HDFS for Geographically Distributed File System

Konstantin V. Shvachko

November 20, 2014
Barcelona, Spain

Dimensions of Scalability

For distributed storage system

- Ability to support ever increasing requirements for
  - Space: more data
  - Objects: more files
  - Load: more clients
- RAM is limiting HDFS scale
- Other dimensions of scalability

Tallest LEGO tower in Budapest

Geographic Scalability

Scaling file system across multiple data centers

- Running HDFS on multiple Data Centers
  - Distributed across the world
  - As a single cluster

Main Steps

Three main stages to the goal

- STAGE I: The role of Coordination Engine
- STAGE II: Replicated namespace
  - Active-active HA
- STAGE III: Geographically-distributed HDFS
  - Requirements and architecture principles
  - Benefits: load-balancing, self-healing, simultaneous data ingest
  - Features:
    - Selective data replication,
    - Heterogeneous storage Zones

The Requirements

Requirements

File System with hardware components distributed over the WAN

- Operated and perceived by users as a ***single system***
  - Unified file system view independently of where the data is physically stored
- ***Consistency***
  - Everybody sees the same data
- Continuous ***Availability***
  - Disaster recovery
- ***Scalability***
  - Support for multiple Data Centers
- Seamless block level replication

Architecture Principles

Strict consistency of metadata with fast data ingest

- Synchronous replication of **metadata** between data centers
  - Using Coordination Engine
  - Allows strict consistency of the namespace
- Asynchronous replication of **data** over the WAN
  - Data blocks replicated in the background
  - Allows fast LAN-speed data creation

Coordination Engine

For Consistent State Replication

Coordination Engine

Determines the order in which operations are applied to the system

- *Coordination Engine* allows to agree on the order of events submitted to the engine by multiple proposers
  - Anybody can Propose
  - Engine chooses a single Agreement every time and guarantees
  - Learners observe the agreements in the same order they were chosen
  - An agreement triggers a corresponding application specific action

Central Coordination

Simple coordination without fault tolerance

- Easy to Coordinate
  - Single NameNode as an example of a Central Coordination Engine
  - Performance and availability bottleneck
  - Single point of failure

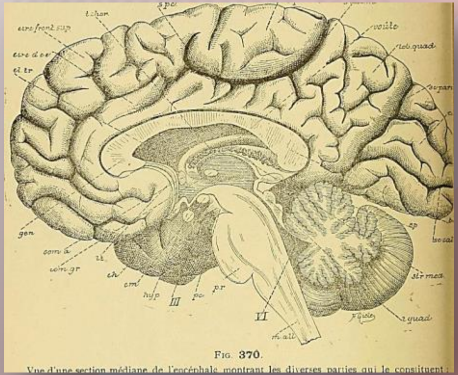
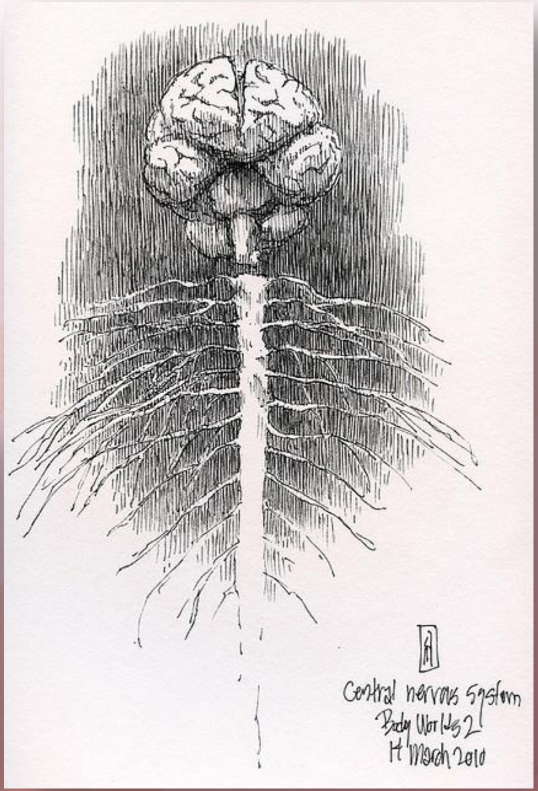


