Consulting Opportunities Using Apache Hadoop

IEEE Presentation
June 23, 2012

Hadoop was invented by Doug Cutting who named it after his son's toy elephant. It and the logo above are owned and maintained by the Apache Software Foundation, whose logo is the feather shown above..

Multiples of bytes

- kilobyte (KB) 10^3
- megabyte (MB) 10^6
- gigabyte (GB) 10^9
- terabyte (TB) 10^12
- petabyte (PB) 10^15
- exabyte (EB) 10^18
- zettabyte (ZB) 10^21
- yottabyte (YB) 10^24

Library of Congress ~ 10 PB

Big Data

- “Big Data” = Exceeds in quantity, transfer speed, and/or data type variety. (3V's)
- Big Data worth $100B growing 10% per year
- As of June 2012, ~2.5 Exabytes (10^18) of business data are created DAILY
- Twitter receives 12TB of data a day
- Modern airline eng: 20TB/hr performance data
- NASA Sq. Kilometer Array: 100's of TB/sec = 10's of Exabytes/day
- Machines dominate data generation!!!

What do you do ...?
(Tipping Points)

- When your data size exceeds what can be put on or handled by one machine?
- When your daily calculations take longer than a day?
- When your fancy compute systems have terrible ROA? E.g. 10% utilization
- When the Availability = %time fully functional of your computer system is unacceptable?

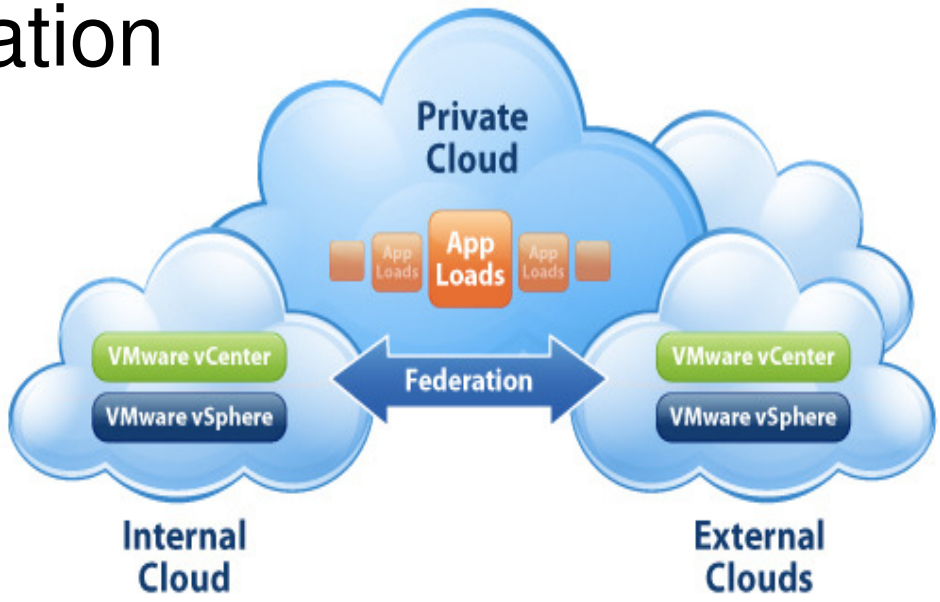

BIG DATA
— GOT IT...
NOW WHAT?
EMC

Cloud Computing

- Elastically and Independently provision:
  - Compute resources
  - Storage
  - Network capacity: access and internal
  - Resource Pooling and Load Balancing
- Only pay for what you use; includes transmission, maintenance, etc.
- Security and Reliability “Guaranteed”, but ...
- Vendors: Amazon, IBM, Silicon Graphics, Rackspace, ...

Private and Internal Clouds

- Use open source infrastructure (Eucalyptus, OpenNebula)
- House computers on client site
- Increases asset utilization
- Market is growing!



CRN's Coolest 100 Cloud Vendors

- 20 Platform and Development (Cloudera, MapR, Hortonworks, RedHat...)
- 20 Storage and Data Center (Apple, Carbonite, Dropbox, EMC, Hitachi, VMware...)
- 20 Infrastructure (Amazon, AT&T, Cisco, Dell, Eucalyptus, HP, Rackspace, Verizon...)
- 20 Applications & Software (Citrix, Google, Intuit, Microsoft, Oracle, Quest, Salesforce.com, SAP, Taleo...)
- 20 Security (CA, Check Point, IBM, McAfee, NTT, Symantec, Trend Micro, ...)

And, …, many cool startups...

Cloud Consulting Ideas

- Key Problem: Pro's and Con's of moving to the cloud. Cost Analysis. Private vs. Public.
- Choice of vendor(s) – match business models
- Are Availability requirements met? 99.9%?
- Avoiding platform lock-in (IEEE standards)
- Transition challenges and costs
- Future Big Data needs? If yes, ask many questions, e.g. can you run on real rather than virtual machines?
- Performance Issues - monitoring

10 Year Background

Moore's Law flattening out

Path to Memory is a big obstacle

Need to hook up multiple computers

Must balance disk and network performance!

Key papers from Google:

- GFS (2003)
- MapReduce (2004)
- BigTables (2008)

