IEEE Cluster, New Orleans, LA, September 1, 2009

Implementing WebGIS on Hadoop: A Case Study of Improving Small File IO Performance on HDFS

Xuhui Liu, Jizhong Han,
Yunqin Zhong, Chengde Han

Xubin He

Institute of Computing Technology
Chinese Academy of Sciences

Tennessee Tech University

Outline

- Introduction
- Problem and Our Approach
- Performance Evaluation
- Conclusion

WebGIS and Hadoop

- WebGIS
  - Distributed
  - Data-intensive
    - Huge spatial data set
    - Large amount of concurrent users
- Hadoop
  - Inspired from GFS
  - HDFS (Hadoop Distributed File System)
  - MapReduce programming model

Deploying WebGIS on Hadoop

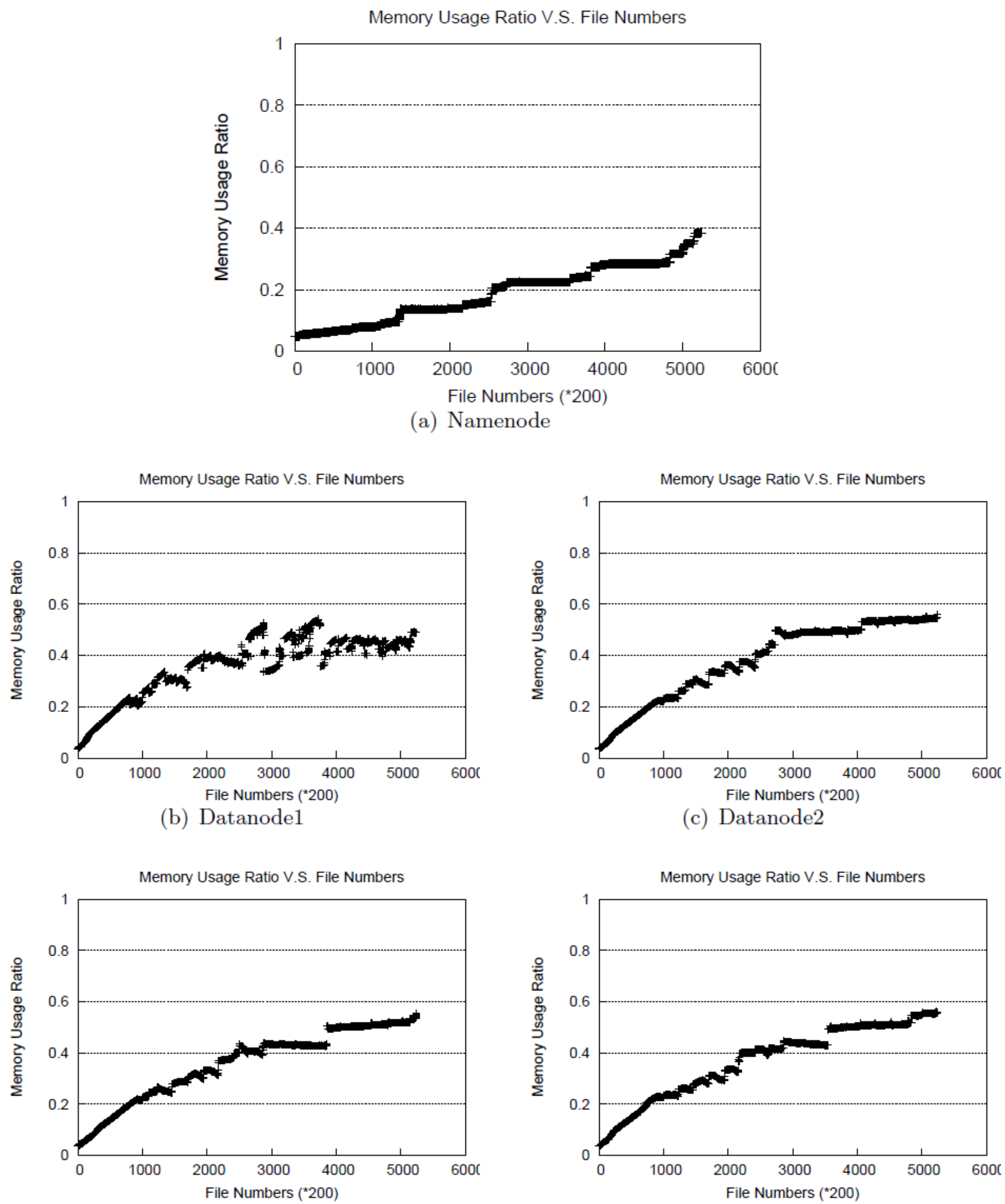
- Pros:
  - Distributing WebGIS
  - Storage and Computing capacity are well combined
    - HDFS for storage
    - MapReduce for computing

