A Distributed Storage Schema for Cloud Computing based Raster GIS Systems

Presented by

Cao Kang, Ph.D.

Geography Department, Clark University

Cloud Computing and Distributed Database Management System

- Why distributed database in Cloud?
  - Better scalability, availability, reliability

  Match each other: GFS/Bigtable, Hadoop/HBase
- Categories:
  - NoSQL Distributed Database Management System (NDDBMS)
  - Relational Database Management System (RDBMS) sharding

What is NoSQL

- NoSQL is the term used to designate database management systems that differ from classic RDBMS in some way. *
  - Data stores may not require fixed table schemas
  - Usually avoid join operations
  - Typically scale horizontally

* Wikipedia: http://en.wikipedia.org/wiki/NoSQL_%28concept%29

Why NoSQL Distributed DBMS?

- Scalabilities: Scale Out vs. Scale Up
  - Keep adding more CPUs and memories into one expensive giant server or buy two smaller servers
  - Capacity and cost do not go up in a linear way
  - RDBMS (non-sharding) can scale up, but not scale out
- Cost: can use commodity computers
- Easy to work with: More friendly than RDBMS sharding

Current NDDBMS on the Market

- Proprietary NDDBMS:
  - Google Bigtable
  - Amazon Dynamo
- Opensource NDDBMS:
  - HBase
  - Cassandra

Why HBase?

- HBase offers good scalability.
- HBase is built to use commodity hardware.
- HBase can host huge volumes of data.
- HBase offers high availabilities and reliabilities.
- HBase offers strong consistency through its data operations.

vs. RDBMS

vs. Cassandra

HBase Data Model – Conceptual View

- Conceptual View:
  - Column based (vs. RDBMS row based)
  - Each column family may contain multiple qualifiers
  - Each cell is identified by: *{row, column, version}*

Row Key	**Time Stamp**	**Column** *"water height inside dam"*	**Column** *"submerged village area"*		**Column** *"hurricane"*
"Key state park"	t9		"village a"	"20000 m^2"	
	t8		"village b"	"5000 m^2"	
	t6	"50 m"			"Ike"
	t5	"49 m"			
	t3	"42 m"			

column family : qualifier = value

submerged_village_area : village_a = “20000 m^2”

HBase Data Model – Physical Storage

Conceptual view

Row Key	Time Stamp	Column *"water height inside dam"*	Column *"submerged village area"*		Column *"hurricane"*
"Key state park"	t9		"village a"	"20000 m^2"	
	t8		"village b"	"5000 m^2"	
	t6	"50 m"			"Ike"
	t5	"49 m"			
	t3	"42 m"			

Physical storage

Row Key	Time Stamp	Column *"submerged village area"*	
"Key state park"	t9	"village a"	"20000 m^2"
	t8	"village b"	"5000 m^2"

Row Key	Time Stamp	Column *"water height inside dam"*
"Key state park"	t6	"50 m"
	t5	"49 m"
	t3	"42 m"

Row Key	Time Stamp	Column *"hurricane"*
"Key state park"	t6	"Ike"

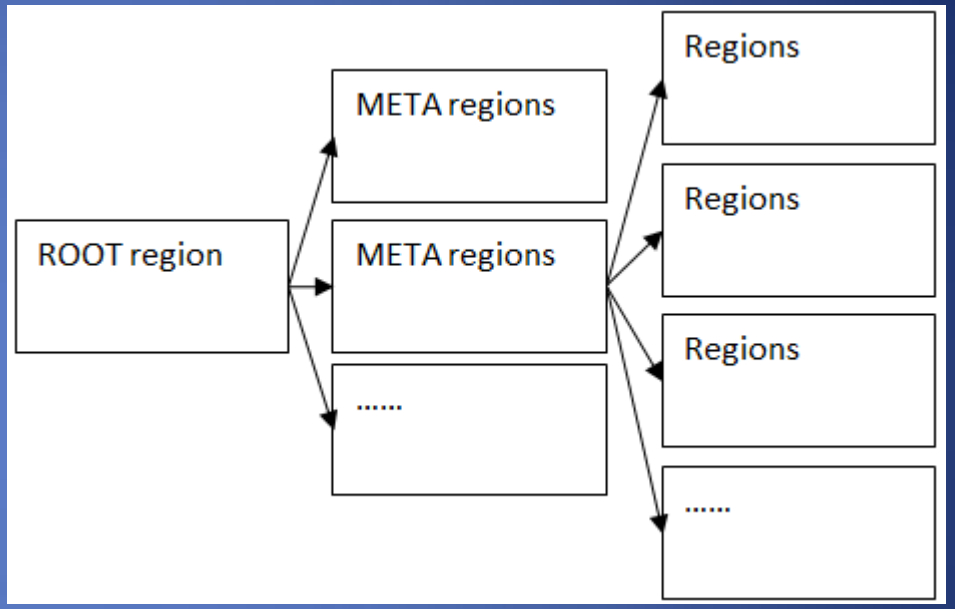
No “null” storage necessary

HBase Data Model

- Each column family is stored in a separate unit.
- Physically, all column family members are stored together on the file system.
- Tables are split into pieces based on row ranges, these pieces of tables are called **Regions**.