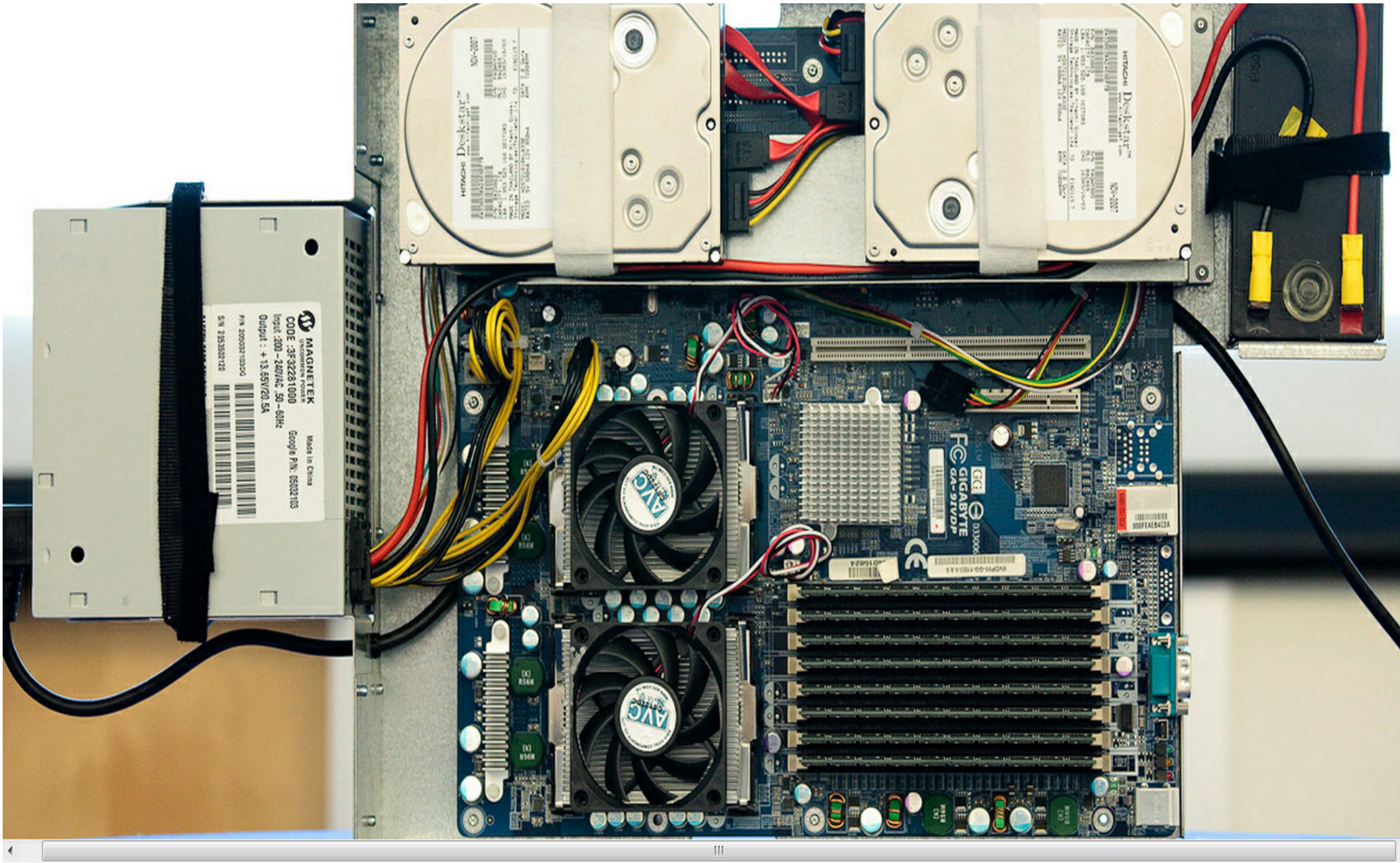
Sample Early Adopters

Facebook, Linkedin, Yahoo!– predict "People You May Know" etc
Journey Dynamics – GPS records predict traffic speed
Vestas 15,000 cores, Petabytes of data, wind energy analysis
MobileAnalytics.Net – Natural Language Processing, text to speech, machine generated video and audio
New York Times – newspaper archive image conversion to PDF
Spadac.com – geospatial data indexing
UNC Chapel Hill – analyze gene sequence data
Visa – Credit card fraud analysis
Ebay – Mining data
Amazon.com – Cloud services and retail data mining
Others: Amazon/A9, AOL, Baidu, eBay, eHarmony, Fox, Netflix, Interactive Media, Twitter, Zvents, CIA, NSA, …

Data Center Terms for Big Data

- Rack mounted server (2-4 core 8GHz CPU, 16-32 GB DRAM, 2-4 1-4TB Disks)
- Rack (40-80 servers, 1 10 GB/s Ethernet Switch)
- Cluster (30-40 Racks, 1 100GB/s Switch)
- Data Center (100's of Clusters, large routers)
  - Google has interesting patents + 1 data center description
  - Facebook has published their “green” data center specifications and designs

