Apache Hadoop & a new paradigm for data processing

Doug Cutting

the opportunity

- data accumulating faster than ever
- storage, CPU & network cheaper than ever
- but conventional database tech
  - isn't priced at commodity hardware prices
  - doesn't scale well to thousands of CPUs & drives

problem: scaling reliably is hard

- need to store petabytes of data
  - on 1000s of nodes, MTBF < 1 day
  - something is always broken
- need fault tolerant store
  - handle hardware faults transparently and efficiently
  - provide availability
- need fault-tolerant computing framework
  - even on a big cluster, some things take days

problem: bandwidth to data

- need to process 100TB dataset
- on 1000 node cluster reading from LAN
  - 100Mb all-to-all bandwidth
  - scanning @ 10MB/s = 165 min
- on 1000 node cluster reading from local drives
  - scanning @ 200MB/s = 8 min
- moving computation beats moving data

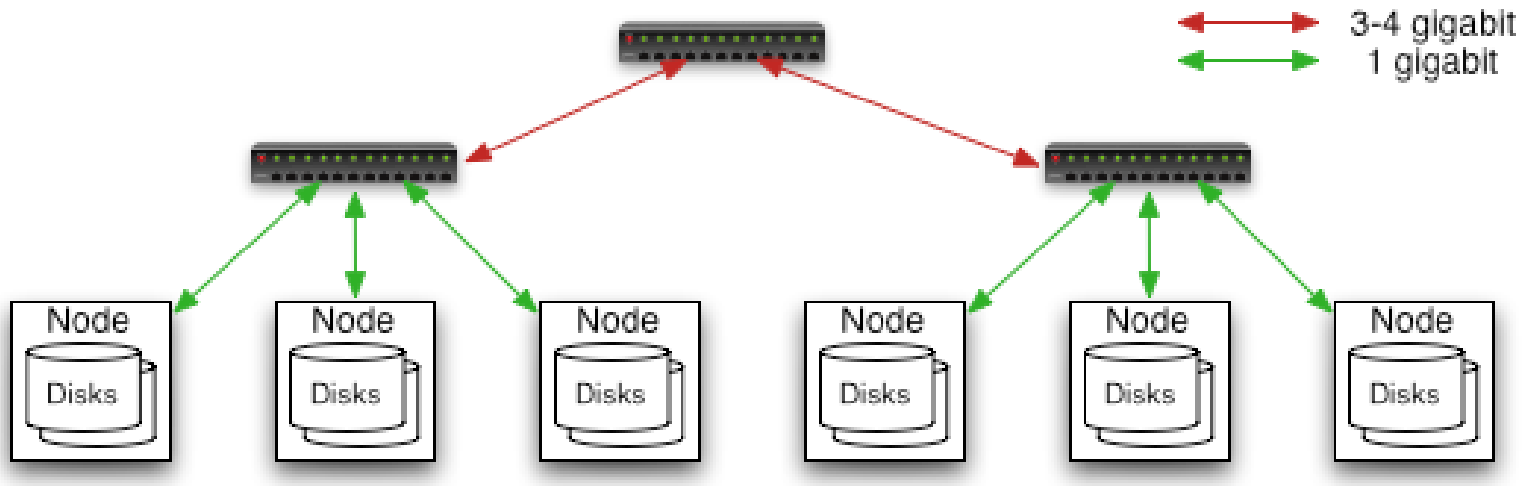
Apache Hadoop: a new paradigm

- scales to thousands of commodity computers
- can effectively use all cores & spindles
  - simultaneously to process your data
- new software stack
  - built on a different foundation
- in use already by many
  - most big web 2.0 companies
  - many Fortune 500 companies now too

new foundations

- commodity hardware
- sequential file access
- sharding of data & computation
- automated, high-level reliability
- open source

commodity hardware



- typically in 2-level architecture
  - nodes are commodity PCs
  - 30-40 nodes/rack
- offers linear scalabilty
  - at commodity prices

how I got here

- started in the 80's, building full-text indexes
- first implemented with a B-Tree
  - foundation of relational DBs
  - log(n) random accesses per update
  - seek time is wasted time
- too slow when updates are frequent
  - instead use batched sort/merge
    - to index web at Excite in 90's

B-Tree

- foundation of RDBMS
- designed around random-access
  - log(db_size) seeks per access
  - n*log(n) seeks to build an index

seek versus transfer

- seeks are wasted time
- sequential access has no seeks
- w/o seeks, data is processed faster
- even with SSD

Lucene (2000):

- open-source full-text search library
- sorts batches of updates
- then merges with previously sorted data
- only n/k seeks to build an index
- compared to B-Tree
  - much less wasted time

open source

- Apache
  - supports diverse collaborative communities
- as a developer, I like it because it...
  - doesn't disappear
  - is what it is
- users like it because...
  - the price is right
  - transparency of product and process

open source

- companies like it because...
  - better code
    - publication encourages quality
    - sharing encourages generality
  - happier employees
    - respect from wider peer pool
  - costs shared with collaborators
    - QA, documentation, features, support, training, etc.