Fig. 370.
Vue d'une section médiane de l'encéphale montrant les diverses parties qui le constituent ;



Distributed Coordination Engine

Fault tolerant coordination using multiple acceptors

- Distributed Coordination Engine operates on participating nodes
  - Roles: Proposer, Learner, and Acceptor
  - Each node can combine multiple roles
- Distributed coordination
  - Proposing nodes submit events as *proposals* to a quorum of acceptors
  - Acceptors agree on the order of each event in the *global sequence* of events
  - Learners learn *agreements* in the same deterministic order

Consensus Algorithms

Consensus is the process of agreeing on one result among a group of participants

- Coordination Engine guarantees the same state of the learners at a given GSN
  - Each agreement is assigned a unique *Global Sequence Number (GSN)*
  - GSNs form a monotonically increasing number series – the order of agreements
  - Learners start from the same initial state apply the same deterministic agreements in the same deterministic order
  - GSN represents “logical” time in coordinated systems
- PAXOS is a consensus algorithm proven to tolerate a variety of failures
  - Quorum-based Consensus
  - Deterministic State Machine
  - *Leslie Lamport: Part-Time Parliament (1990)*

```
Client   Proposer      Acceptor     Learner
   |         |          |  |  |       |  |
   X-------->|          |  |  |       |  |  Request
   |         X--------->|->|->|       |  |  Prepare(1)
   |         |<---------X--X--X       |  |  Promise(1,{Va,Vb,Vc})
   |         X--------->|->|->|       |  |  Accept!(1,Vn)
   |         |<---------X--X--X------>|->|  Accepted(1,Vn)
   |<---------------------------------X--X  Response
   |         |          |  |  |       |  |
```

HDFS

State of the Art

HDFS Architecture

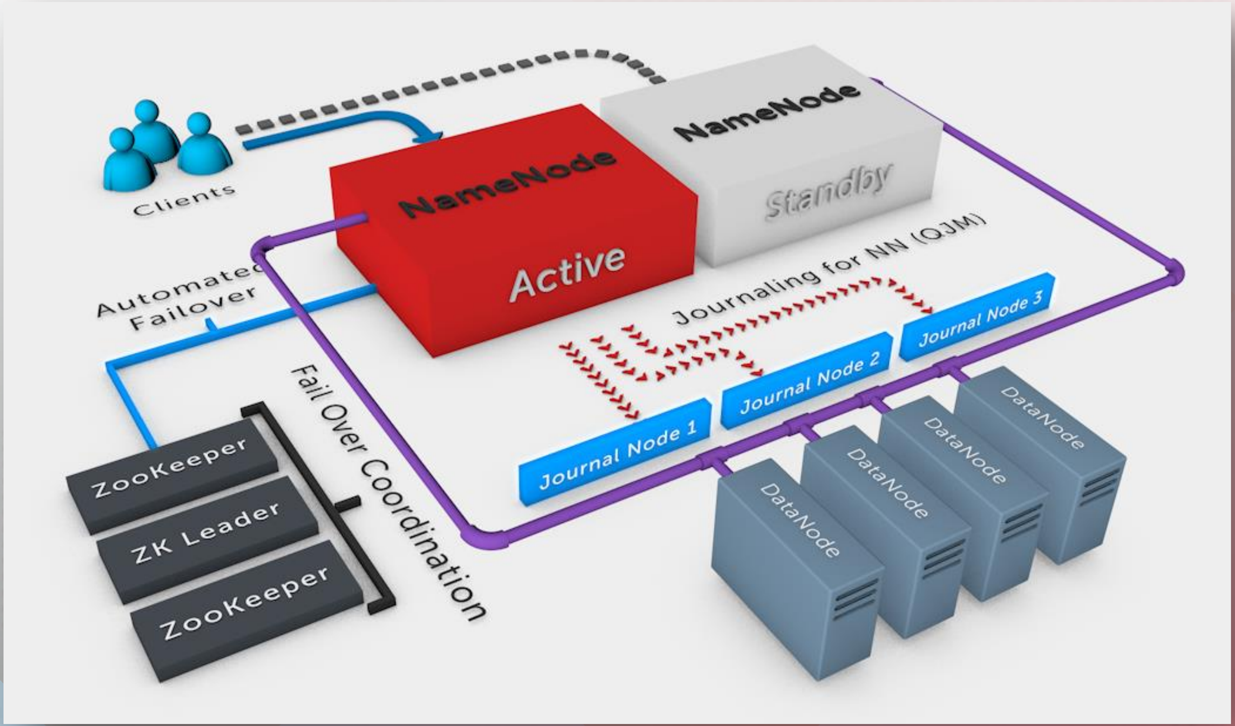
Reliable distributed file system for storing very large data sets