Google's Server



Google Server Backside

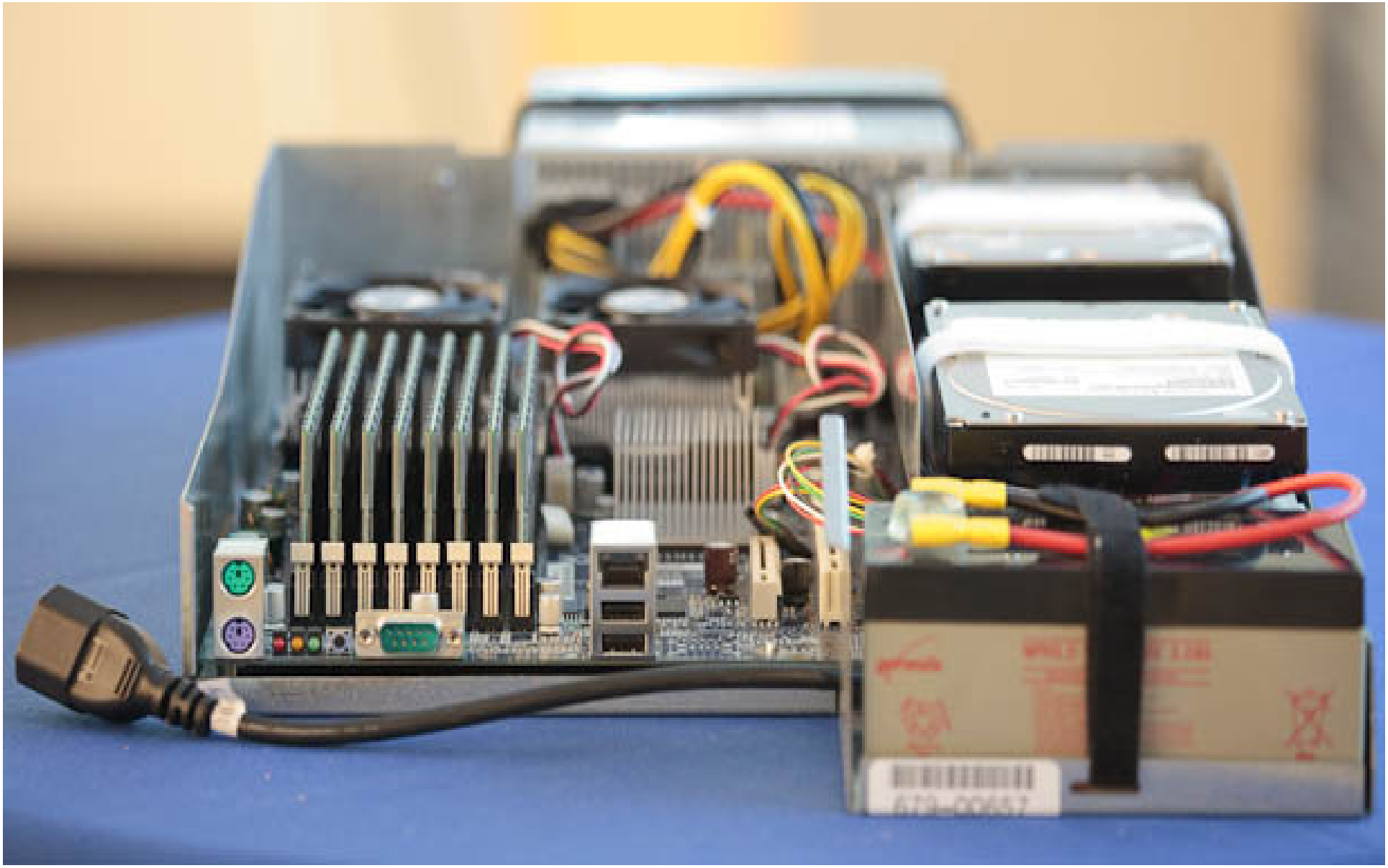
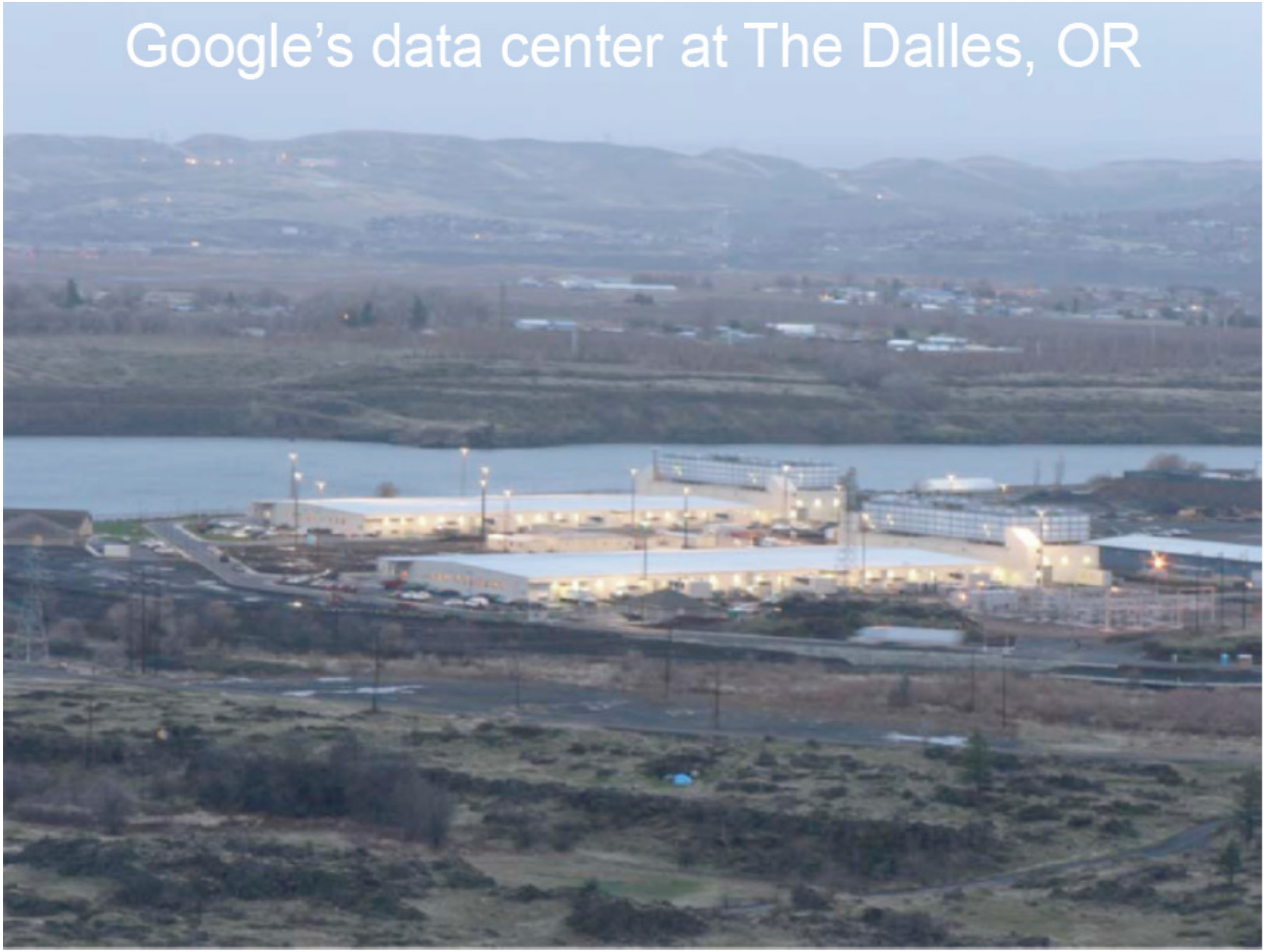


