HDFS Design Principles

The Scale-out-Ability of Distributed Storage

SVForum

Software Architecture & Platform SIG

Konstantin V. Shvachko

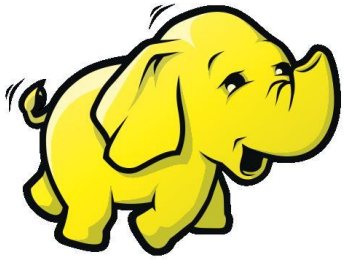
May 23, 2012

Big Data

- Computations that need the power of many computers
  - Large datasets: hundreds of TBs, tens of PBs
  - Or use of thousands of CPUs in parallel
  - Or both
- Big Data management, storage and analytics
  - Cluster as a computer

What is Apache Hadoop

- Hadoop is an ecosystem of tools for processing "Big Data"



- Hadoop is an open source project

The Hadoop Family

HDFS	Distributed file system
MapReduce	Distributed computation
Zookeeper	Distributed coordination
HBase	Column store
Pig	Dataflow language, SQL
Hive	Data warehouse, SQL
Oozie	Complex job workflow
BigTop	Packaging and testing

Hadoop: Architecture Principles

- Linear scalability: more nodes can do more work within the same time
  - Linear on data size:
  - Linear on compute resources:
- Move computation to data
  - Minimize expensive data transfers
  - Data are large, programs are small
- Reliability and Availability: Failures are common
  - 1 drive fails every 3 years **=>** Probability of failing today 1/1000
  - How many drives per day fail on 1000 node cluster with 10 drives per node?
- **Simple computational model**
  - hides complexity in efficient execution framework
- Sequential data processing (avoid random reads)

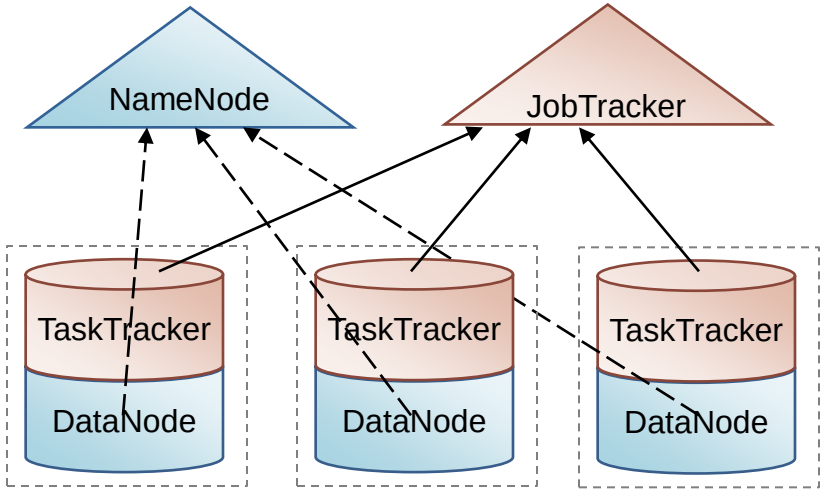
Hadoop Core

- A reliable, scalable, high performance distributed computing system
- The Hadoop Distributed File System (HDFS)
  - Reliable storage layer
  - With more sophisticated layers on top
- MapReduce – distributed computation framework
- Hadoop scales computation capacity, storage capacity, and I/O bandwidth by adding commodity servers.
- Divide-and-conquer using lots of commodity hardware

Hadoop Cluster Components

- HDFS – a distributed file system
  - NameNode – namespace and block management
  - DataNodes – block replica container
- MapReduce – a framework for distributed computations
  - JobTracker – job scheduling, resource management, lifecycle coordination
  - TaskTracker – task execution module

.