- **HDFS metadata is decoupled from data**
  - Namespace is a hierarchy of files and directories represented by Inodes
  - INodes record attributes: permissions, quotas, timestamps, replication
- NameNode keeps its entire state in RAM
  - Memory state: the namespace tree and the mapping of blocks to DataNodes
  - Persistent state: recent checkpoint of the namespace and journal log
- File data is divided into blocks (default 128MB)
  - Each block is independently replicated on multiple DataNodes (default 3)
  - Block replicas stored on DataNodes as local files on local drives

HDFS Cluster

Active-Standby Architecture

- Single active NameNode fails over to Standby using ZooKeeper and QJM
- Thousands of DataNodes store data blocks
- Tens of thousands of HDFS clients connected to active NameNode



Standard HDFS operations

Central Coordination via single active NameNode

- Active NameNode workflow
  1. Receive request from a client,
  2. Apply the update to its memory state,
  3. Record the update as a journal transaction in persistent storage,
  4. Return result to the client

- HDFS Client (read or write to a file)
  – Send request to the NameNode, receive replica locations
  – Read or write data from or to DataNodes

- DataNode
  – Data transfer to / from clients and between DataNodes
  – Report replica states to NameNode(s): *new, deleted, corrupt*
  – Report its own state to NameNode(s): *heartbeats, block reports*

Consensus Node

Coordinated Replication of HDFS Namespace

Replicated Namespace

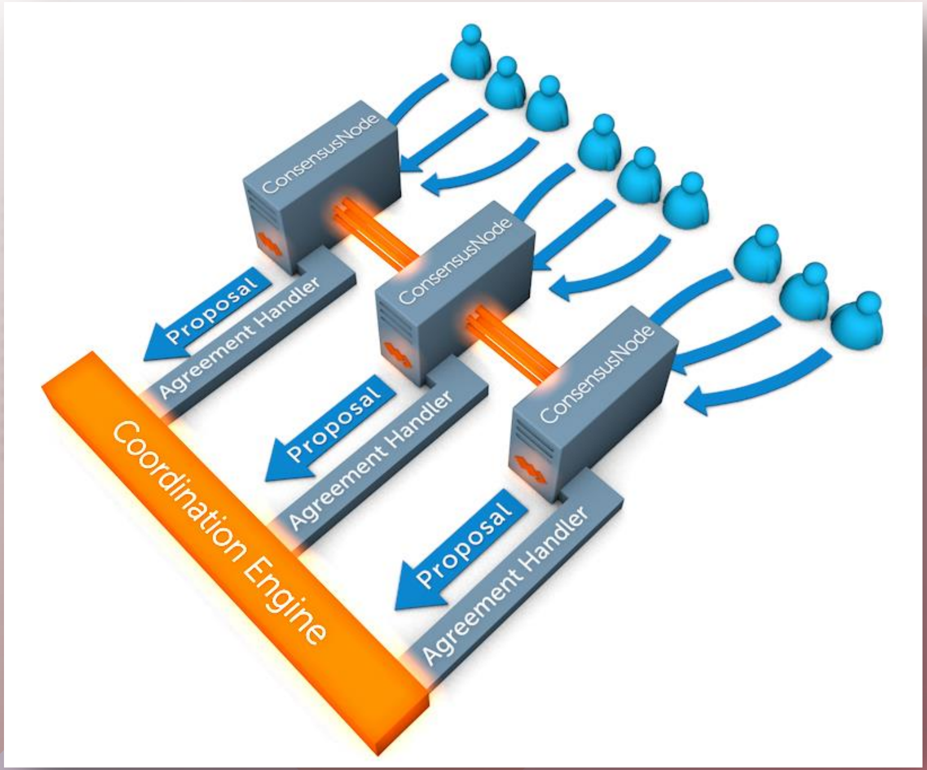
Coordination Engine provides equivalence of multiple namespace replicas

