Sort Shuffle(지금꺼) → 잘 모르겠음. ㅎ sort&spill 을 해서 output file을 채우고 min heap merge해서 “reduce” task 로 만든다고..

Tungsten Sort(new!)

5. Spark Data Frame

language frontend(SQL, Python, Java, Scala 등등) → Data frame logical plan → tungsten backend(JVM, LLVM, GPU 등등)

- Interface

data frame은 필드 이름, 필드 데이터 타입, 통계 등등의 스키마를 포함한 RDD를 말한다.

모든 language로 통일된 transformation interface를 사용하면 모든 transform이 JVM으로 전달됨.

dataframe은 RDD처럼 접근될 수 있으며, 이 경우 Row 객체의 RDD로 transform된다.

- internals

내부적으로는 RDD와 동일함.

data는 행렬 형식으로 저장되며, 행 chunk의 크기는 spark.sql.inMemoryColumnar로 정해진다.

각 partition의 각 column에는 partition pruning(?)에 대한 min-max 값이 저장된다.

표준 RDD보다 우수한 압축률을 제공함.

작은 하위 column 집합에 대해 더 빠른 성능을 제공함.

#### Spark의 핵심은 무엇일까??? ###

Spark - 분산 프레임웍

spark = RDD + Interface

1. Motivation

MapReduce의 단점때문. 복잡한 multi-stage한 처리(머신러닝, 그래프), interactive하고 ad-hoc한 쿼리 등을 잘 처리하지 못함. pregel같은 특수목적 분산 프레임웍들이 나왔지만, 더 근본적인 접근이 필요해짐.

MR이 iteration에서 느린 이유: iteration할 때마다 stage간의 자료 공유가 HDFS(read & write)를 거치기 때문임. RAM을 사용해서 시간을 줄이자.

RAM을 사용하다가 fault가 일어나면? Fault-tolerant & efficient한 RAM? Fault-tolerant를 위해 replicating이나 checkpointing 쓰면 또 느려짐.

→ RAM도 read-only로 사용한다면? RDD(Resilient Distributed Datasets)

immutable하다. Storage→RDD 나 RDD→RDD 로의 변환만 가능.

immutable하므로 부모로부터 만들어진 계보(lineage)만 기록해도 fault-tolerant.

코딩을 하는 것은 실제로 계산 작업을 하는 것이 아니라 점점 lineage 계보를 DAG(directed acyclic graph)로 디자인해 나가는 것.

2가지 RDD operation

transformation(map, reduce, join 등등): 데이터의 이합집산, MR보다 명령어가 많음. 실제 계산은 일어나지 않음.

actions: 실제로 transformation된 결과를 도출하는 명령.

Lazy-execution

인터프리터에서 transformations로 코딩하여도 lineage만 생성되고 계산은 안됨. action을 명령해야 쌓여있던 계산들이 실행됨. Lineage로 대강의 execution plan이 다 만들어진 뒤 시행하므로 자원이 배치될 상황등을 미리 고려해서 계산이 가능.

2가지 type의 Dependencies

narrow dependency → 한번에 다 처리할 수 있는 일은 모아서 하는게 좋다. 해당 작업이 한 node에서 해결이 가능. 네트워크를 사용하지 않으므로 엄청 빠름(memory의 속). partition이 fault되도 그 node에서 복원이 가능하다.

Wide dependency → 여기저기에서 읽어와야 하는 일은 느리다. 여러 node를 돌아다녀야 하므로 네트워크의 속도. 셔플을 해야한다(?!). fault가 일어나면 복원비용이 비싸다. checkpointing해주는게 더 좋을 수도 있다.

Job scheduling

DAG에 따라 계산해나감. 이미 계산된 partition은 제외. 필요한 partition들을 lineage를 이용해 만들어가며 계산한다. partition이 수행될 node는 data locality(특히 HDFS에서)를 이용해서 결정됨.