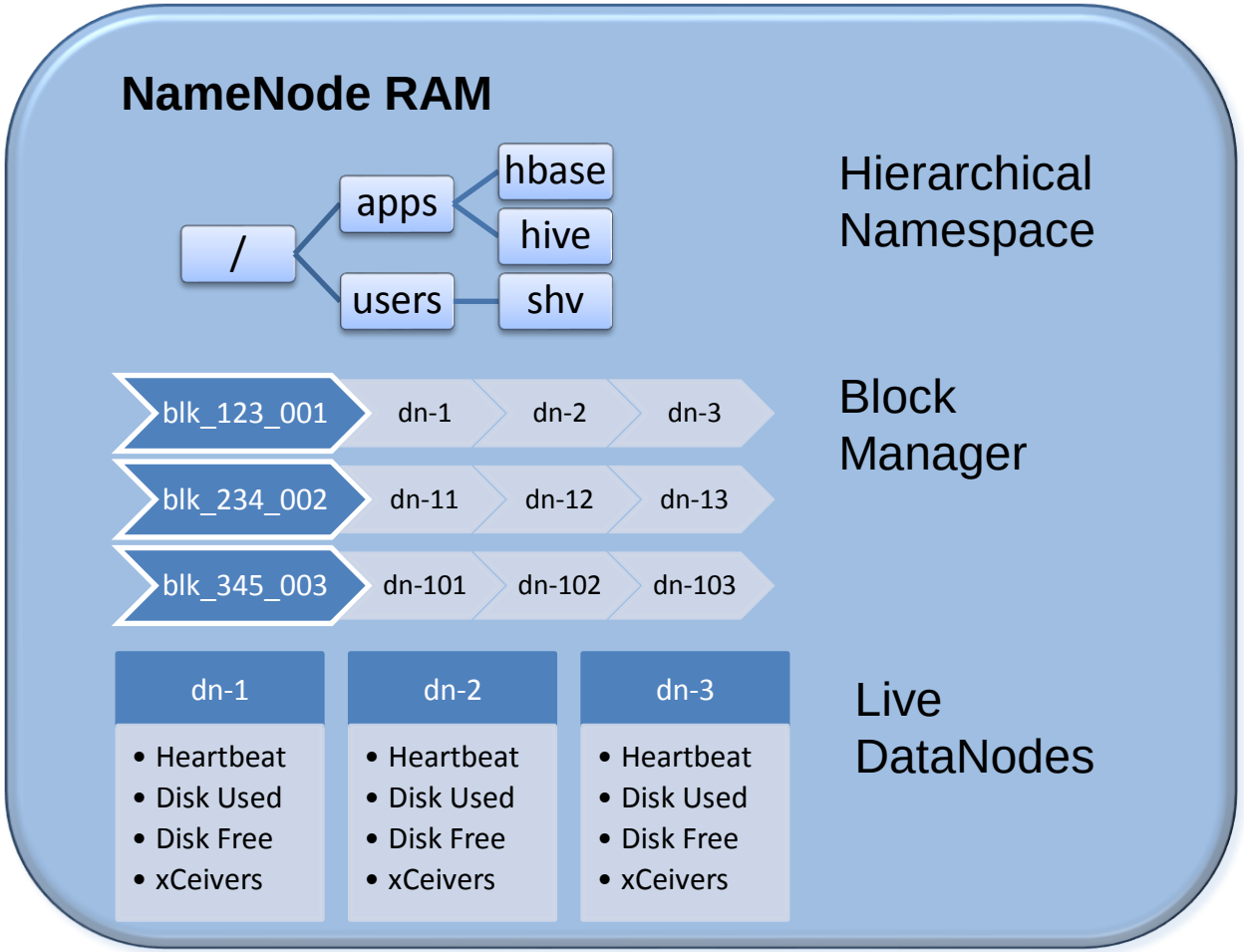


HDFS
THE DESIGN PRINCIPLES

Hadoop Distributed File System

- The name space is a hierarchy of files and directories
- Files are divided into blocks (typically 128 MB)
- Namespace (metadata) is decoupled from data
  - Lots of fast namespace operations, not slowed down by
  - Data streaming
- Single NameNode keeps the entire name space in RAM
- DataNodes store block replicas as files on local drives
- Blocks are replicated on 3 DataNodes for *redundancy and availability*

NameNode Transient State



NameNode Persistent State

- The durability of the name space is maintained by a write-ahead ***journal*** and ***checkpoints***
  - Journal transactions are persisted into ***edits*** file before replying to the client
  - Checkpoints are periodically written to ***fsimage*** file
    Handled by Checkpointer, SecondaryNameNode
  - Block locations discovered from DataNodes during startup via **block reports**.
    Not persisted on NameNode
- Types of persistent storage devices
  - Local hard drive
  - Remote drive or NFS filer
  - BackupNode
- Multiple storage directories
  - Two on local drives, and one remote server, typically NFS filer

DataNodes

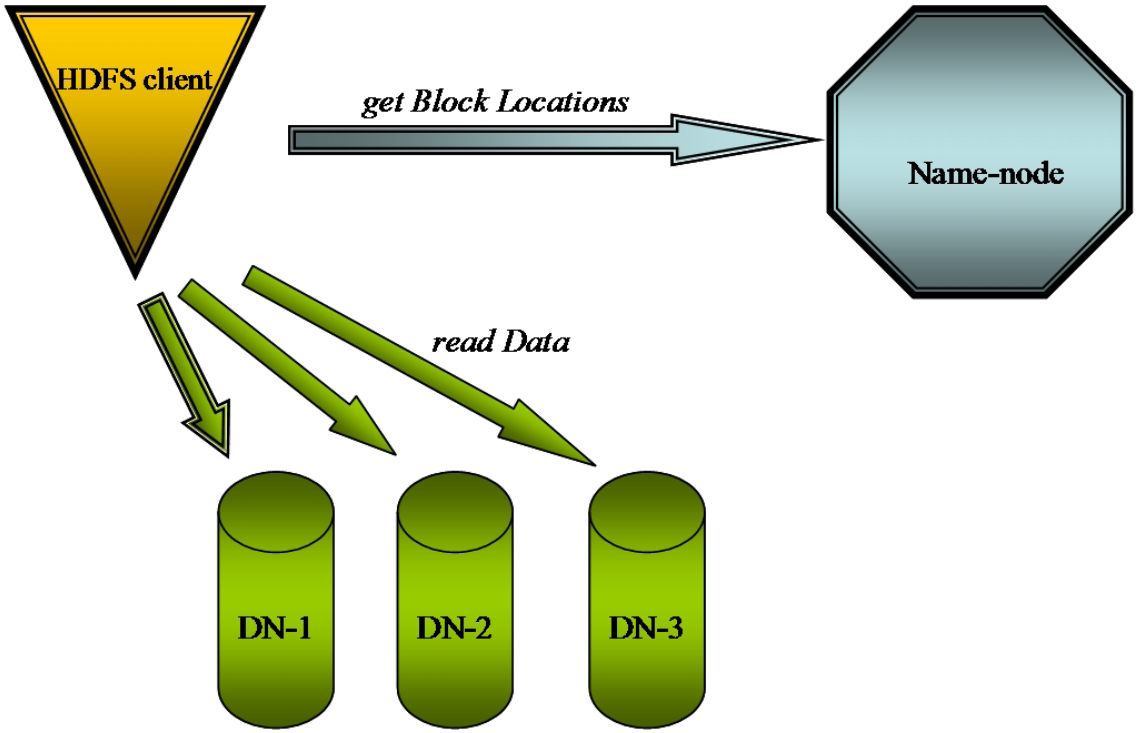
- DataNodes register with the NameNode, and provide periodic block reports that list the block replicas on hand
  - block report contains *block id, generation stamp* and *length* for each replica
- DataNodes send heartbeats to the NameNode to confirm its alive: 3 sec
- If no heartbeat is received during a 10-minute interval, the node is presumed to be lost, and the replicas hosted by that node to be unavailable
  - NameNode schedules re-replication of lost replicas
- Heartbeat responses give instructions for managing replicas
  - Replicate blocks to other nodes
  - Remove local block replicas
  - Re-register or shut down the node
  - Send an urgent block report