Nutch (2002)

- open-source web search engine
- db access per link in crawled page
  - monthly crawl requires >10k accesses/second
- sort/merge optimization applicable
  - but distributed solution required

Nutch (2002)

- distribute by *sharding* URL space
  - for N nodes, url goes to node hash(url) % N
- batch-based
  - split updates into file per shard
  - copy shard updates to shard's node
  - merge updates w/ existing db there
- steps performed manually
- begged for automation!

Nutch (2004)

- Google publishes GFS & MapReduce papers
- together, provide automation of
  - sort/merge+sharding
  - reliability
- we then implemented these in Nutch

Hadoop (2006)

- Yahoo! joins the effort
- split HDFS and MapReduce from Nutch

HDFS

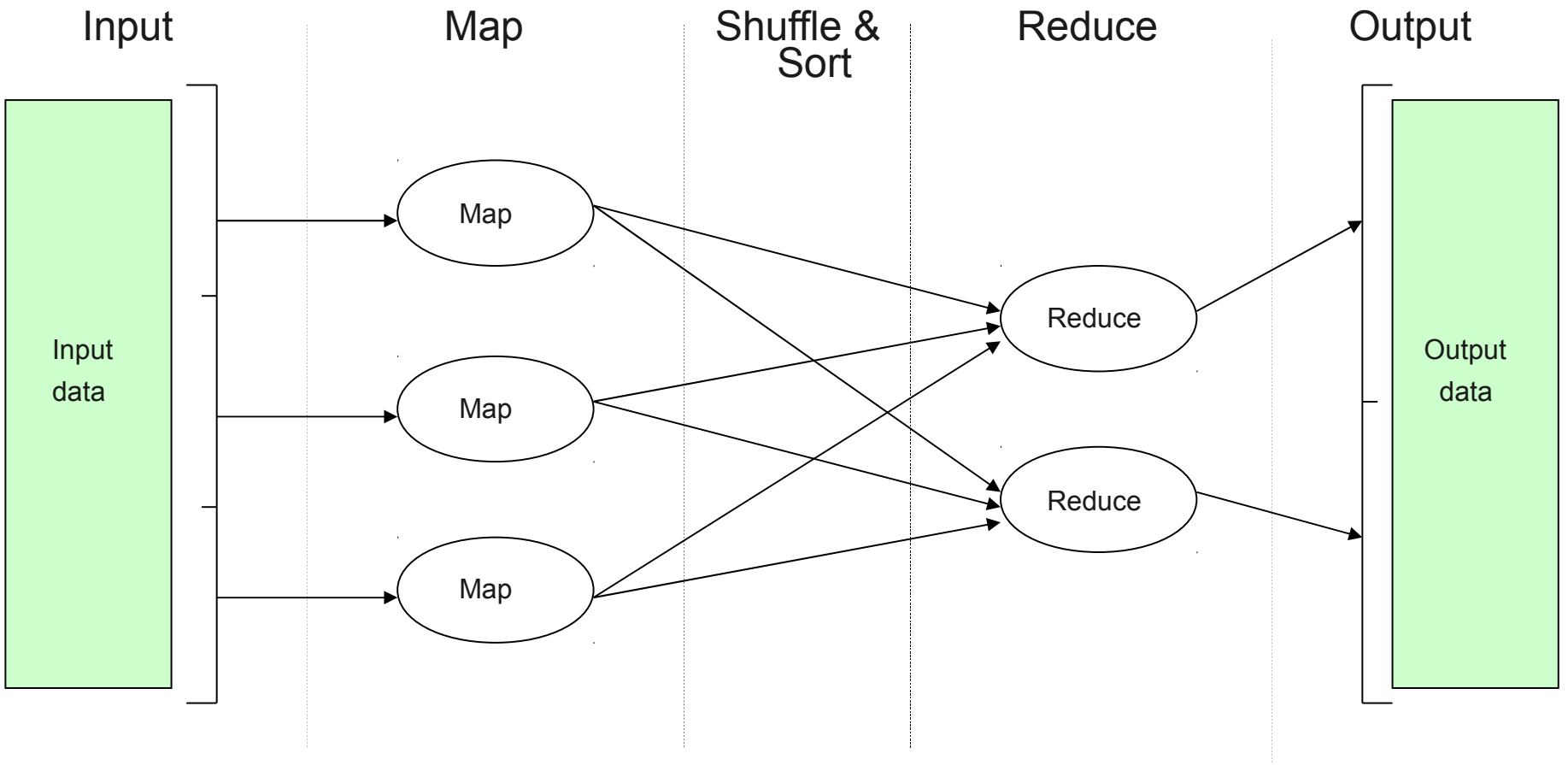
- namenode:
  - filename → blockid*
  - blockid → datanode*
- datanode:
  - blockid → byte*
- open a file:
  - talk to namenode
- read/write data:
  - talk to a datanode