Photo by Stephen Shankland of CNET


Google’s data center at The Dalles, OR

Facebook Data Center NC

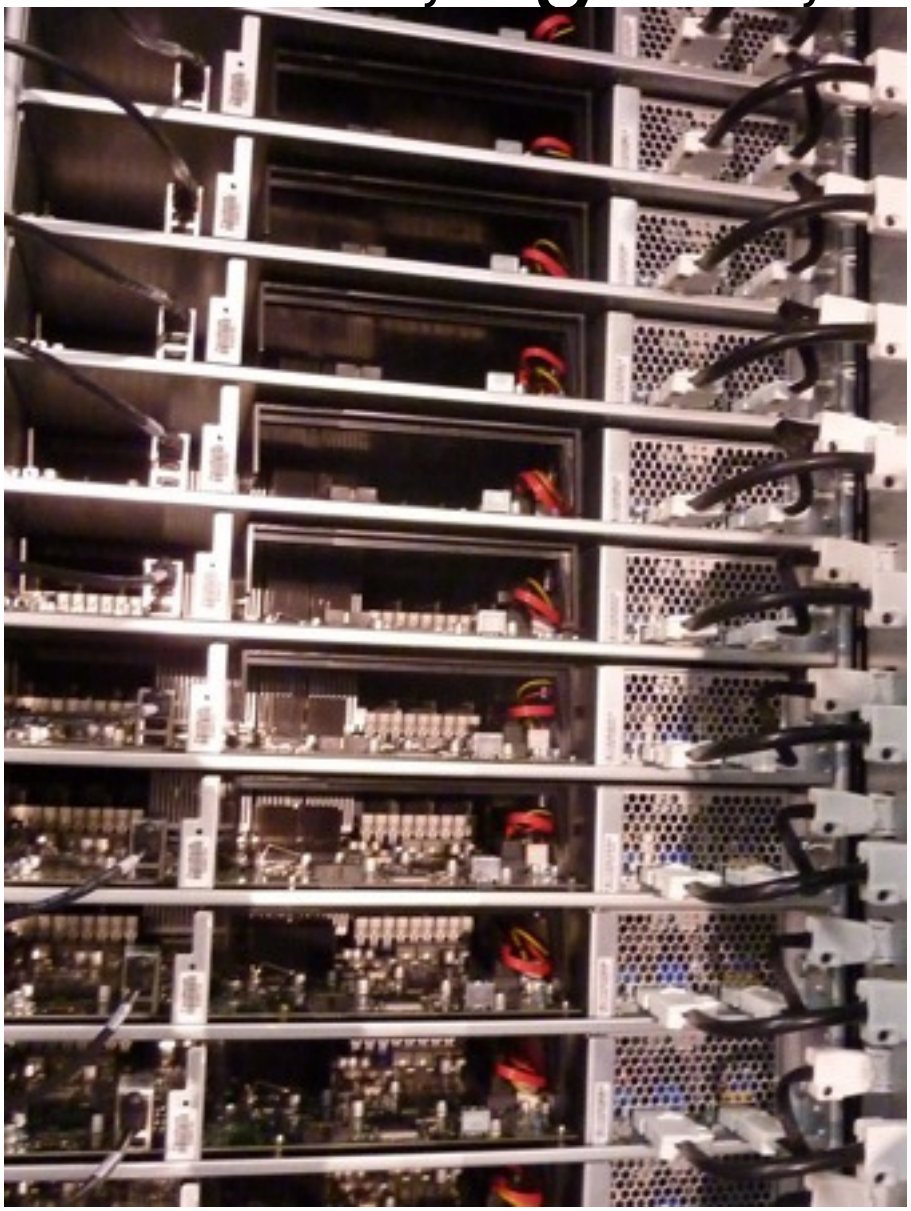
An aerial view of Facebook's new data center in Forest City, North Carolina. *Image: Courtesy Facebook*

Facebook Air Filter

Facebook Just Add Water

Facebook Rack of Servers

1.5U, no big UPS, no AC, tall heat sinks, no tools, lighter, cheaper



Google's 1AAA Container (1160 servers, many/datacenter)

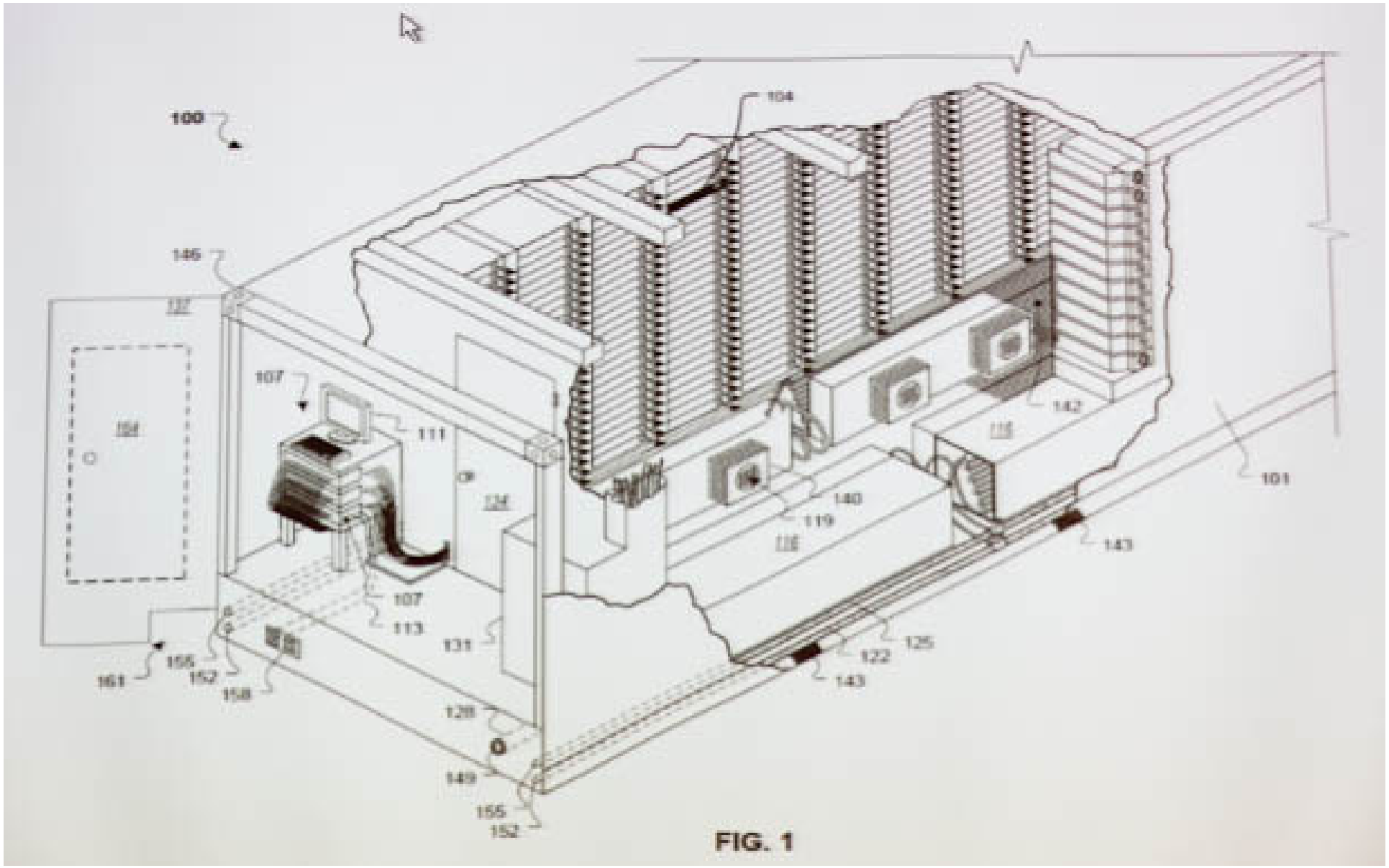


FIG. 1

Raised Floors in Google's Containers

eBay Data/Compute Servers

- 1U with 64 bit CentOS (use Enterprise Red Hat on “main” servers)
- 2 quad core machines
- 48 GB RAM
- 1 Gig Ethernet for nodes
- 12-24 TB storage
- A rack has 38-42 servers
- Rack switch has uplink of 40Gb/s to the core switches


FRUSTRATION
AHEAD
!