HDFS Client

- Supports conventional file system operation
  - Create, read, write, delete files
  - Create, delete directories
  - Rename files and directories
- Permissions
- Modification and access times for files
- Quotas: 1) namespace, 2) disk space
- Per-file, when created
  - Replication factor – can be changed later
  - Block size – can not be reset after creation
- Block replica locations are exposed to external applications

HDFS Read

- To read a block, the client requests the list of replica locations from the NameNode
- Then pulling data from a replica on one of the DataNodes

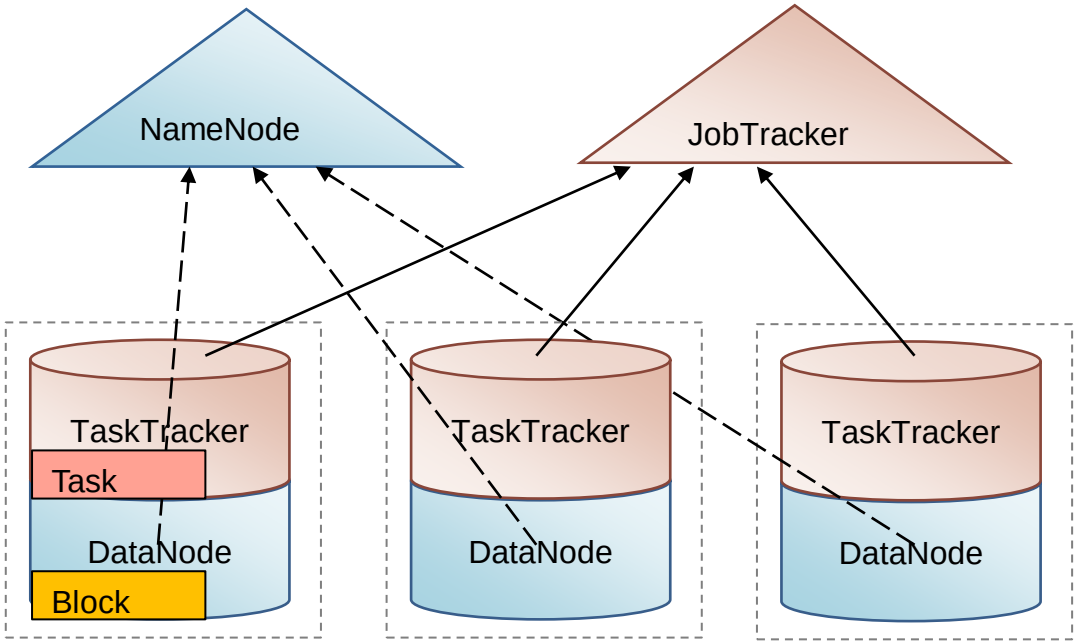


HDFS Read

- Open file returns DFSInputStream
- DFSInputStream for a current block fetches replica locations from NameNode
  - 10 at a time
  - Client caches replica locations
- Replica Locations are sorted by their proximity to the client
  - Choose first location
- Open a socket stream to chosen DataNode, read bytes from the stream
- If fails add to dead DataNodes
  - Choose the next DataNode in the list
- Retry 2 times

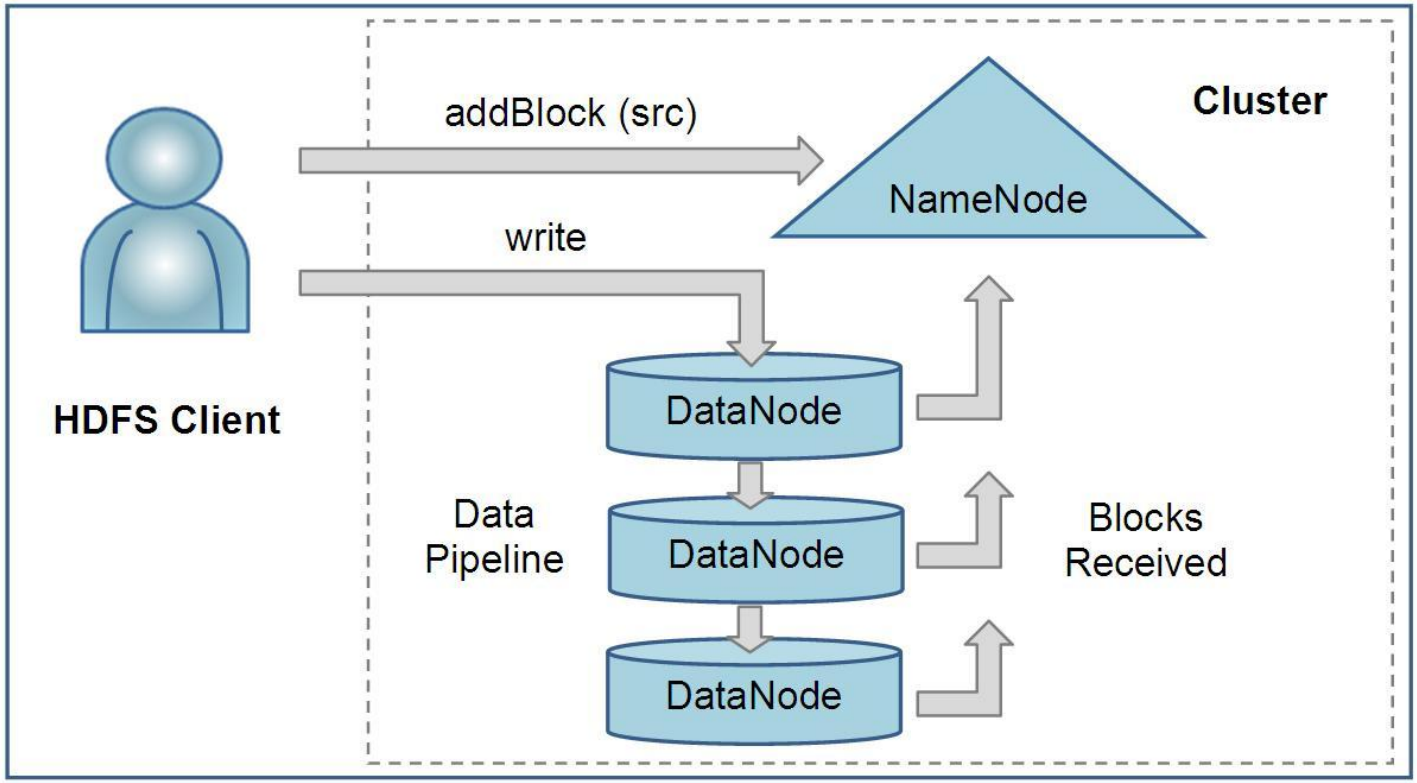
Replica Location Awareness

- MapReduce schedules a task assigned to process block B to a DataNode serving a replica of B
- Local access to data