HDFS

- scales well
  - files sharded across commodity hardware
  - efficient and inexpensive
- automated reliability
  - each block replicated on 3 datanodes
  - automatically rebalances, replicates, etc.
  - namenode can have hot spare

MapReduce

- simple programming model
  - generalizes common pattern

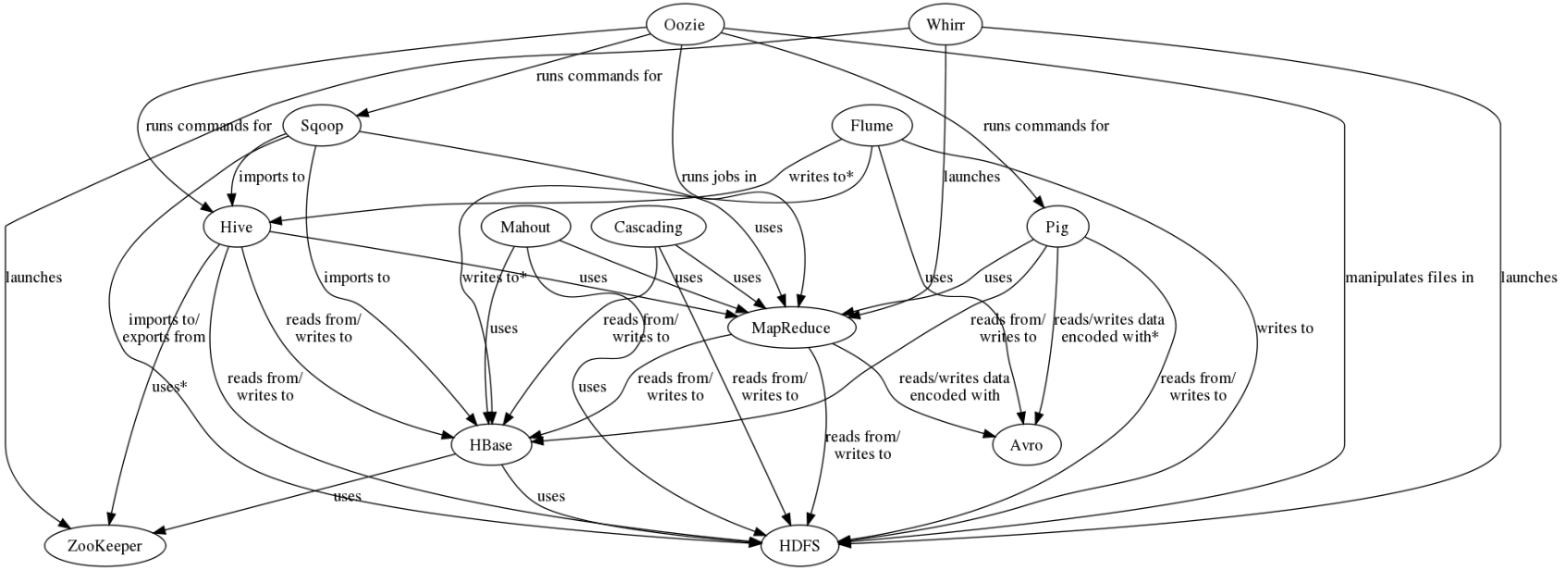


MapReduce

- compute on same nodes as HDFS storage
  - i/o on every drive
  - compute on every core
  - massive throughput
- sequential access
  - directly supports sort/merge
- automated reliability & scaling
  - datasets are sharded to tasks
  - failed tasks retried

Hadoop: the ecosystem

- active, growing, community
  - multiple books in print
  - commercial support available
  - expanding network of complementary tools