HBase Architecture

- ROOT Region Server
- META Region Server
- Data Region Server



In theory, a three-tier HBase can host up to 2^{34} regions with each region hosting 256M of data, which equals up to 4096 petabytes of data.

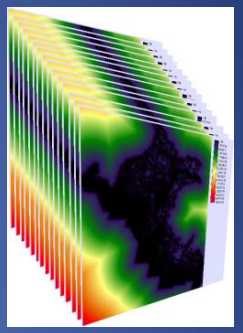
Limits of HBase for GIS Systems

1. The database cell size should be kept small, which in general should not be larger than 20M.
2. The number of column families cannot be infinite.

Storage Schema for Raster GIS Systems

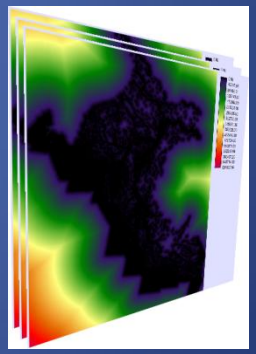
- Two Different Data Types
  - High 3rd Dimensional Data (H3D data)

    Such as time series data (100+ bands)



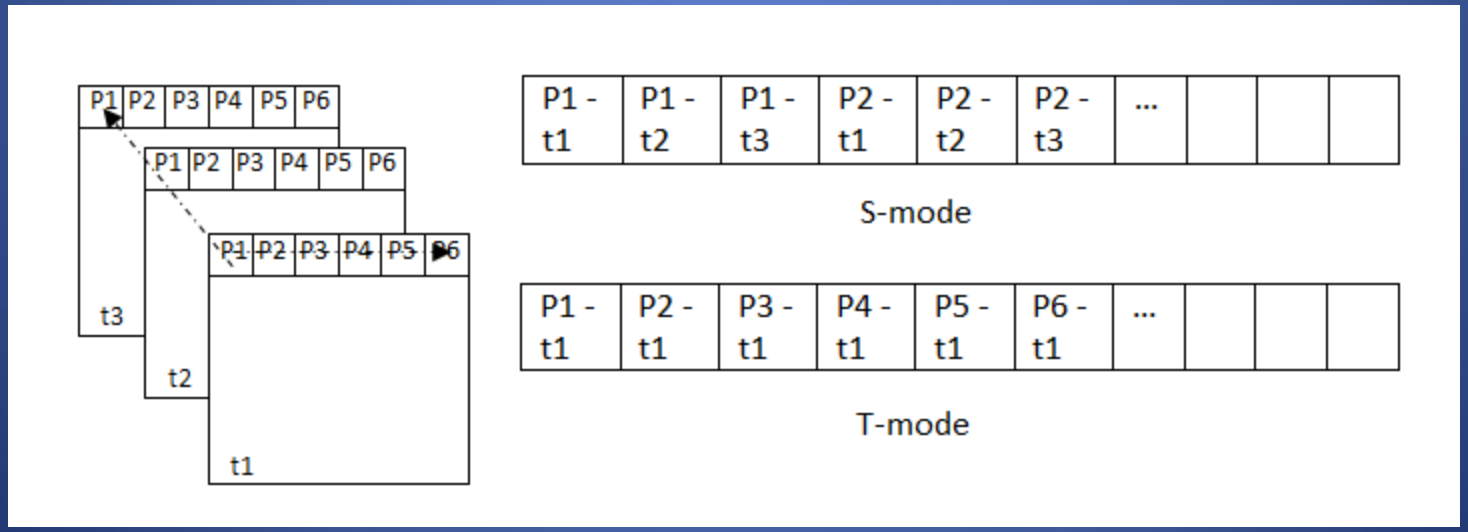
  - Low 3rd Dimensional Data (L3D data)

    Such as ETM+ RGB data (3 bands)



