big data.

definition.

possible solution { scale up / vertically — expensive
scale down / horizontally

failure.

develop programming model

cloud computing: services through internet.

3 services: infra. platform. software

" computing services cpu. memory network

" development envir

" applications

2 deploy models { public
private.

5 characteristics

Lec2 storage

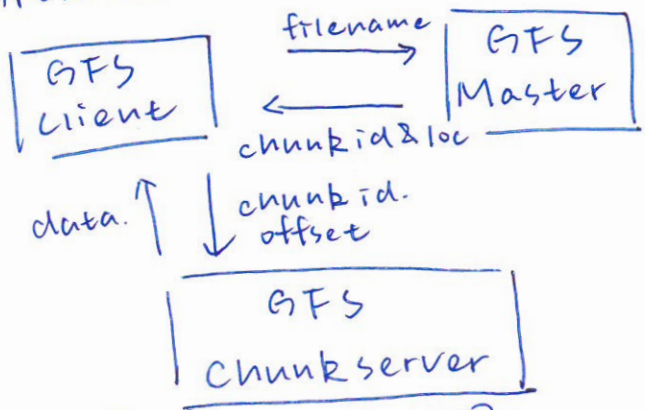
file system on single machine: X scale.
large file x

↓

Distributed file system

GFS.

Architecture.



Master: ① metadata (×3) ② heartbeat msg

Chunkserver.

client

chunk size: 64 mb.

{ ~~adv~~ pro: ↓ metadata
↓ interaction
con: wasted space↑

Consistency Model

{ consistent
defined.

operations.

{ read
update
write
append
delete

Lec3 ~~storage~~ NoSQL.

Database revolution

No DBMS → hierarchical → Relational DB → NoSQL. BAty
1st ACID 2nd NewSQL 3rd

SQL. problems: scalability.

ACID properties: Atomicity, consistency, Isolation, Durability

Theory consistency — strong consistency
— eventual consistency

CAP theorem: consistency, availability, partition tolerance

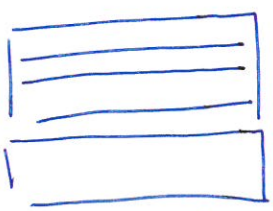
BigTable. HBase (CP). Cassandra (AP)

NoSQL BASE properties: Basic Availability, soft state. Eventually Consistent

NoSQL Data Model { key-value
column-oriented. e.g. BigTable. HBas
Cassandra.
document
graph. e.g. Neo4j

BigTable Data Model

Table columns: column family + qualifier



rows } tablet
} tablet
> Table

timestamp

client

Arch. Master

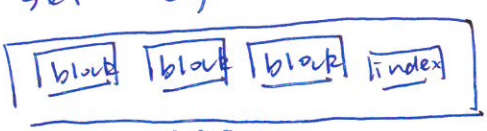
Tablet Server

Master: ① assign tablet ② balance load ③ monitor schema changes

Tablet server: ① manage tablet

Building blocks: GFS + chubby (lock service)

SSTable
SSTable
} tablet.



Lec5 Parallel Processing.

Problem: x load data in single computer

challenge: ① split computation ② manage failure.

MapReduce.

Programming model

- map phase: (key, value) pair → intermediate ( ) pair
- shuffule phase: group key value pairs by key
- reduce phase: grouped pairs → final result.

execution:

files split → master & workers → master assign. work

map worker output inter pairs → buffered in memory

→ write to disk.

master forward loc → reducer worker

→ notify master → wake up user program

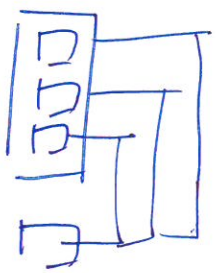
Fault tolerance: heartbeat

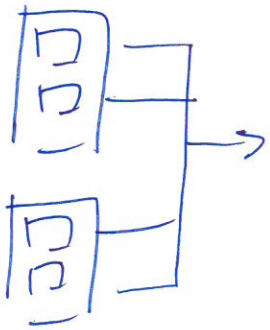
- map
  - in progress:
  - completed:

  > re-execute.
- reduce
  - in progress: re.
  - completed: result stored.

Algorithm

join
- map side join.
- reduce side join.





problem: a chain MAPreduce.