HDFS Write

- To write a block of a file, the client requests a list of candidate DataNodes from the NameNode, and organizes a write pipeline.



HDFS Write

- Create file in the namespace
- Call addBlock() to get next block
  - NN returns prospective replica locations sorted by proximity to the client
  - Client creates a pipeline for streaming data to DataNodes
- HDFS client writes into internal buffer and forms a queue of Packets
- DataStreamer sends a packet to DN1 as it becomes available
  - DN1 streams to DN2 the same way, and so on
- If one node fails the pipeline is recreated with remaining nodes
  - Until at least one node remains
  - Replication is handled later by the NameNode

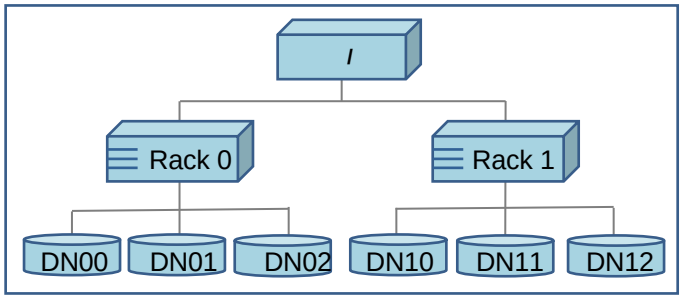
Write Leases

- HDFS implements a single-writer, multiple-reader model.
- HDFS client maintains a lease on files it opened for write
  - Only one client can hold a lease on a single file
  - Client periodically renews the lease by sending heartbeats to the NameNode
- Lease expiration:
  - Until soft limit expires client has exclusive access to the file
  - After **soft limit** (10 min): any client can reclaim the lease
  - After **hard limit** (1 hour): NameNode mandatory closes the file, revokes the lease
- Writer's lease does not prevent other clients from reading the file

Append to a File

- Original implementation supported write-once semantics
  - After the file is closed, the bytes written cannot be altered or removed
- Now files can be modified by reopening for append
- Block modifications during appends use the copy-on-write technique
  - Last block is copied into temp location and modified
  - When “full” it is copied into its permanent location
- HDFS provides consistent visibility of data for readers before file is closed
- *hflush* operation provides the visibility guarantee
  - On *hflush* current packet is immediately pushed to the pipeline
  - hflush waits until all DataNodes successfully receive the packet
- *hsync* also guarantees the data is persisted to local disks on DataNodes

Block Placement Policy



- Cluster topology
  - Hierarchal grouping of nodes according to
  - network distance
- Default block placement policy - a tradeoff
  - between minimizing the write cost,
  - and maximizing data reliability, availability and aggregate read bandwidth

1. *First replica on the local to the writer node*
2. *Second and the Third replicas on two different nodes in a different rack*
3. *the Rest are placed on random nodes with restrictions*
   - no more than one replica of the same block is placed at one node and
   - no more than two replicas are placed in the same rack (if there is enough racks)

- HDFS provides a configurable block placement policy interface
  - experimental

System Integrity

- **Namespace ID**
  - a unique cluster id common for cluster components (NN and DNs)
  - Namespace ID assigned to the file system at format time
  - Prevents DNs from other clusters join this cluster
- **Storage ID**
  - DataNode persistently stores its unique (per cluster) Storage ID
  - makes DN recognizable even if it is restarted with a different IP address or port
  - assigned to the DataNode when it registers with the NameNode for the first time
- **Software Version** (build version)
  - Different software versions of NN and DN are incompatible
  - Starting from Hadoop-2: version compatibility for rolling upgrades
- Data integrity via **Block Checksums**

Cluster Startup

- NameNode startup
  - Read image, replay journal, write new image and empty journal
  - Enter SafeMode
- DataNode startup
  - Handshake: check Namespace ID and Software Version
  - Registration: NameNode records Storage ID and address of DN
  - Send initial block report
- SafeMode – read-only mode for NameNode
  - Disallows modifications of the namespace
  - Disallows block replication or deletion
  - Prevents unnecessary block replications until the majority of blocks are reported
  - Minimally replicated blocks
  - SafeMode threshold, extension
  - Manual SafeMode during unforeseen circumstances

Block Management

- Ensure that each block always has the intended number of replicas
  - Conform with block placement policy (BPP)
  - Replication is updated when a block report is received
- **Over-replicated blocks**: choose replica to remove
  - Balance storage utilization across nodes without reducing the block's availability
  - Try not to reduce the number of racks that host replicas
  - Remove replica from DataNode with longest heartbeat or least available space
- **Under-replicated blocks**: place into the replication priority queue
  - Less replicas means higher in the queue
  - Minimize cost of replica creation but concur with BPP
- **Mis-replicated block**, not to BPP
- **Missing block**: nothing to do
- **Corrupt block**: try to replicate good replicas, keep corrupt replicas as is

HDFS Snapshots: Software Upgrades

- Snapshots prevent from data corruption/loss during software upgrades
  - allow to rollback if software upgrades go bad
- Layout Version
  - identifies the data representation formats
  - persistently stored in the NN's and the DNs' storage directories
- NameNode image snapshot
  - Start `hadoop namenode –upgrade`
  - Read checkpoint image and journal based on persisted Layout Version
    - New software can always read old layouts
  - Rename `current` storage directory to `previous`
  - Save image into new `current`
- DataNode upgrades
  - Follow NameNode instructions to upgrade
  - Create a new storage directory and hard link existing block files into it