Pig & Hive

- higher-level query languages
  - that generate MapReduce jobs
- Pig has imperative dataflow language
  - often used for ETL
- Hive uses SQL
  - often used for data warehouse

Avro

- common data format
  - expressive schema language
    - supports evolution
  - efficient binary encoding
  - self-describing file format
  - RPC
  - Java, C, C++, Python, Ruby, PHP
- works with MapReduce

Mahout

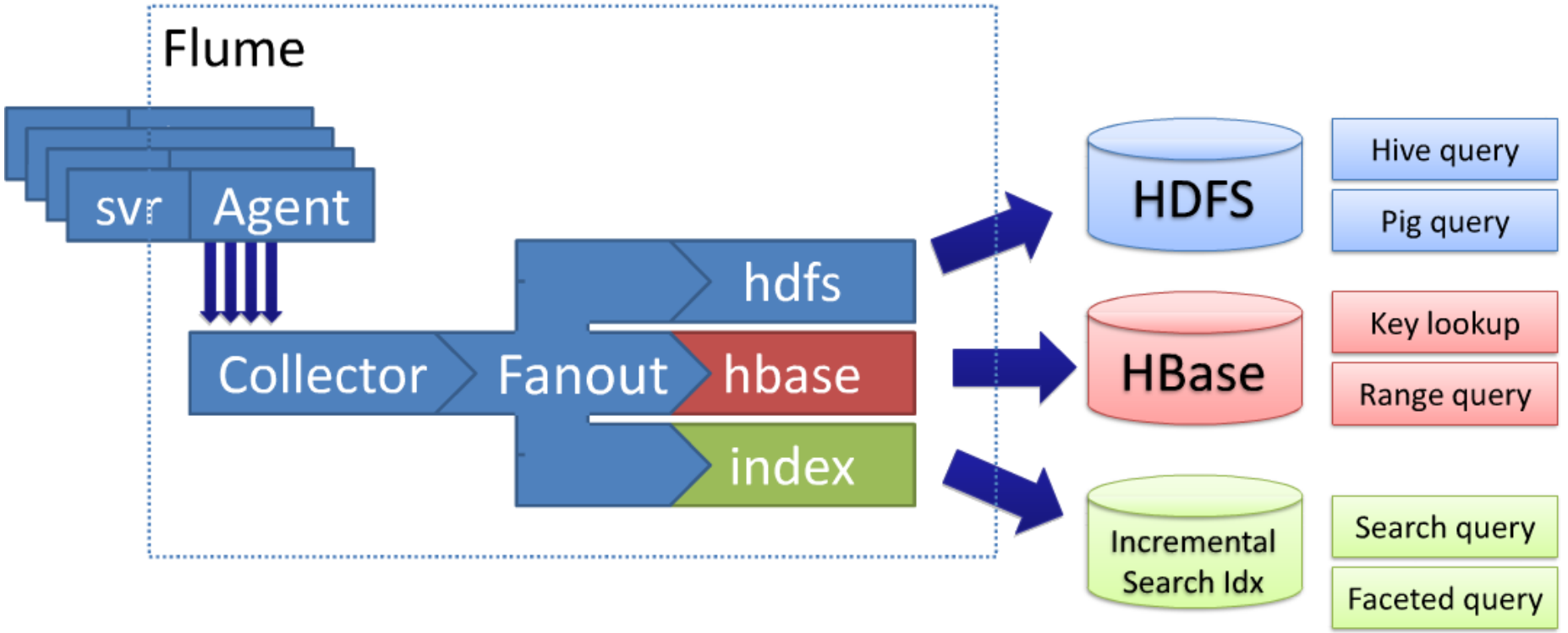
- scalable machine learning library
- includes algorithms for
  - classification
  - clustering
  - collaborative filtering / recommendation
- most use MapReduce

HBase

- inspired by Google's BigTable
- real-time DB
  - data in HDFS
  - access by primary key, or scan
    - no indexes built in
  - massive throughput
  - works with MapReduce

Flume

- data collection framework
  - distributed, reliable, available



pattern of adoption

- initially adopt Hadoop for particular application
  - for cost effective scaling
- then load more datasets & add more users
  - & find it more valuable for new, unanticipated apps
- having all data in one place, usable, empowers
- “We don't use Hadoop because we have a lot of data, we have a lot of data because we use Hadoop.”

key advantages of paradigm

- cost-effective
  - scales linearly on commodity hardware
- general purpose
  - powerful, easy-to-program
- low barrier to entry
  - no schema or DB design required up-front
  - just load raw data & use it

Thank you!

Questions?