- Coordinated NameNode is called a ***ConsensusNode*** or ***CNode***
- ConsensusNodes play equal *active* role on the cluster
  - Provide *write and read access* to the namespace
- The namespace replicas are consistent with each other
  - Each CNode maintains a replica of the same namespace
  - Any CNode can initiates an update, which is propagated to all CNodes
- Coordination Engine establishes the global order of namespace updates
  - All CNodes apply the same *deterministic* updates in the same *deterministic* order
  - Systems, which start from the same state and apply the same updates are equivalent

Coordinated HDFS Cluster

All-active Consensus Nodes replicate namespace via Coordination Engine

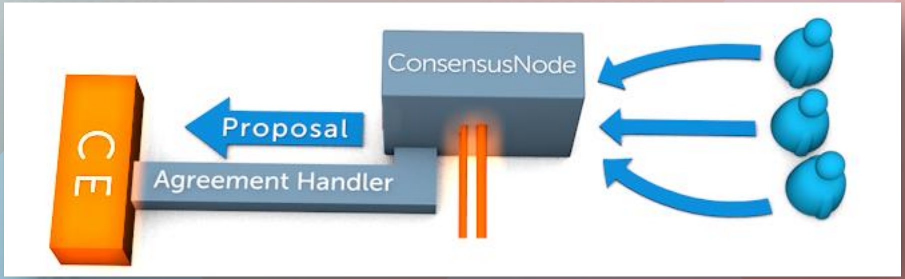
- Independent CNodes – single namespace
- Load balancing client requests
- Coordinated updates
- Proposal, Agreement



Coordinated HDFS operations

Updates to the namespace when a file or a directory is created are coordinated

- ConsensusNode workflow
  1. Receive request from a client
  2. *Submit the update to the Coordination Engine as a proposal*
     Wait for agreement
  3. Apply the agreed update to its memory state,
  4. Record the update as a journal transaction in persistent storage (optional)
  5. Return result to the client
- HDFS Client and DataNode operations remain unchanged



Strict Consistency Model

One-Copy-Equivalence as known in replicated databases

- *Coordination Engine sequences namespace modification proposals into the* ***global sequence of agreements***
  - Applied to namespace replicas in the order of their Global Sequence Number
- *CNodes have identical namespace states when they reach the same GSN*
- ConsensusNodes may have different states at a given moment of "clock" time
  - As the rate of consuming agreements may vary
- ***One-copy-equivalence***
  - each replica presented to the client as if it has only one copy

21

Consensus Node Proxy

Reads can be directed to any ConsensusNode as they do not modify namespace

- CNodeProxyProvider – a pluggable substitute of FailoverPorxyProvider
  - Defined via Configuration
- Main features
  - Randomly chooses CNode when client is instantiated
  - Sticky until a timeout occurs
  - Fails over to another CNode
  - Smart enough to avoid SafeMode
- Further improvements
  - Take into account network proximity