Administration

- Fsck verifies
  - Missing blocks, block replication per file
  - Block placement policy
  - Reporting tool, does not fix problems
- Decommission
  - Safe removal of a DataNode from the cluster
  - Guarantees replication of blocks to other nodes from the one being removed
- Balancer
  - Rebalancing of used disk space when new empty nodes are added to the cluster
  - Even distribution of used space between DataNodes
- Block Scanner
  - Block checksums provide data integrity
  - Readers verify checksums on the client and report errors to the NameNode
  - Other blocks periodically verified by the scanner

Hadoop Size

- Y! cluster 2010
  - 70 million files, 80 million blocks
  - 15 PB capacity
  - 4000+ nodes. 24,000 clients
  - 50 GB heap for NN
- Data warehouse Hadoop cluster at Facebook 2010
  - 55 million files, 80 million blocks. Estimate 200 million objects (files + blocks)
  - 2000 nodes. 21 PB capacity, 30,000 clients
  - 108 GB heap for NN should allow for 400 million objects
- Analytics Cluster at eBay (Ares)
  - 77 million files, 85 million blocks
  - 1000 nodes: 24 TB of local disk storage, 72 GB of RAM, and a 12-core CPU
  - Cluster capacity 19 PB raw disk space
  - Runs upto 38,000 MapReduce tasks simultaneously

Benchmarks

- **DFSIO**

Throughput MB/sec	Read	Write	Append
Hadoop-0.22	100	84	83
Hadoop-1.*	96	66	n/a

- **Observed on busy cluster**
  - Read: 1.02 MB/s
  - Write: 1.09 MB/s

- **TeraSort ("Very carefully tuned user application")**

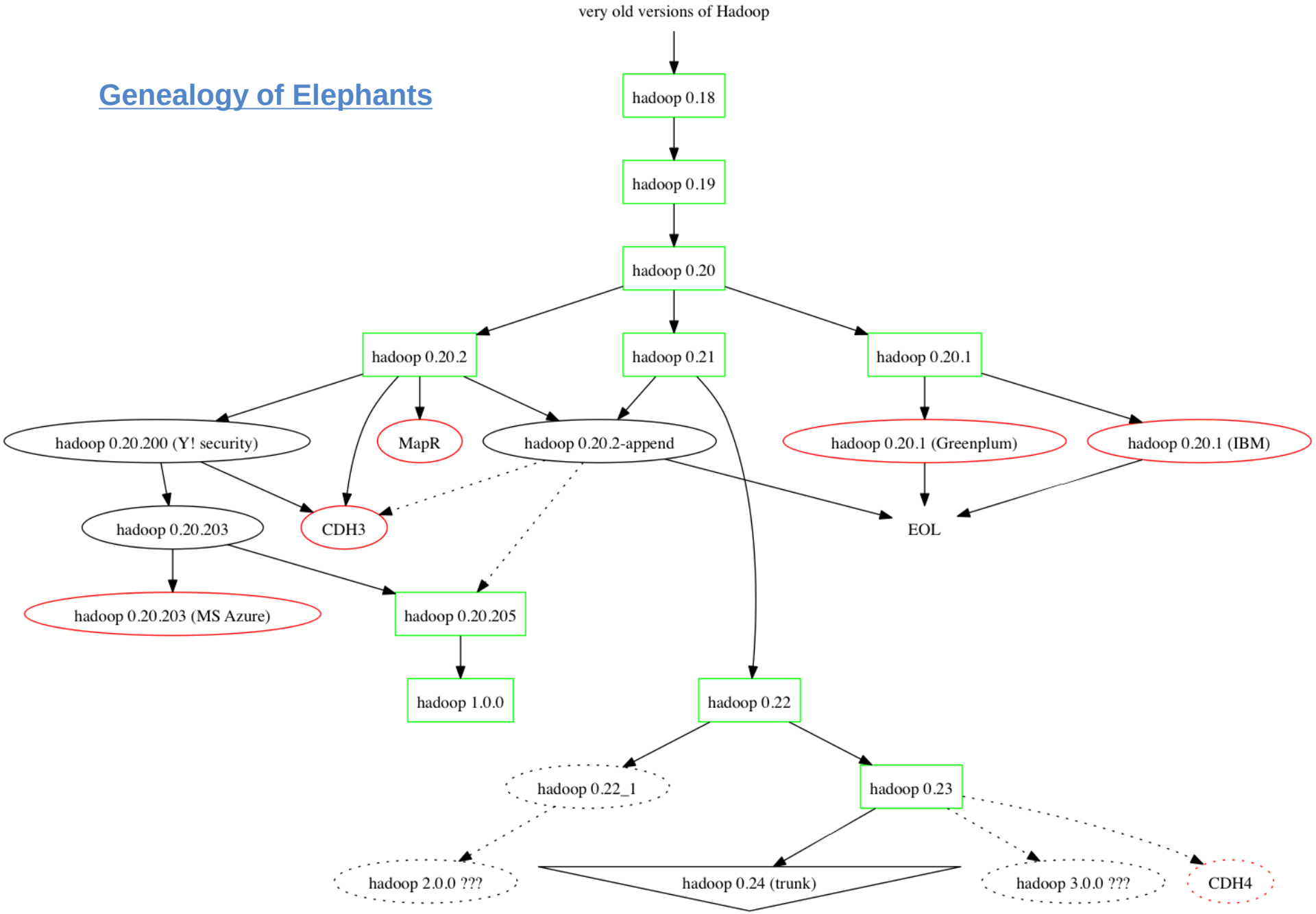
Bytes (TB)	Nodes	Maps	Reduces	Time	HDFS I/O Bytes/s	
					Aggregate (GB/s)	Per Node (MB/s)
1	1460	8000	2700	62 s	32	22.1
1000	3558	80,000	20,000	58,500 s	34.2	9.35