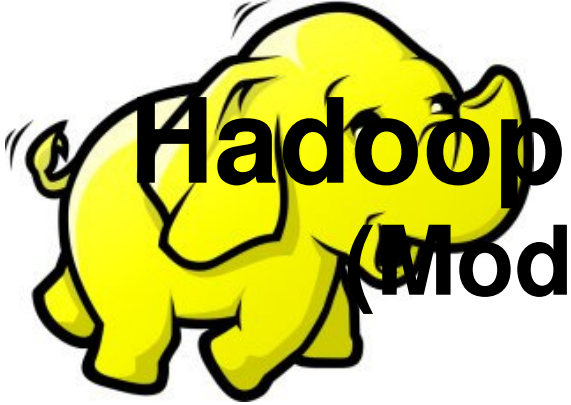
Google: Failures/Year/Cluster

- ~0.5 heat problems (pwr down most eq < 5 min, 2 day MTTR
- ~1 PDU failure (500-1000 machines gone, ~6 hrs MTTR
- ~1 rack move (~500-1000 machines down, ~6 hrs MTTR
- ~1 network rewiring (rolling ~5% down over 2 day span)
- ~20 rack failures (40-80 machines disappear, 1-6 hrs MTTR
- ~6 racks see 50% packet loss (?? MTTR)
- ~8 net maint failures (30 min connectivity loss MTTR)
- ~12 router reloads (2-3 min MTTR, web and clusters)
- ~3 router failures (web access 1 hr MTTR)
- ~1000 machine failures; ~many thousands disk failures
- Many minor, hw and sw problems; many Internet problems

Don't Yell; Deal With It!

Technology to address Big Data

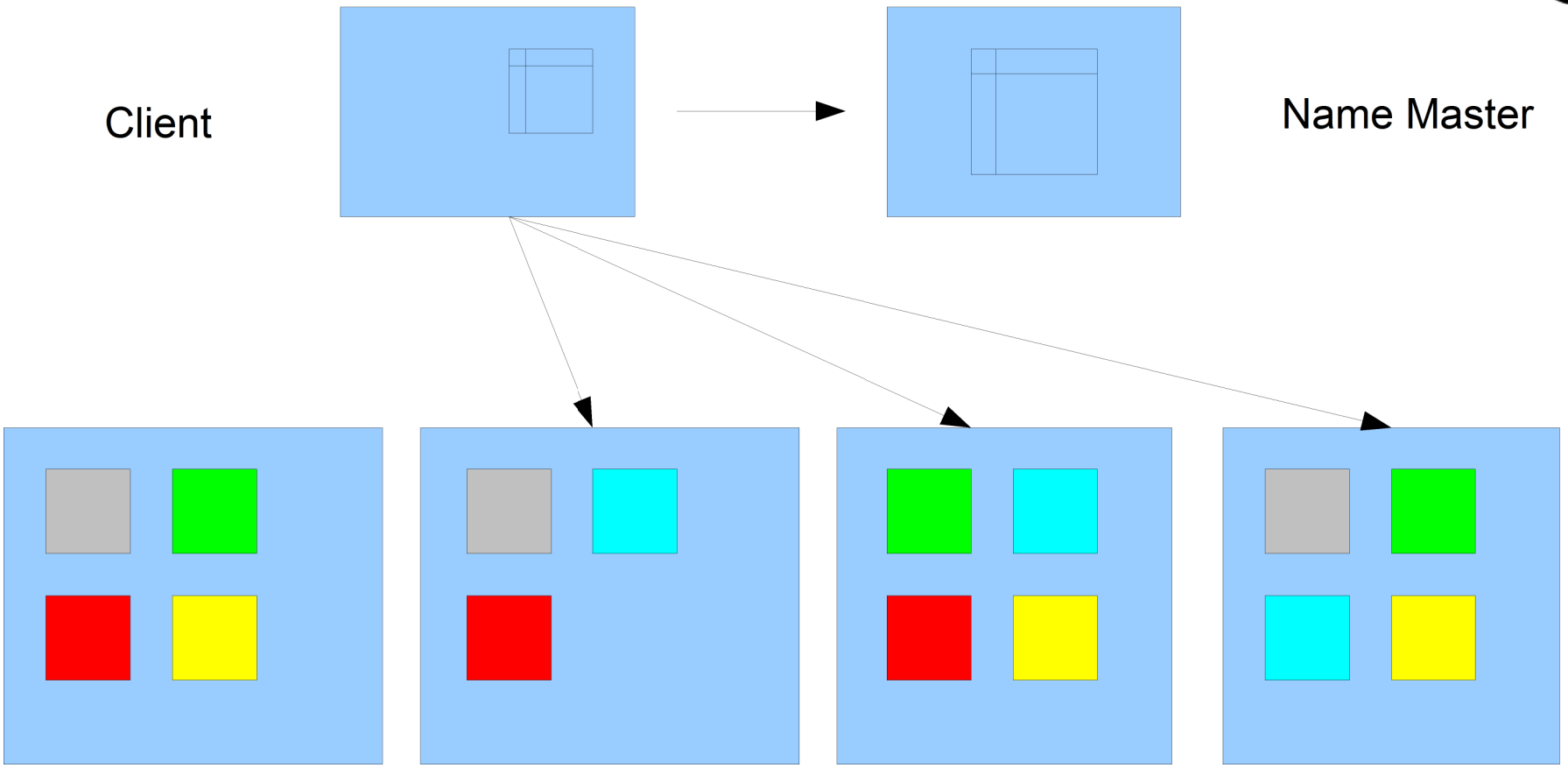
- Distributed File System for very large files, distributed redundantly over many machines
- MapReduce computing environment provides distribution and fault tolerance
- Integrate your RDBMS!!! The above technology will **not** solve traditional problems for which a RDBMS does well!!!
- Additional infrastructure (Lots new here!)



Hadoop Distributed File System (Modeled after Google's GFS)

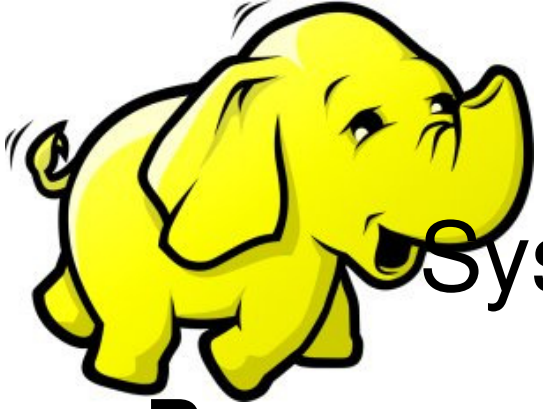
- Divide (big) file into large “chunks” 64-128MB.
- Replicate each chunk at least 3 times, storing each chunk as a linuxfile on a data server. Usually put 2 in the same rack.
- Optimize for reads and for record appends.
- Before every read, checksum the data.
- Maintain a master name server to keep track of metadata, incl. chunk locations, replicas,...
- Clients cache metadata and read/write directly to the data servers.