Stale Read Problem

A few read requests must be coordinated

1. Same client fails over to a CNode, which has an older namespace state (GSN)
   - CNode1 at GSN 900: mkdir(p) –> ls(p) –> failover to CNode2
   - CNode2 at GSN 890: ls(p) –> directory not found
2. One client modifies namespace, which needs to be seen by other clients MapReduce use case:
   - JobClient to CNode1: create job.xml
   - MapTask to CNode2: read job.xml –> FileNotFoundException

1) Client connects to CNode only if its GSN >= last seen
   - May need to wait until CNode catches up

2) Must coordinate file read
   - Coordinated read: submit proposal, wait for agreement
   - Special files only: configuration-defined regexp
   - CNode coordinates read only once per file, then marks it as coordinated

Block Management

Elected Block Replicator is the only CNode that schedules block replication and deletion

- Block Manager keeps information about file blocks and DataNodes
  - Blocks Map, DataNode Manager, Replication Queues
  - New block generation: `<blockId, generationStamp, locations>`
  - Replication and deletion of replicas that are under- or over-replicated
- Consistent block management problem
  - Collision of BlockIDs if the same ID generated by different CNodes
  - Over-replicated block: if CNodes decide to replicate same block simultaneously
  - Missing block data: if CNodes decide to delete different replicas of the block
- New block generation – all CNodes
  - Assign nextID and nextGeneratoinStamp while processing the agreement
- Replication and deletion of replicas – a designated CNode (Block Replicator)
  - Block Replicator is elected using Coordination Engine and reelected if fails
  - Only Block Replicator sends DataNode commands to transfer or delete replicas

GeoNode

Geographically Distributed HDFS

Scaling HDFS Across Data Centers

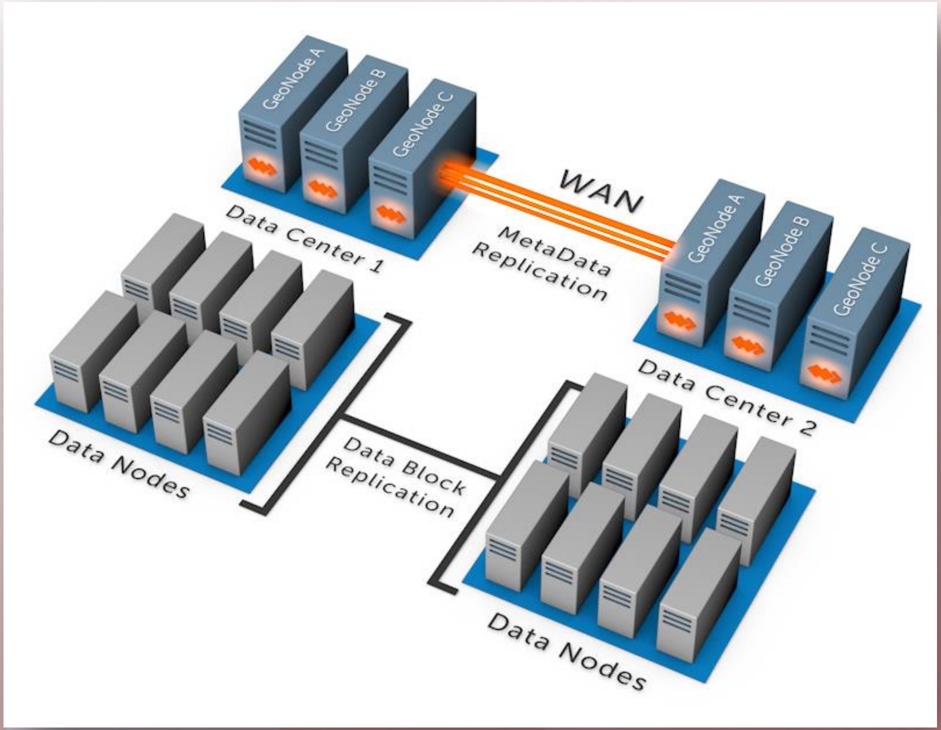
Continuous Availability and Disaster Recovery over the WAN