데이터 덩어리는 partition으로 쪼개어져서 machine(worker인듯)들에 나뉘어져 있음. driver에서 내려온 transform들이 partition마다 수행됨. 필요에 따라 (ex shuffle) partition 내용이 다른 machine으로 이동됨.

수행중 RAM 메모리가 부족해지면? LRU로 partition을 날림. (cache랑 비슷)

수행중 fault가 나면 recovery는? Lineage는 용량이 적어서 잘 저장해놓고 있다가 특정 partition에서 문제가 생기면 다른 node에서 가져와 실행.

Checkpointing? lineage로 복원이 가능하지만 wide dependency에서는 다시 계산하는 것보다 disk에 checkpointing헸다가 읽어오는 게 빠를 수 있음. read-only이므로 recovery할때 멈출 필요없이 parallel로 백그라운드에서 async로 돌린다..

Spark vs in-memory HDFS를 사용하는 HadoopBM

spark가 압도적. hadoop의 software stack이 너무 복잡해서 비용이 비쌈. HDFS라는 구조를 다루는 비용도 비쌈. Bin 계산을 하는데 java object로 또 돌리는데 비싸다(???). spark는 java object상태로 계속 활용한다(???)

*############### intellij #################*

JAVA\_HOME=/home/dongeun/jdk1.8

연구실에서 논문쓰는거 10월말까지 <한국정보과학회 동계학술발표회>

초안이 적어도 10월 7일까지

core/src/main/scala/org/

1. CourceGrainedSchedulerBackend

->makeoffers 함수 : mapping

-> recieve : dag scheduler에서 날아오는거 받는데인가봄...ㅎㅅㅎ;; reviveOffers라는 메시지가 날라오면 mapping하는거군

2. TaskSchedulerImpl

3. TaskSetManager

worker 노드들마다 executor가 많이있음

spark의 scheduler는 두가지 종류가 있음

DAG scheduler -> job이나 stage의 순서를 스케쥴링~

Task scheduler -> task를 어떤 node(executor)에 할당할까아ㅏ (locality를 이용한데)

일의 단위 Job->stage->task (작아지는 순서)

course-grained : task set을 뭉텅이로 다 준비되면 같이 실행하는 것

fine-grained : 하나라도 준비되면 먼저 실행하는거

scala

val → 처음에 값을 설정하면 바꿀 수 없음

var → 변수

map → list => new list 반환하는 함수

case?????

1. CourseGrainedSchedulerBackend.scalar

# def receive -> DAG scheduler로부터 메시지를 받음. ReviveOffers인 경우 makeOffers() 호출

# def makeOffers -> 가짜 resource 를 모든 executor에 제공. (parameter가 있는경우: 원하는 executor에만 제공)

val activeExecutors : 살아있는 executor list

val workOffers : executor ID → WorkerOffer(executor ID, executor host, free core) 로의 map

# def launchTasks(resourceOffers(workOffers)) → resource offers에 의해 return된 task들을 실행

2. TaskSchedulerImpl.scala

# def resourceOffers : cluster manager로부터 불려서 slave에게 resource를 제공해준다. Active Task set중 priority에 따라 respond 해줌. 각각의 node를 round-robin방식으로 task를 할당하여 cluster에 task가 균형있게 분포되도록 한다. Task set은 scheduling order에 따라, node는 locality level이 증가하는 순서에 따라 할당한다.

=> Task set을 node에 할당하는 과정인듯하다.

task set하나당 그중 max locality

val hostToExecutors : HashMap[String, HashSet[String]] → 각각의 host가 가진 executor set

Val shuffledOffers : offer를 랜덤하게 섞어서 task를 항상 같은 worker set에 할당하는 일을 피한다.

Val sortedTaskSets : rootPool.getSortedTeskSetQueue (rootPool이 뭔지?!)

var launchedTaskAtCurrentMaxLocality : T/F

# def resourceOfferSingleTaskSet (taskSet, maxLocality, shuffledOffers, availableCpus, tasks):

=>

for(task ← taskSet.resourceOffer(execId, host, maxLocality)

return된 Task description의 task를 execute하기 위해 executer ID도 배정해주고, CPU도 배정해주고 launch task함! 성공하면 true return.