HDFS


Client
Name Master
Chunk/Data Server
Chunk/Data Server
Chunk/Data Server
Chunk/Data Server

Goals to Distribute Applications

- Distribute the application across many machines
- Put the computations close to the data
- Near linear scaling n to N machines
- Handles failures (machine, storage, network)
- Monitor progress and deal with slow jobs

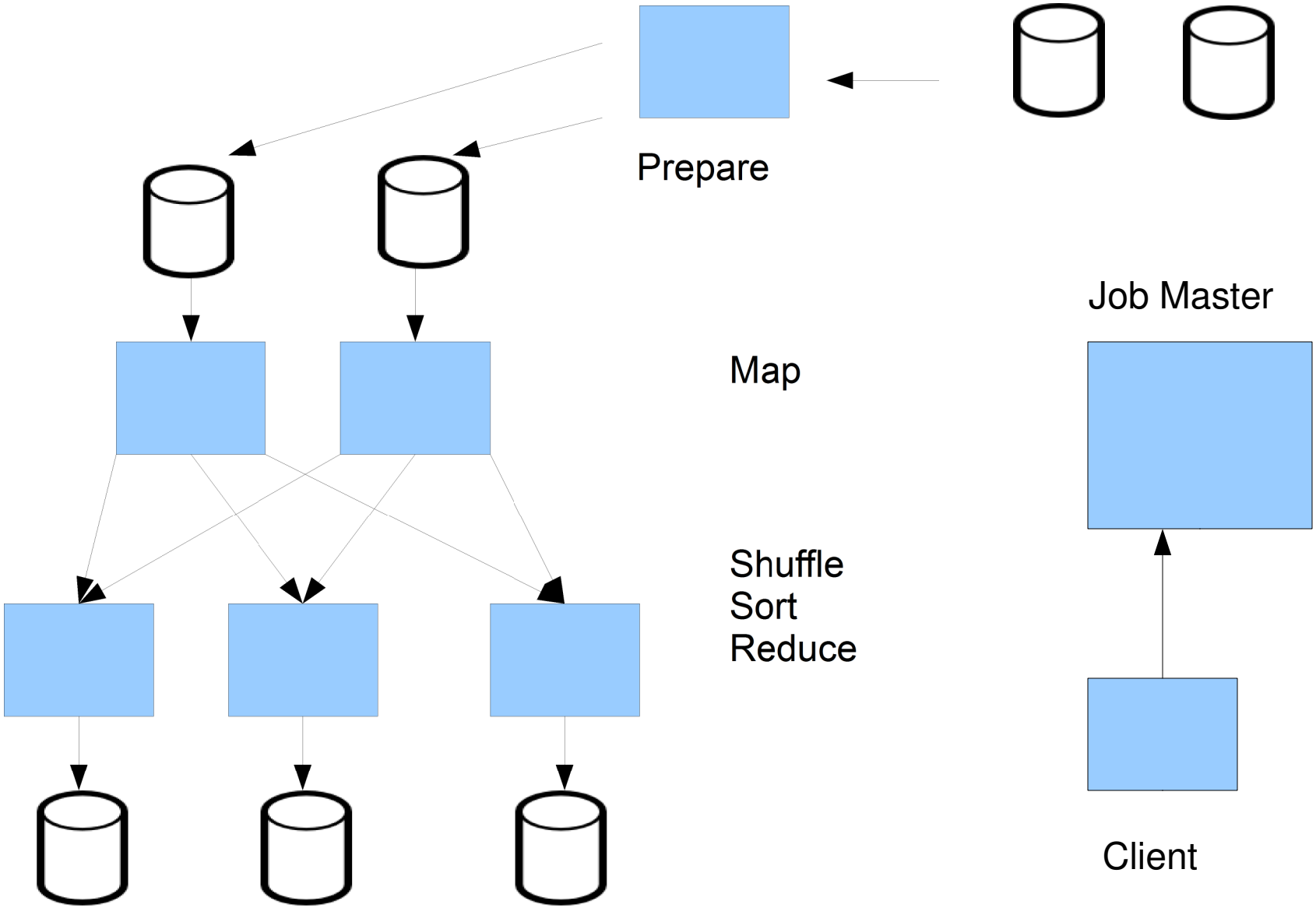


MapReduce

System to Process Big Files

- **Prepare** a Big File for input as (key1, value1) pairs into multiple Mapper jobs
- **Map**(key1, value1) generates intermediate (key2, value2) pairs and separates this output into R files for input into R parallel Reducers.
- When all mappers done, the R reducers read and **Sort** their input files.
- **Reduce**(key2, value2) does a data reduction processing stage. Output = R Big Files of (key3, value3) pairs (not sorted nor combined.)

MapReduce System



Sort on Key

- Map(key, value) = (key, value) identity fcn
- Reduce (key, value) = (key,value) identity fcn

- With multiple reducers, Map needs to have a partitioner class or a hash function so that k1 < k2 implies hash(k1) < hash(k2), an “ordered hash function”

String Search

- Map (docName, contents) → (docName, Contents)
  - If string In Contents Then emit(docName, Contents)
- Reduce(docName, Contents) = identity

- Note Google(string) does a little more, it emits the line or two that contains string, and it orders them by “goodness”

Sorted String Search

- Suppose you want to search for a string in a list of documents, but you want the output sorted.
- Map(docName, Contents)
  - Is the identity, except it hashes the input into R output files with d<d' implies hash(d) < hash(d')
    - emit(docName, Contents) to file(hash(docName)/R)
- Reduce(docName, Contents)
  - If string In Contents Then emit(docName, Contents)
- Exercise: Make more efficient

Time Series

- A *time series* is a sequence (t1,v1), (t2,v2), … where t1 $\leq$ t2 $\leq$ … are points in time.
- Medical, financial, image. Smart Grid data, …
- Surprise: genome data (text representation.)
- TS programming languages (EPS, APL, R, …)
- RDMS' put time series in a two column table; at best ok for small ts, but not for Big Data. Also SQL not so hot for ts. Ditto spreadsheets.