- The system should appear, act, and be operated as a **single cluster**
  - Instant and automatic replication of data and metadata
- Parts of the cluster on different data centers should have **equal roles**
  - Data could be ingested or accessed through any of the centers
- Data creation and access should typically be at the **LAN speed**
  - Running time of a job executed on one data center as if there are no other centers
- **Failure scenarios:** the system should provide service and remain consistent
  - Any GeoNode can fail
  - GeoNodes can fail simultaneously on two or more data centers
  - An entire data center can fail. E.g. due to WAN partitioning
  - Tolerate DataNode failures traditional for HDFS
    - Simultaneous failure of two DataNodes
    - Failure of an entire rack

WAN HDFS Cluster

Multiple active GeoNodes sharing namespace via Coordination Engine over WAN

- Wide Area Network replication
- Metadata – synchronous
- Data – asynchronous



Geo-Architecture Principles

- *Foreign* vs *Native* notations for nodes, block replicas, clients
  - Foreign means on a different data center. Native – on the same data center
- Metadata between GeoNodes is coordinated instantaneously via agreements
- Data between data centers replicated asynchronously in the background
  - Replicas of a block can be stored on DataNodes of one or multiple data centers
  - GeoNode may learn about the file and its blocks before native replicas created
- DataNodes do not communicate to foreign GeoNodes
  - Heartbeats, block reports, etc. sent to native GeoNodes only
  - GeoNodes cannot send direct commands to foreign DataNodes
- DataNodes copy blocks over the WAN to provide foreign block replication
  - Copy single replica over the WAN, allow native replication of that replica
- Clients optimized to access or create data on native DataNodes
  - But can read data from another data center if needed