WHAT IS NEXT

The Future: Hadoop 2

- HDFS Federation
  - Independent NameNodes sharing a common pool of DataNodes
  - Cluster is a family of volumes with shared block storage layer
  - User sees volumes as isolated file systems
  - ViewFS: the client-side mount table
  - Federated approach provides a static partitioning of the federated namespace
- High Availability for NameNode
- Next Generation MapReduce
  - Separation of JobTracker functions
  1. Job scheduling and resource allocation
     - *Fundamentally centralized*
  2. Job monitoring and job life-cycle coordination
     - *Delegate coordination of different jobs to other nodes*
  - Dynamic partitioning of cluster resources: no fixed slots

Genealogy of Elephants



Major Hadoop Versions

- Hadoop 1.0.3 (security branch) 2012-05-16
  - Security, HBase support, No append
  - Stable
- Hadoop 0.22.0 2011-12-10
  - Full append support, symlinks, BackupNode, Disk-fail-in-place, File concatenation
  - Beta
- Hadoop 2 2012-05-23
  - Fedaration – static partitioning of HDFS namespace
  - Yarn – new implementation of MapReduce
  - HA (manual failover)
  - Alpha
- No stable unifying release, containing all the good features

AVAILABILITY

Why High Availability is Important?

- Nothing is perfect:
  - Applications and servers crash
  - Avoid downtime
- Conventional for traditional RDB and enterprise storage systems
- Industry standard requirement

And Why it is Not?

- Scheduled downtime dominates Unscheduled
  - OS maintenance
  - Configuration changes
- Other reasons for Unscheduled Downtime
  - 60 incidents in 500 days on 30,000 nodes
  - 24 Full GC – the majority
  - System bugs / Bad application / Insufficient resources
  - “Data Availability and Durability with HDFS”
    R. J. Chansler USENIX ;login: February, 2012
- Pretty reliable

NameNode HA Challenge

- Naïve Approach
  - Start new NameNode on the spare host, when the primary NameNode dies:
  - Use LinuxHA or VCS for failover
- Not 

- NameNode startup may take up to 1 hour
  - Read the Namespace image and the Journal edits: 20 min
  - Wait for block reports from DataNodes (SafeMode): 30 min

Failover Classification

- ***Manual-Cold*** (or no-HA) – an operator manually shuts down and restarts the cluster when the active NameNode fails.
- ***Automatic-Cold*** – save the namespace image and the journal into a shared storage device, and use standard HA software for failover.
  It can take up to an hour to restart the NameNode.
- ***Manual-Hot*** – the entire file system metadata is fully synchronized on both active and standby nodes, operator manually issues a command to failover to the standby node when active fails.
- ***Automatic-Hot*** – the real HA, provides fast and completely automated failover to the hot standby.
- ***Warm HA*** – BackupNode maintains up to date namespace fully synchronized with the active NameNode. BN rediscovers location from DataNode block reports during failover. May take 20-30 minutes.

HA: State of the Art

- Manual failover is a routine maintenance procedure: Hadoop Wiki
- Automatic-Cold HA first implemented at ContextWeb
  uses DRBD for mirroring local disk drives between two nodes and Linux-HA as the failover engine
- AvatarNode from Facebook - a manual-hot HA solution.
  Use for planned NameNode software upgrades without down time
- Five proprietary installations running hot HA
- Designs:
  - HDFS-1623. High Availability Framework for HDFS NN
  - HDFS-2064. Warm HA NameNode going Hot
  - HDFS-2124. NameNode HA using BackupNode as Hot Standby
- Current implementation Hadoop-2
  - Manual failover with shared NFS storage
  - In progress: no NFS dependency, automatic failover based on internal algorithms

Automatic-Hot HA: the Minimalistic Approach

- Standard ***HA software***
  - LinuxHA, VCS, Keepalived
- ***StandbyNode***
  - keeps the up-to-date image of the namespace via Journal stream
  - available for read-only access
  - can become active NN
- ***LoadReplicator***
  - DataNodes send heartbeats / reports to both NameNode and StandbyNode
- ***VIPs*** are assigned to the cluster nodes by their role:
  - NameNode – nn.vip.host.com
  - StandbyNode – sbn.vip.host.com
- ***IP-failover***
  - Primary node is always the one that has the NameNode VIP
  - Rely on proven HA software