Moving Averages – Smoothing Data

- A TS *moving average* over time T at time t is the average of the vi for ti in (t-T, t]. Typically one takes t to be one of the ti to guarantee at least one value to be averaged. Move t to repeat..
- Three day's average stock price for IBM (typically thousands of trades with actual quantity varying daily). Usually computed daily.

Example: Daily 3-day Average Stock Market Trades

- Multiple trade streams funneled to the mappers. Map=Identity hashes symbol into R buckets; sort (time, symbol) in R reducers:
- Reduce(time, symbol, quantity, price)
  - count3[*] := count2[*] := count1[*] := sum3[*] := sum2[*] := sum1[*] := day0 := 0
  - Symbols3 := Symbols2 := Symbols1 : = { }
  - Loop
    - Input(time, symbol, quantity, price)

Example 3day moving average continued

- If EOF Or day(time) > day0 Then - - flush what we've got
  - Begin For s In Symbols3 + Symbols2 + Symbols1 Do
    - Output (s, day(time), (sum1[s]+sum2[s]+sum3[s])/(count1[s] +count2[s]+count3[s]+1))
  - If EOF Then Return
  - day0 := day(time)
  - count3[*] := count2[*]; count2[*] := count1[*]
  - sum3[*] := sum2[*]; sum2[*] := sum1[*]
  - sum1[*] := count1[*] := 0
  - Symbols3 := Symbols2; Symbols2 := Symbols1
  - Symbols1 := { } End
- **Symbols1 += symbol - - set addition**
- **sum1[symbol] += quantity*price**
- **count1[symbol] += quantity**
- End Loop

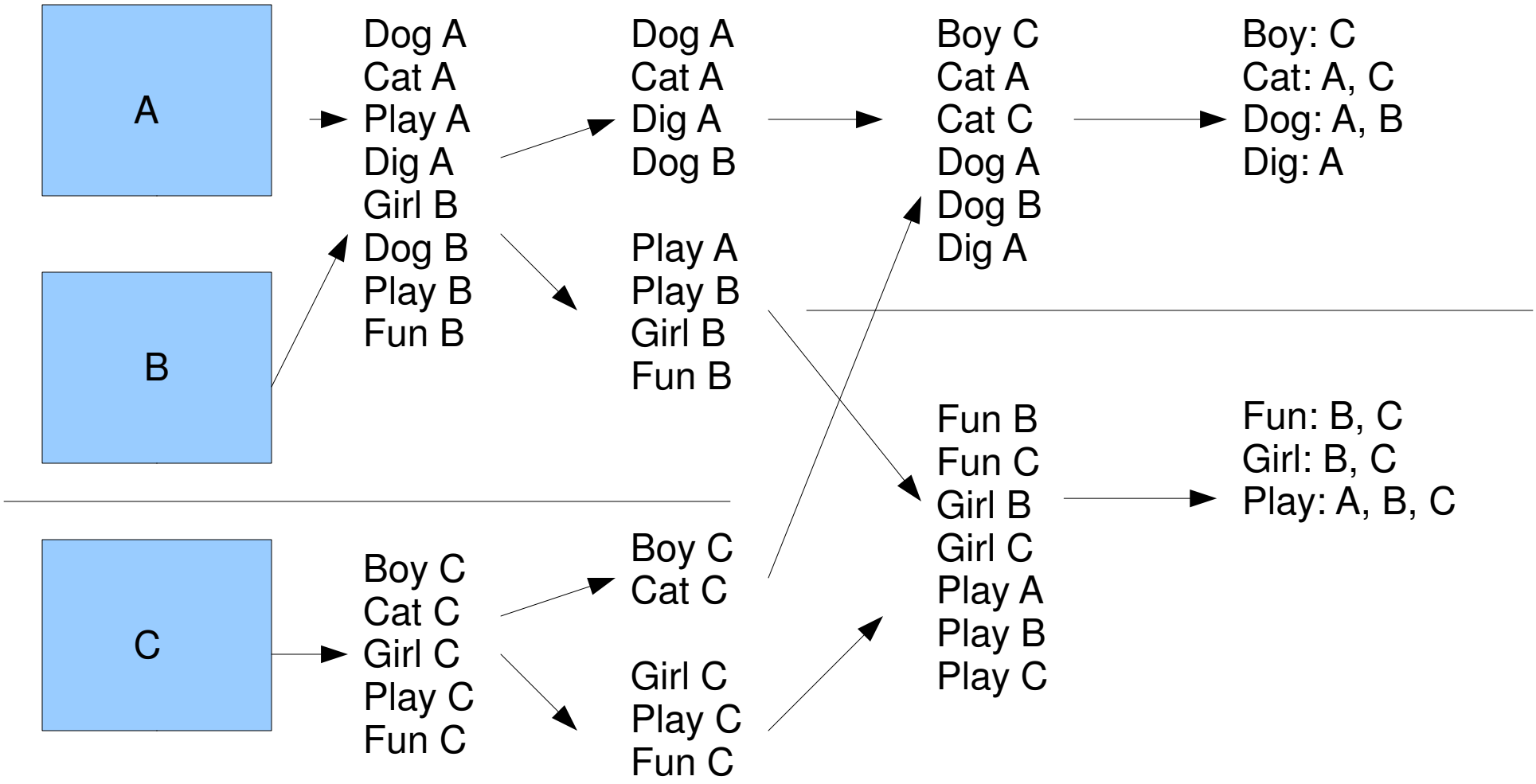
Time Series
Lessons Learned

- Often the Mapper only hashes so that each reducer gets selected data.
- The sort is essential to the Reducer's algorithm.
- Standard programming idiom: Watch for EOF or change in a parameter. If yes, output accumulated results and reset.
- OK to store locally modest amount of data.
  - Watch for data scaling problems
  - Here, a larger R = #Reducers helps
- Hadoop will schedule if R > # Machines avail.