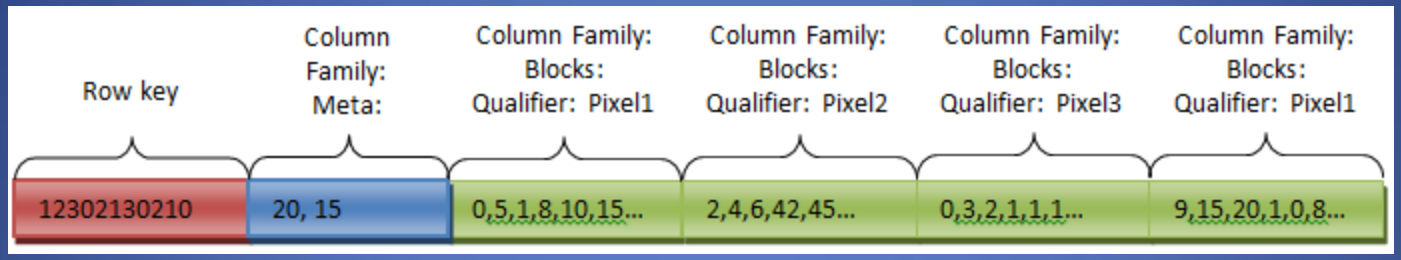
Two Pixel Storage Modes

- Store H3D Data in S-mode (band interleaved by pixel)
- Store L3D Data in T-mode (band sequential)



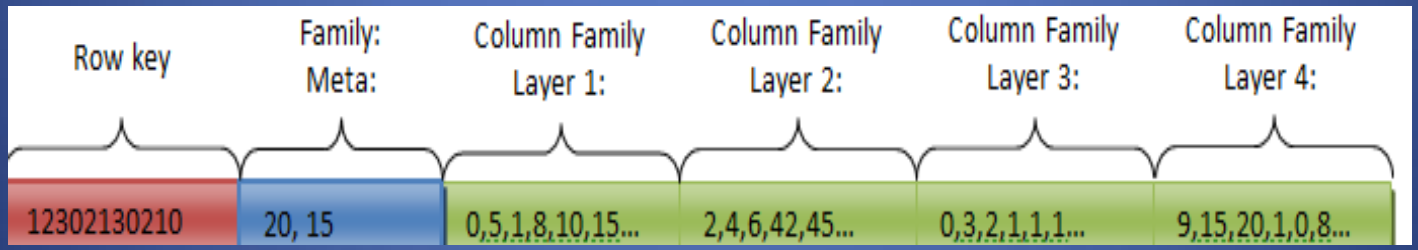
Data Models in HBase

- High 3rd Dimensional Data



Time locality has higher priority than space locality.