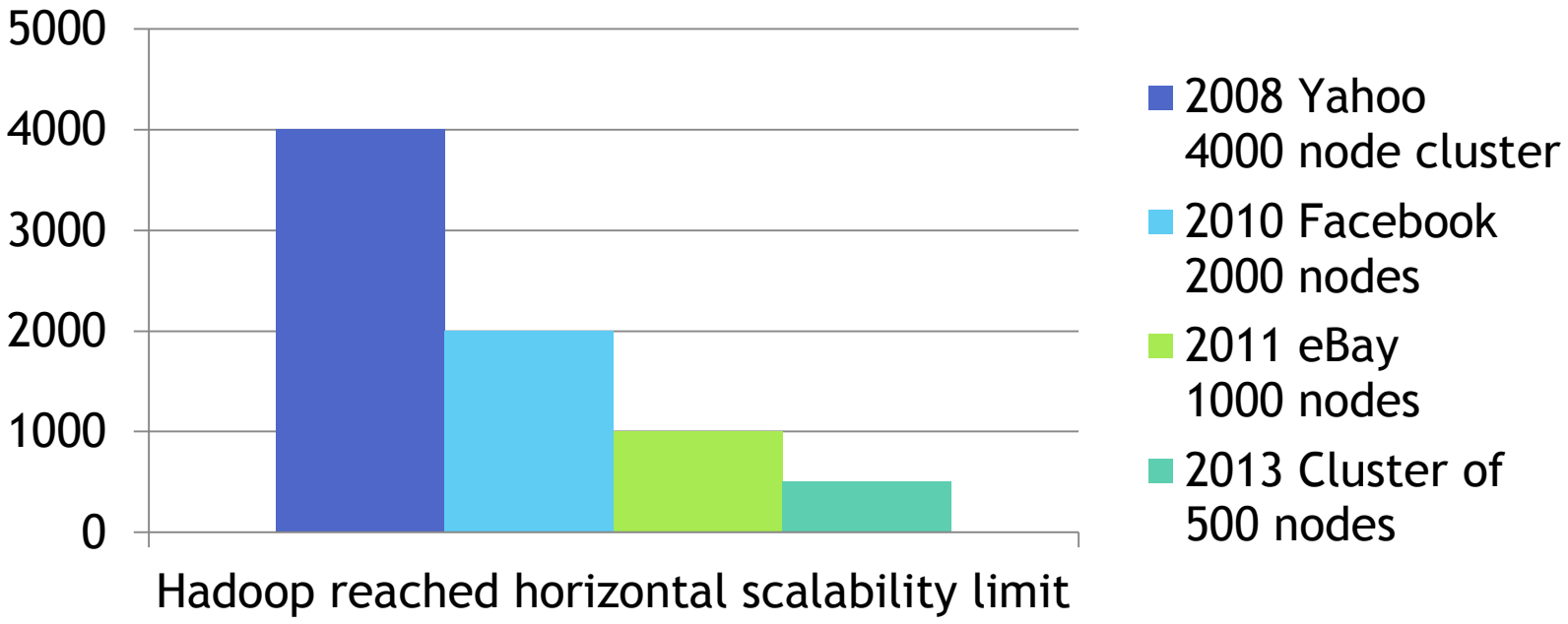
SCALABILITY

Limitations of the Implementation

- **Single master architecture**: a constraining resource
- Limit to the number of **namespace objects**
  - A NameNode object (file or block) requires < 200 bytes in RAM
  - Block to file ratio is shrinking: 2 –> 1.5 -> 1.2
  - 64 GB of RAM yields: 100 million files; 200 million blocks
  - Referencing 20 PB of data with and block-to-file ratio 1.5 and replication 3
- Limits for linear **performance** growth
  - linear increase in # of workers puts a higher workload on the single NameNode
  - Single NameNode can be saturated by a handful of clients
- Hadoop MapReduce framework reached its scalability limit at 40,000 clients
  - Corresponds to a 4,000-node cluster with 10 MapReduce slots
- "HDFS Scalability: The limits to growth" USENIX ;login: 2010

From Horizontal to Vertical Scaling

- Horizontal scaling is limited by single-master-architecture
- Vertical scaling leads to cluster size shrinking
  - While Storage capacities, Compute power, and Cost remain constant



Namespace Partitioning

- Static: Federation
  - Directory sub-trees are statically distributed between separate instances of FSs
  - Relocating sub-trees without copying is challenging
  - Scale x10: billions of files

- Dynamic
  - Files, directory sub-trees can move automatically between nodes based on their utilization or load balancing requirements
  - Files can be relocated without copying data blocks
  - Scale x100: 100s of billion of files

- Orthogonal independent approaches.
  - Federation of distributed namespaces is possible

Distributed Metadata: Known Solutions

- Ceph
  - Metadata stored on OSD
  - MDS cache metadata
  - Dynamic Metadata Partitioning
- GFS Colossus: from Google S. Quinlan and J.Dean
  - 100 million files per metadata server
  - Hundreds of servers
- Lustre
  - Plans to release clustered namespace
  - Code ready
- VoldFS, CassFS, MySQL – prototypes

HBase Overview

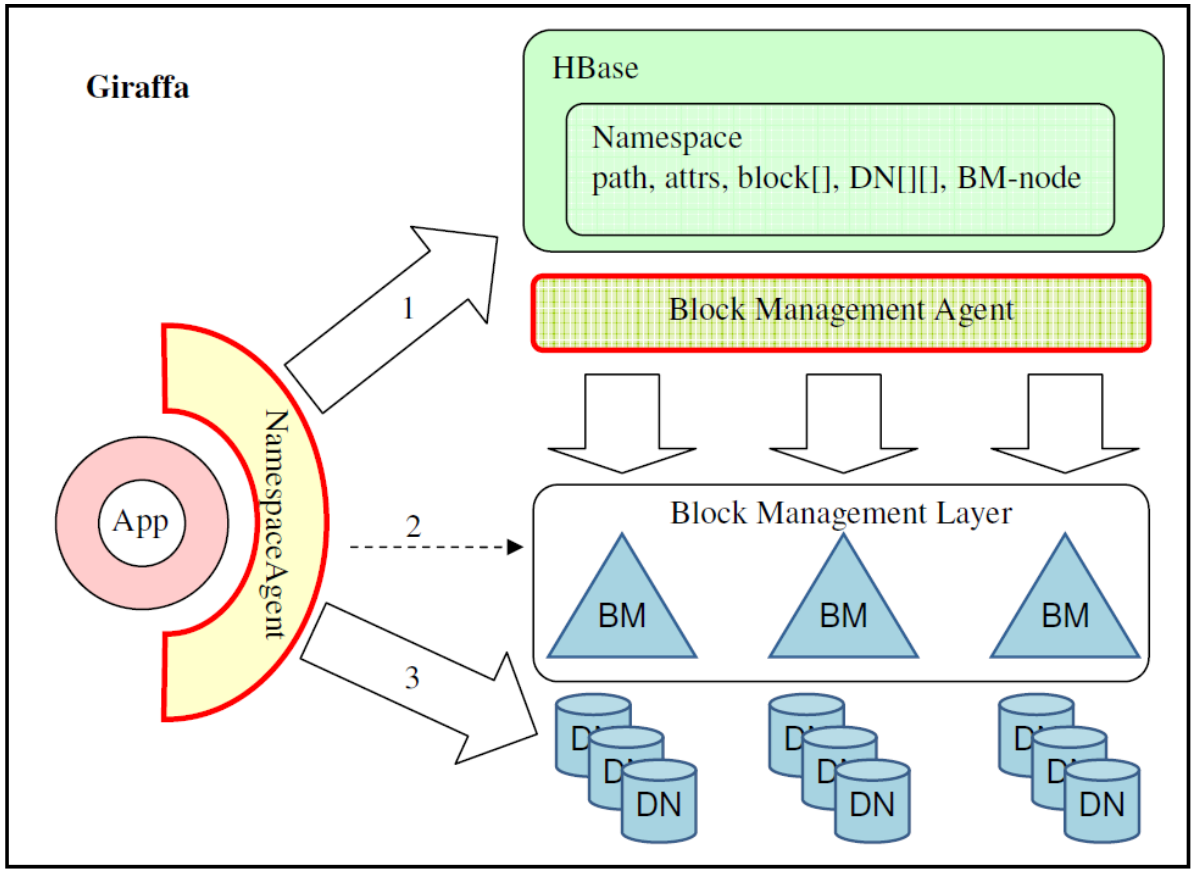
- Distributed table storage system
- Tables: big, sparse, loosely structured
  - Collection of rows
  - Has arbitrary number of columns
- Table are Horizontally partitioned into regions. *Dynamic partitioning*
- Columns can be grouped into Column families
  - Vertical partition of the table
- Distributed cache: Regions loaded into RAM on cluster nodes
- Timestamp: user-defined cell versioning, 3-rd dimension.
  - Cell id: <row_key, column_key, timestamp>