Index a Document

- map(pageName, pageText)
  - For word w In pageText
    - Emit(w, pageName) to file(hash(w))
- Use ordered hash to output to reducers
- reduce(word, values)
  - For each pageName in values Until word changes Do AddToOutputList(pageName)
  - Emit(word, “: ”, pageNameList)

Example Word Index



Joining Data Sets

- Have two data types, one includes references to elements of the other; would like to incorporate data by value, not by reference
- MapReduce Needs Self-identifying records
- B = {b-id, pk, b-data}
- A = {a-id, fk, a-data}

Example (Inner) Join

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Wikipedia

Example (Inner) Join Result

Joined table

LastName	DepartmentID	DepartmentName
Rafferty	31	Sales
Jones	33	Engineering
Steinberg	33	Engineering
Robinson	34	Clerical
Smith	34	Clerical

Joining Data Sets 2/2

- Map(k, ...)
  - If k = a-id & fk is defined, then
    emit ((fk,a-id), a-data)
  - If k = b-id, then emit ((pk,b-id), b-data)
- Reduce (sort puts (pk,b-id) first then all the (equal 1st component) (fk,a-id)'s next, ...)
  - If pk, record b-data (e.g. in memory)
  - If fk, emit(a-data, fk, recorded b-data)

More Lessons Learned

- Sometimes significant work can be divided between Map and Reduce (Word Count and Index parse for words; Select and Search evaluate the predicate.
- Sometimes local storage is needed (Index and Join) be careful that this scales!
- Often self-identifying records are needed (Join) which need to be constructed early (Prepare).
- Next we'll see an example where multiple MR jobs are needed.

Multistage Pipeline
Term Frequency & Inverse Document Frequency

- t denotes term=word; d names a document in D
- tf(t,d) = number times t occurs in d
- wc(d) = number terms in d
- ndocs(t,D) = number of docs in D containing t
- totdocs(D) = number of docs in D
- idf(t,D) = log(totdocs(D)/ ndocs(t,D))
- tfidf(t,d,D) = tf(t,d)*idf(t,D)
- ASSUME t is in some d In D

tf*idf Job Pipeline

- Job 0 - compute N = totdocs(D)
- Job 1 - compute tf(t,d), pass N as a parameter
- Job 2 - compute wc(d) adding the tf(t,d) over t. Pass N, output all tf(t,d) under key “ktf”, and output all wc(d) under key “kwc”.
- Job 3 - compute ndocs(t,D) adding for each d, if tf(t,d)>0 then 1 else 0. Pass N, output all tf(t,d) and wc(d) under key “ktf”, and output ndocs(t,D) under key “kndocs”.
- Job 4 - calculate, for each (t,d) both idf(t,D) = log(totdocs(D)/ndocs(t,D)) and tfidf(t,d,D) then output((t,d),(tf(i,d),wd(d),ndocs(t,D),idf(t,d), tfidf(t,d,D)))

Job 0 - totdocs(D)

- Prepare sets up many mappers and one reducer
- map(d,1) = (D,N)
  - v := 0
  - Loop read(d, n); v += 1; End Loop
  - If EOF Then emit(D,v)
- reduce(X,v) = N - - =total docs in D = totdocs(D)
  - N := 0
  - Loop read(X, v); N += v; End Loop
  - If EOF Then emit(D,N) - - single element output

Job 1 – Term Frequency in Doc

- Job 1: Term/Word frequency and count
  - map((d, contents, N) = ((t, d), 1, N)
    - - - parse for t implicitly; N = totdocs(D)
    - - - hash so that each reducer gets all (t,d)'s together with no (t,d) splits across reducers
  - reduce((t, d), v, N) = ((t, d), n, N) - - n = tf(t,d)
    - For each (t,d) add up all the v=1's to get n
    - Emit((t,d), n, N)
    - Note the output (t,d)'s are unique and inclusive

Job 2: Word Counts for d in D

- Map((t,d), n, N) = (d,(t,n,N)) --n = tf(t,d), N = |D|
  - Hash the d's so that each reducer gets all the d values for the (t,d)
- Reduce(d,(t, n, N)) = (k, (t, d, tf(t,d), wc(d), m, N))
  - For each d, v := 0; Loop over t,
    - k:= “ktf”; m:= If n>0 Then 1 Else 0
    - emit(k, (t, d, n, 0, m, N)) - - pass on the n=tf(t,d) info & m
    - v += n - - n = tf(t,d) - - Note, no need to parse d again!
    - End Loop over t - - now v=wc(d)
  - k:= “kwc”
  - emit(k, (“”,d, 0, v, 0, N))
  - End Loop on d

Job 3 – ndocs(t,D)

- Pass each reducer output file back to a job3 mapper
- map(k,(t, d, n, w, m, N)) = identity

  - -if k= “ktf”, n=tf(t,d) and m=1 if tf(t,d)>0 else 0 & w = wc(d) if k = “kwc” & N = totdocs(D)

- Again hash d so that no d's are split across reducers, and sort so that “kwc” is before “ktf” and all d's are together.
- reduce(k, (t, d, n, w, m, N)) = (k, (t,d,n,w,v,N))
  - If k = “kwc” Then
    - wc := w
    - Skip to next record - - wc=wc(d)

Job 3 continued = ndocs(t,D)

- If k = “ktf” Then For each t, v:=0; Loop over d
    - emit(k, (t,d,n,wc,0,N)) - - Note wc is folded in!
    - v += m - - add all the m's to get ndocs(t,D)
    - End Loop over d
  - k := “kndocs”
  - emit(k, (t, “”, 0, 0, v, N)) - - v = ndocs(t,D)
  - End For each t

Job 4 – Calculate idf(t,D) and tf*idf(t,d,D)

- Map(k,(t,d,n,w,v,N)) = identity
  - Hash d's so the (t,d)'s aren't split across reducers; sort so that “kndocs” is first and all d's are together.
- Reduce(k,(t,d,n,w,v,N))
  - Loop
  - Read(k, (t,d,n,w,v,N)
  - If k = “kndocs” Then ndocs:=v; Skip - - ndocs(t,D)
  - If k = “ktf” Then tf := n; idf := log(N/ndocs); tfidf := tf*idf - - tf(t,d), idf(t,D), ndocs(t,D), tfidf(t,d,D)
  - Output ((t, d), (tf,idf,wc,ndocs,N,tfidf)
  - End Loop

Summary TF-IDF

- Several small jobs add up to a full algorithm
- Lots of code reuse is possible – stock classes exist for aggregation, identity, etc. Standard idioms for code.
- Job 0 is not necessary if card(D) is known
- Needed self-identifying records
- There are other ways to do tf*idf, e.g. using external files.
- Same techniques to iterate and converge.

Consulting and Careers

- “Data Scientist” “Hadoop Administrator”- new job titles
- IBM's Big Data University (2,400 at their bootcamps, 14,000 registered for on-line)
- Hadoop “certification” (Cloudera, IBM, Hortonworks, MapR, Informatica)
- 3rd Hadoop World (Oct 23-25, 2012 NYC)
- A deluge of Hadoop activities in Silicon Valley
- Consult on using Google Apps
- Performance experts needed!!!
- Security experts needed!!!
- Analysis/algorithms experts needed !!!

Thank You!

- Questions???