↓

FlumeJava

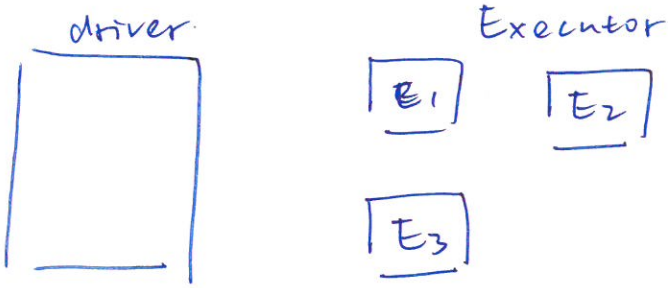
Lec 6 Spark 1

motivation: data sharing (MR in slow storage)

⇓

Spark: keep things in memory

Arch:

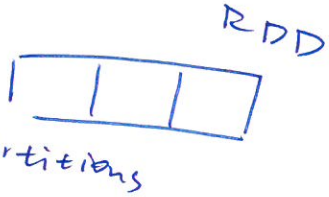


Driver: ① interact with user. ② distribute work with executor.

Executor: ① execute ② report state

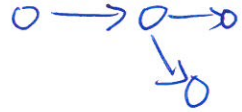
Programming Model:

RDD (Resilient distributed dataset)



operations — transformations

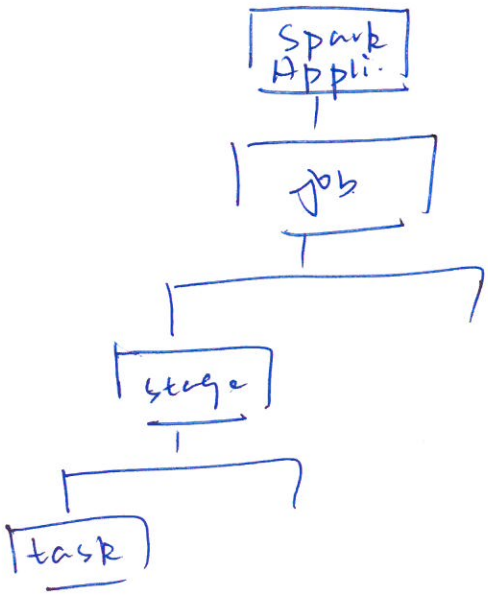
\ actions

Lineage  logical execution plan

dependencies (edges) — narrow

\ wide

Fault tolerance : re-compute



Lec7 Structured Processing = Spark SQL

MR: problem: no schema. x query

↓

Hive: + structured data on MR.

exe: parsing query to tree → generating logical plan

→ physical plan

↓

Shark: ~~modify~~ run on Hive.

limitation: x integrated with Spark. x optimized

↓

Spark SQL.

RDD: distributed data collections.

x know schema.

↓

DataFrame: structured data.

x type checked by compiler.

↓ (x compile time type safety)

Dataset:

base RDD + DF.

compile type safety + optimizer.

Execution: code → logical plan → physical plan

optimization

optimization — catalyst = reorder op.

prune partition

\ Tungsten: ↑ cpu. memory.

(off-heap memory)

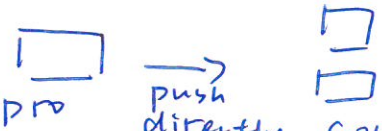
Lec 8. streaming ~~processing~~ storage : log.

streaming data: unbounded input

storage problem:

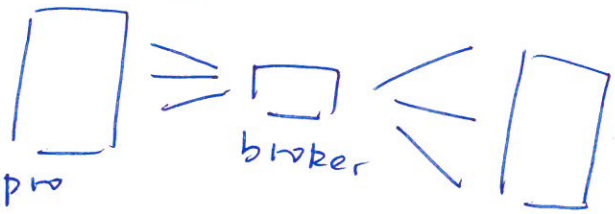
message system: spread info from producer to consumer.

direct messaging :



drawback: consumer crash
msg too fast.

message broker :



{ typical: delete msg once consumed

log-based : store in log { producer append.
consumer read sequentially

partitioned: hosted on diff machines
seq num within each partition

topic: partitions with same type msg

kafka: replicate partition → fault tolerance

zookeeper: manage broker & consumer
keep offset

---

streaming processing

1. continous processing: data → ⊓⊓⊓ → output

micro-batch : data → ⊓⊓ ⊓⊓ (batch) → output

2. Record-at-a-time API: low-level full control

Declarative API: what to do v.

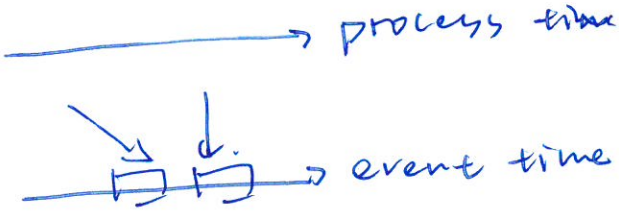
3. Event time : actually occurred

Process time : received.