Giraffa File System

- Goal: build from existing building blocks
  - minimize changes to existing components
- HDFS + HBase = Giraffa
- Store metadata in HBase table
  - Dynamic table partitioning into regions
  - Cashed in RAM for fast access
- Store data in blocks on HDFS DataNodes
  - Efficient data streaming
- Use NameNodes as block managers
  - Flat namespace of block IDs – easy to partition
  - Handle communication with DataNodes
  - Perform block replication

Giraffa Architecture

- Dynamic namespace table partitioning with HBase
- Data streaming to HDFS



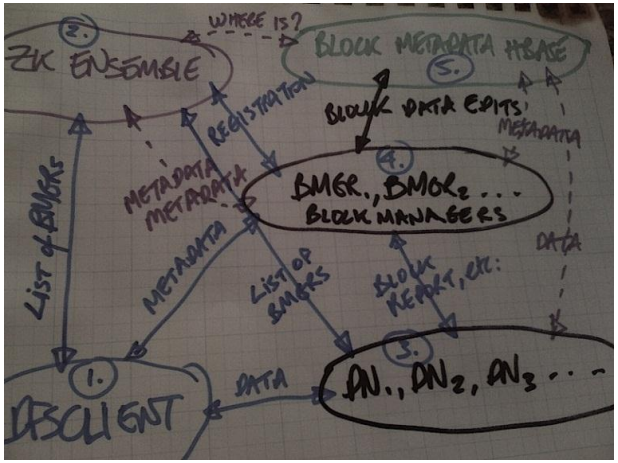
Giraffa Facts

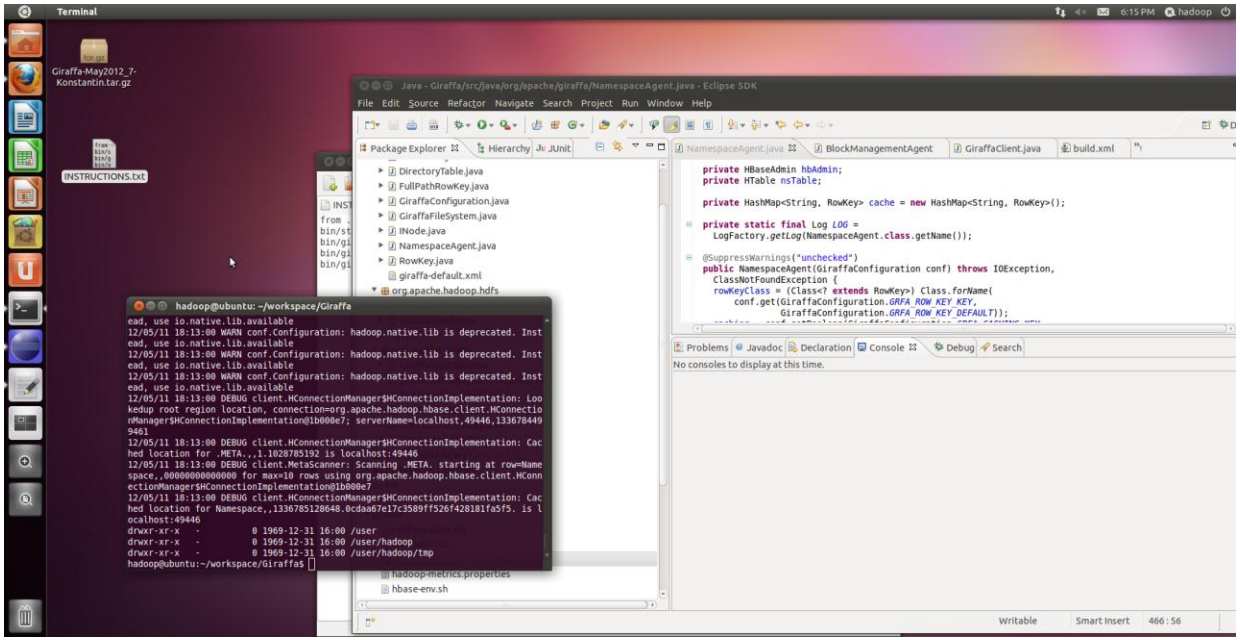
- HBase * HDFS = high scalability
  - More data & more files
- High Availability
  - no SPOF, load balancing
- Single cluster
  - no management overhead for operating more node

	HDFS	Federated HDFS	Giraffa
Space	25 PB	120 PB	1 EB = 1000 PB
Files + blocks	200 million	1 billion	100 billion
Concurrent Clients	40,000	100,000	1 million

Current State

- Design stage
- One node cluster running





The End