3. TaskSetManager.scala

# def resourceOffer : task를 찾아서 bookkeping을 하고 task description을 return한다. task의 locality를 update한다.

????? BlackList가 뭘까. -> task를 쓸수없는거…? Failure가 너무 많이 떠서 scheduling안하려고 제외하는 항목.

????? map{case()=>()}앞에꺼가 case조건에 맞으면 살행. :: 는 concate인걸로~

Var currentLocalityIndex = getLocalityIndex(taskLocality)

# def getLocalityIndex(locality) : locality <= myLocalityLevels(index)를 만족하는 가장 작은 index를 return

→ 더 작은 범위의 local을 찾는 것임!

Var myLocalityLevels = computeValidLocalityLevels()

# def computeValidLocalityLevels : TaskSet에서 사용된 locality level을 계산한다. 모든 tasks는 addPendingTask를 이용해서 이미 queue에 추가되었다고 가정한다. levels.toArray를 return한다. task set 전체의 valid locality 계산.

val levels → task locality(아래 5개) 중 해당하는 것들을 array로 가진 배열.

import TaskLocality.{PROCESS\_LOCAL, NODE\_LOCAL, NO\_PREF, RACK\_LOCAL, ANY}

PROCESS\_LOCAL : best locality possible as processing code and data lives in same JVM

NODE\_LOCAL : data and processing code lives in same server/node. Tasks with this level runs slower than tasks run with PROCESS\_LOCAL level.

NO\_PREF : with this level, there is no locality preference. → data가 어디서 오는지를 결정할 수 없는 경우.

RACK\_LOCAL : data is on same rack of servers.

ANY : data resides on different network.

# def dequeueTask(execId, host, maxLocality) : pending(미결) task를 주어진 node에서 dequeu한다. index와 locality level을 return한다. 주어진 locality constraint에 맞는 task만 searching한다.

???? for문을 돌면서 return하는게 무슨 의미가 있는지? 그냥 맨앞에꺼 return하는거라고.. map{case()=>()} ???

#def dequeueTaskFromList(execId, host, list) : 주어진 list에서 pending task를 dequeue해서 그 index를 return한다. List 가 빈 경우 none을 return. list에 이미 실행된 task를 lazy하게 clean up해주는 역할도 한다.

#def getPendingTasksForExecutor(execId) : executor ID를 위한 pending tasks list를 return

#def getPendingTasksForHost(host) : host를 위한 pending tasks list를 return

################# RDD time measurement##################

@@ time measurement in Scala @@

*// scalastyle:off println*

**override def** time[R] (block: => R): R = {

**val** t0 = System.*nanoTime*()

**val** result = block *// call-by-name*

**val** t1 = System.*nanoTime*()

*println*(**"Elapsed time: "** + (t1 - t0) + **"ns"**)

result

}

*// scalastyle:on println*

var list = List.range(1,1000,1) 이 걸리는 시간을 측정하고 싶다면?

var list = time {List.range(1,1000,1)} 을 사용하면 된다~

@@ RDD.scala @@@

transformation!

1. def map
2. def flatMap
3. def filter
4. def distinct

@@ HadoopRDD.scala @@@

class HadoopPartition -> ??…

class HadoopRDD

def compute -> return값 : InterruptibleIterator[(K,V)](context, new NextIterator[(K,V)])

@@ MapPartitionsRDD.scala @@@

@@ShuffledRDD.scala @@@

\*\*\* spark 켜는 법

HADOOP에서 sbin/start-dfs.sh

SPARK에서 sbin/start-all.sh

\*\*Spark 돌려보기

bin/spark-submit run-example JavaWordCount hdfs://localhost:9000/user/dongeun/input

bin/spark-submit run-example JavaWordCount hdfs://localhost:9000/user/dongeun/input hdfs://localhost:9000/user/dongeun/output