4. windowing — tumbling : all ~~data~~ delete

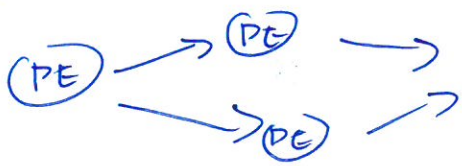
\ sliding : oldest delete

windowing — by processed time : fixed time window
         — by event: handle out-of-order events



Processing Model

Data flow graph =



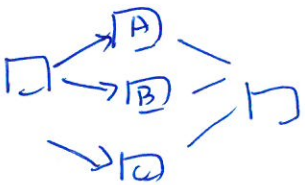
PE (processing element)

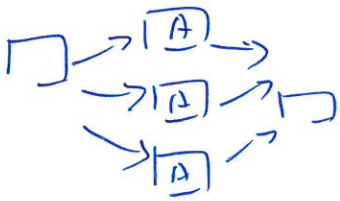
Tasks — stateless: x require prior knowledge
        easy parallelized
      — stateful : involve diff tuples

Logical plan : physical	vertice = PE	edge = streaming connection
	vertice = process	transport connection

MAD logical → physical

1. Parallelization — pipelined 

— task 

— data. 

2. Fault Tolerance:

delivery — at-least-once
         — exactly-once

recovery — active backup : (processing node + backup node) same input
         — passive backup : checkpoint
         — upstream backup : x backup. store until ack

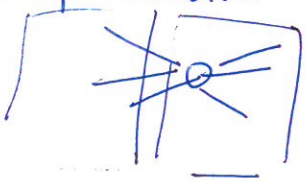
Lec 10 : streaming processing : spark streaming

Dstream (Discretized Stream Processing)

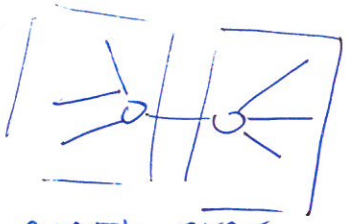
Lec 11 Graph processing

Challenges: partition data & computation

graph partition — edge cut



vertex-cut



Graph processing : process large graph over

① Think like a vertex / Graph parallel

○ compute own ○ depend on others

Pregel: synchronous

Execution: supersteps / iteration.

read (S-1) send (S+1)

Fault tolerance: workers revert to checkpoints

limitation: determined by slowest

GraphLab : asynchronous. shared memory (lock)

Fault tolerance: periodical halt + synchronize

PowerGraph : gather - apply - scatter

syn / asyn.

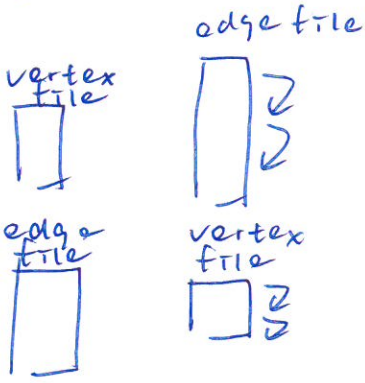
② Think like an edge

vertex-centric breadth first

edge-centric

problem: random access on vertex

streaming partition graph



③ Think like a table

motivation: restrict types

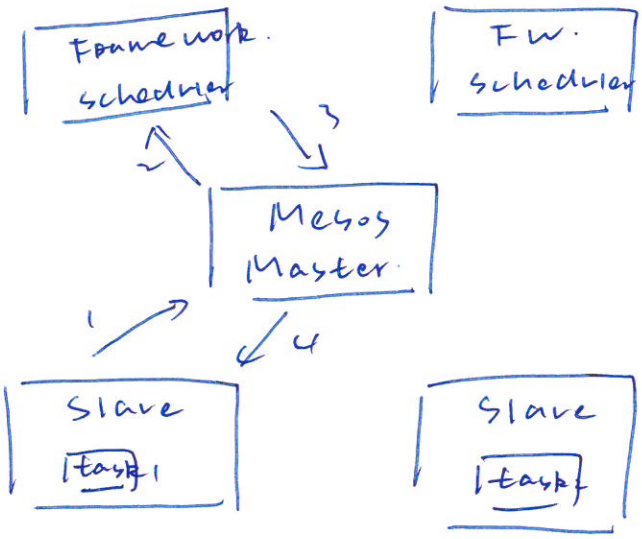
Graphx: Graph parallel in Spark

+ data parallel

Lec 13 Resource Management
= CPU + RAM.

Mesos.
- Fine grained sharing: tasks level
  coarse grained sharing: jobs / framework level
- Resource offer.

Arch:



Allocating resources
- single resource — fair sharing. $\frac{1}{n}$
  - min-max.
  - weighted min-max
- multiple resource — asset fairness (weights).
  - dominant resource fairness

YARN