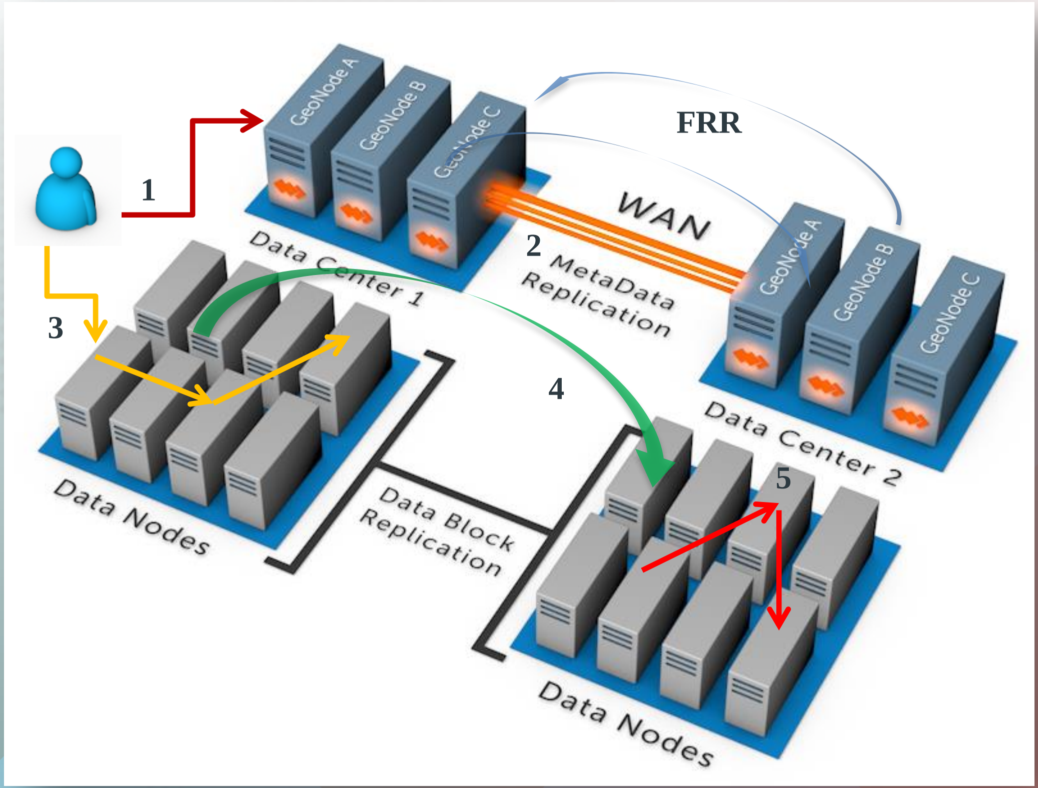
File Create

Create file entry, add blocks, write data, close file

- Client: choose a native GeoNode, send request
- GeoNode: propose, wait for agreement
  - All GeoNodes execute the agreement
  - addBlock: choose native locations only
- Client: create pipeline, write data
- DataNodes report replicas to native GeoNodes
- Client can continue adding next block or closing file
  - Clients do not wait until WAN transfer is completed
- Foreign block replication handled asynchronously

Foreign Block Replication

Block is created on client's data center and replicated to other asynchronously



Foreign Replica Report

- Foreign Replica Report proposal is submitted in two cases
  1. Count of native block replicas reaches full replication for the data center or
  2. Count of native block replicas is reduced to 0
- If block replica is lost it is replicated natively first
  – New locations reported to other data centers after *full replication* is reached
- If no native replicas left, one should be transferred from another data center
- Replication factor of a given block can vary on different data centers