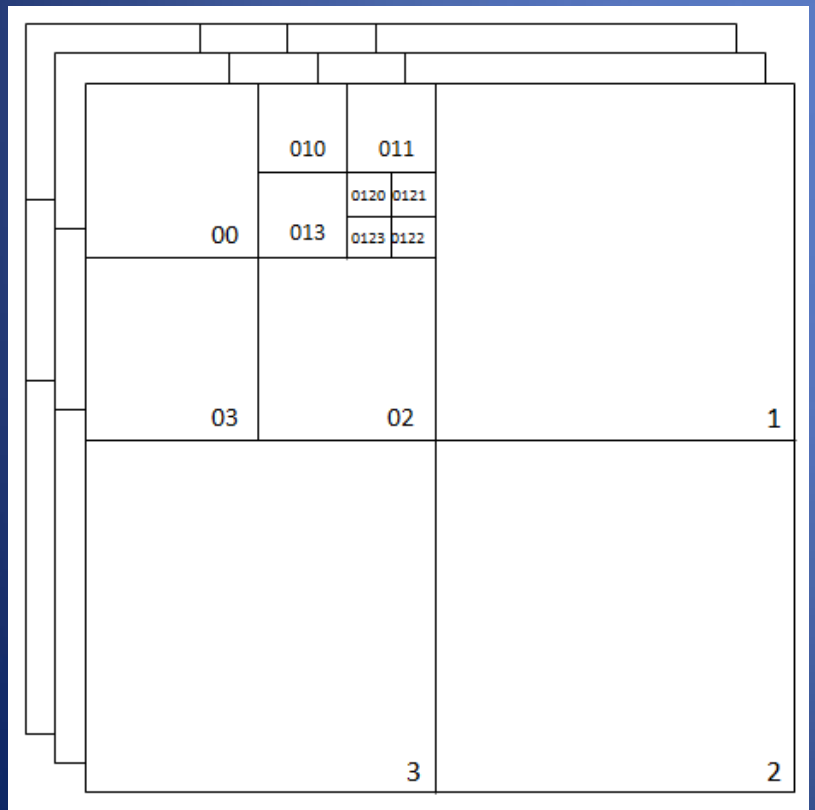
- Low 3rd Dimensional Data

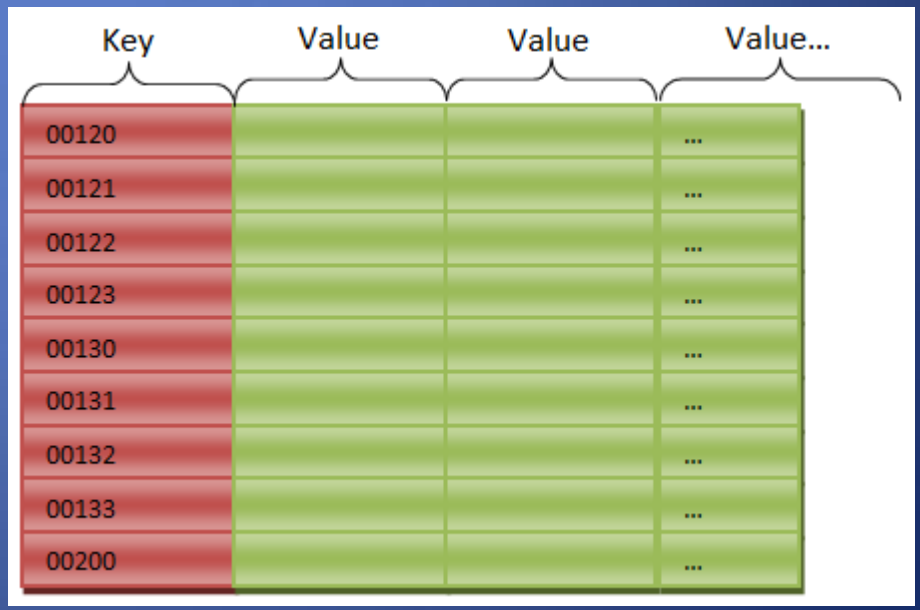


Space locality is better preserved.

Sub-Image Block Generation and Indexing

- A Q-tree alike splitting to sub-divide images
- Keys are sorted lexicographically
- Multi-level indexing based on Q-tree structure can better preserve locality





A Working Example – Google Maps:

- Data in Bigtable

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No

*

- Load balancing

http://mt0.google.com/mt?n=404&v=&x=0&y=0&zoom=16
http://mt1.google.com/mt?n=404&v=&x=1&y=0&zoom=16
http://mt2.google.com/mt?n=404&v=&x=0&y=1&zoom=16
http://mt3.google.com/mt?n=404&v=&x=1&y=1&zoom=16

Before

http://mt1.google.com/vt/lyrs=h@149&hl=en&x=19651&s=&y=24321&z=16&s=Ga
http://mt0.google.com/vt/lyrs=h@149&hl=en&x=19646&s=&y=24323&z=16&s=Galil

Current

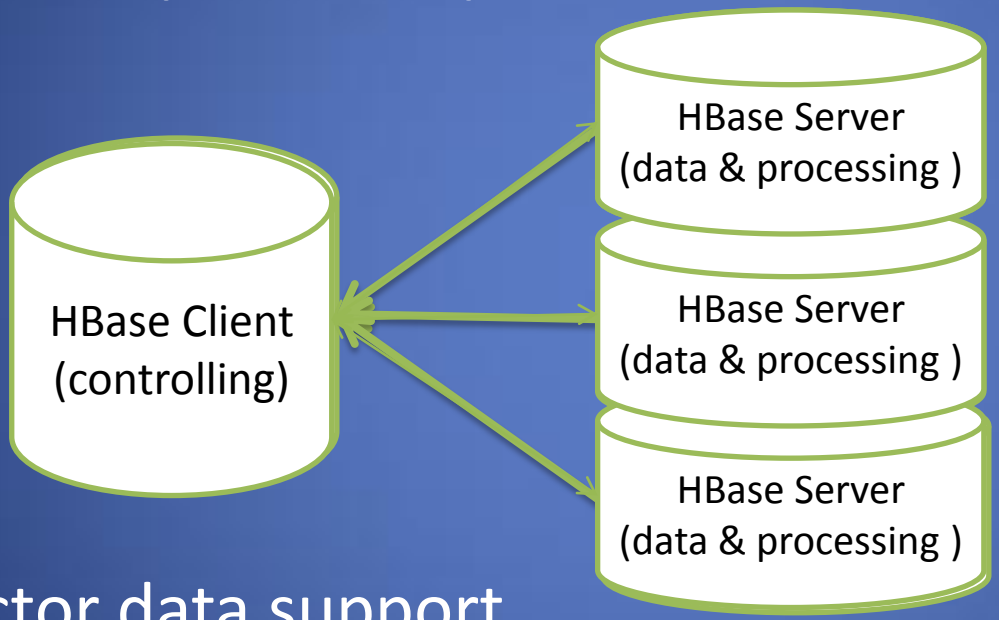
*From Chang, et al, "Bigtable: A Distributed Storage System for Structured Data", OSDI, 2006.

Current Development Status

- Working with HBase 0.2x
  - Block generation is finished
  - Spatial indexing is finished
  - Web front end is close to finish
- Progress with HBase 0.90.2
  - Working on region balancing
  - Need to work on co-processing, which could greatly increase real-time spatial operation speed

Next Step

- Co-processing, which could greatly increase real-time spatial operation speed



- Vector data support

Thank you!
Any questions?