\*\*Spark configuration 설정

conf 폴더에서 vim spark-defaults.conf

https://spark.apache.org/docs/2.1.0/configuration.html

\*\*\*spark rdd

/lab/spark-2.1.0/core/src/main/scala/org/apache/spark/rdd

\*\*spark build

./build/mvn -Pyarn -Phadoop-2.7 -Dhadoop.version=2.7.2 -DskipTests package

\*\*Spark 예시들

/lab/spark-2.1.0/examples/src/main/scala/org/apache/spark/examples

Pi같은거 보기

\*\*\*\*intellij 열기

JAVA\_HOME=~/jdk1.8.0\_144/ ./idea.sh

\*\*\*\*\*출력

// scalastyle:off println

// scalastyle:on println

1. AsyncRDDActions X
2. CartesianRDD O
3. CoalscedRDD O
4. EmptyRDD O
5. LocalCheckpointRDD O
6. OrderedRDDFunctions X
7. ParallelCollectionRDD O
8. PartitionwiseSampledRDD O
9. RDDCheckpointData X
10. ReliableRDDCheckpointData X
11. SubtractedRDD O ->길어 ㅠ
12. ZippedPartitionsRDD O

\*\*\*\*\*\*\*\*\*\*\* python 짠거\*\*\*\*\*\*\*\*\*\*\*

1. fileInput.py

file=open("output.txt", 'r');

for line in file:

line=line.rstrip()

if len(line)>2:

if line[0]=='\*' and line[-1]=='\*':

print(line[1:-1])

file.close()

1. redirection.py

while 1:

try:

line=raw\_input()

except EOFError:

break

else:

line=line.rstrip()

if len(line)>2:

if line[0]=='\*' and line[-1]=='\*':

print(line[1:-1])

\*\*\*\*\*\*\*\*연구실컴 원격접속!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ssh public@115.145.178.202 -p 17113

password: arcslab.400617

ssh dongeun@115.145.175.28

hdfs://master:9000/logs

ssh arcs@115.145.178.222~224 slave5,6,7

password:스페이스+엔터

home/config 폴더에 spark-defaults1.conf 2, 3 넣어놓기

비밀번호 arcslab.400617

10.201.209.150

arcs

dkfzt^8565  
/dev/nvme1

FAIL!!

성능실험 하셈. fio 내일까지

\*\*\*\*\*\*\*scala -> jar 파일 만들기 \*\*\*\*\*\*\*\*\*

sbt package

\*\*\*\*\*\*논문쓰는거\*\*\*\*\*\*

coalesce

\*CoalescedRDD

repartition

\*MapPartitionsRDD 여러개

\*CoalescedRDD 여러개

\*MapPartitionsRDD

\*ShuffledRDD

sample

\*PartitionwiseSampledRDD

sortByKey

\*MapPartitionsRDD 여러개

\*PartitionPruningRDD

\*PartitionwiseSampledRDD

\*MapPartitionsRDD 여러개 2개?

\*PartitionPruningRDD

\*PartitionwiseSampledRDD

\*MapPartitionsRDD 여러개

\*ShuffledRDD

repartitionAndSortWithinPartitions

\*MapPartitionsRDD 여러개

\*PartitionPruningRDD

\*PartitionwiseSampledRDD

\*MapPartitionsRDD 여러개

\*ShuffledRDD

cogroupedRDD 쓰는거

intersection

cogroup

join

\*\*& 일단 transformation 종류 20개

map

filter

flatMap

mapPartitions

mapPartitionsWithIndex

sample

union

intersection

distinct

groupByKey

reduceByKey

aggregateByKey

sortByKey

join

cogroup

cartesian

pipe

coalesce

repartition

repartitionAndSortWithinPartitions

ShuffledRDD 쓰는데

distict

groupByKey

reduceByKey

aggregateByKey

sortByKey

repartition

repartitionAndSortWithinPartitions