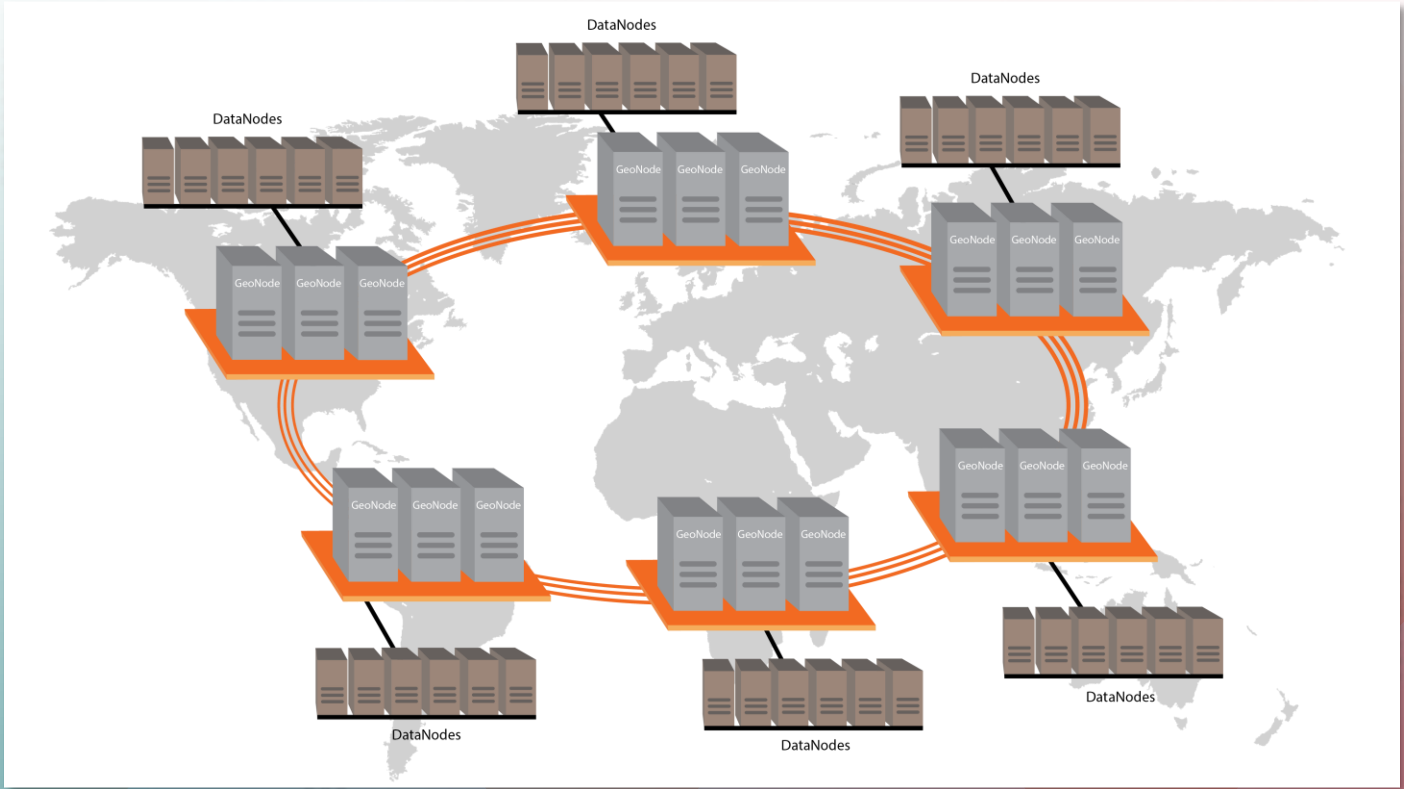
Foreign Block Management

- Designated BlockReplicator per data center
  - Elected among GeoNodes of its data center
  - Handles deletion and replication of blocks among native DataNodes
  - Schedules foreign replica transfers

- BlockMonitor in addition to native replicas analyses foreign block replication
  - Periodically scans blocks
  - If a block with no foreign replicas found – schedules WAN replication

Multi – Data Center Installation

Should I need so many replicas?



Selective Replication

Features

Asymmetric Block Replication

Allow different block replication on different data centers

- Data center specific replication factor
  - Files created with the default for the data center replication
  - setReplication() is applied to single data center
  - Special case of replication = 0: data is not replicated on that data center
- Generalization: data center specific attributes
  - Default replication: per directory, rather than cluster-wide
  - Permissions – *Selective visibility* of files and directories

Selective Data Replication

Two main use cases for restricting data replication

1. **“Saudi Arabia” case** – *Data must never leave specific data center*
   - This is needed to protect data from being replicated outside of a specific geo-location, a country, or a facility. E.g. customer data from a branch in Saudi Arabia of a global bank must never leave the country due to local regulations.
2. **“/tmp” case** – *Data created in a directory by a native client should remain native*
   - Transient data of a job running on a DC do not need to be replicated elsewhere as it is deleted upon job completion and nobody else needs it.

- **Single namespace:** *metadata is always replicated*
- Some files can have **replication of 0** on some data centers
  - Data is not replicated on that data center

SDR Policy

A set of rules defines, which data should remain NATIVE or replicated on ALL data centers

- SDR policy is defined on directories
  - The policy specifies to which data centers file blocks should or should not be replicated to
  - xml document: defines mapping from path to DCs
  - Can be enabled on startup
  - Can be dynamically changed via a special REST call
  - Persisted through restarts

Heterogeneous Storage Zones

Virtual Data Centers representing different types of block storage

- Storage Types: Hard Drive, SSD, RAM
- Virtual data center is a zone of similarly configured Data Nodes
- Example:
  - Z1 *archival* zone: DataNodes with dense hard drive storage
  - Z2 *active data* zone: DataNodes with high-performance SSDs
  - Z3 *real-time access* zone: lot's of RAM and cores, short lived hot data
- SDR policy defines three directories:
  - `/archive` – replicated only on Z1
  - `/active-data` – replicated on Z2 and Z1
  - `/real-time` – replicated everywhere

Thank You

Questions?

HDFS for Geographically Distributed File System

Konstantin V. Shvachko