- Mismatch
  - WebGIS
    - Small file sizes with tens of KB
    - Large number of files
  - Hadoop
    - Large file sizes with hundreds of MB or even several GB

Problems

- Unacceptable execution time:
  - ~7.7 hours to store 550,000 files into HDFS with Hadoop 0.16.0.
  - ~3.8 hours to store 1million files with latest Hadoop 0.20.0
- High memory usage rate

Memory usage while storing 1M files



(a) Namenode

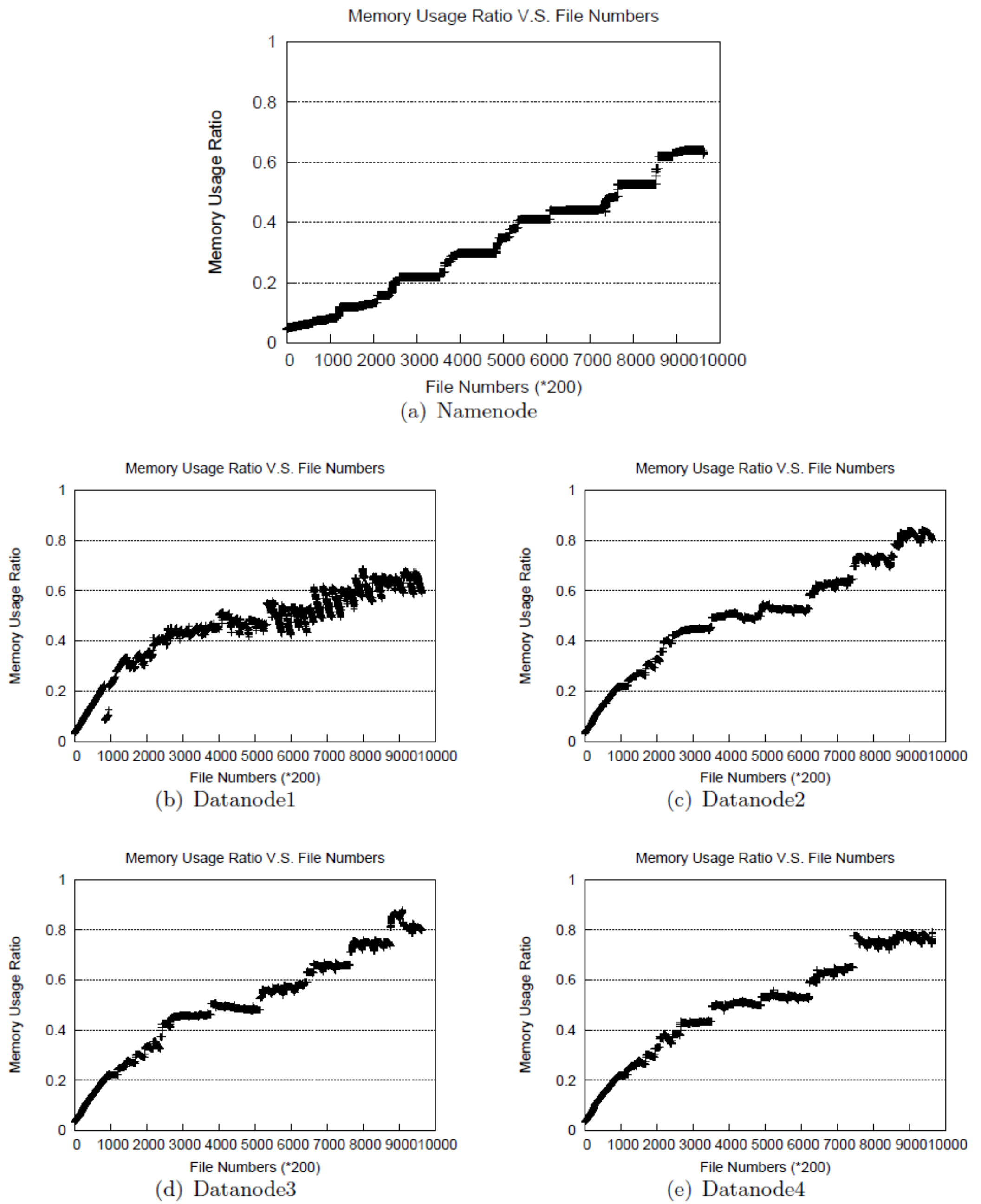
(b) Datanode1

(c) Datanode2

(d) Datanode3

(e) Datanode4

Memory usage while storing 2M files



(a) Namenode

(b) Datanode1

(c) Datanode2

(d) Datanode3

(e) Datanode4

Goal and Approach

- Goal
  - Eliminate gaps between WebGIS application and HDFS system
- Approach
  - Small files are merged to big ones. HDFS divides large file to fixed-size chunks (64MB by default), so, data file size is limited to chunk size in HDFS.
  - Indices are created for every small file

File access patterns in WebGIS

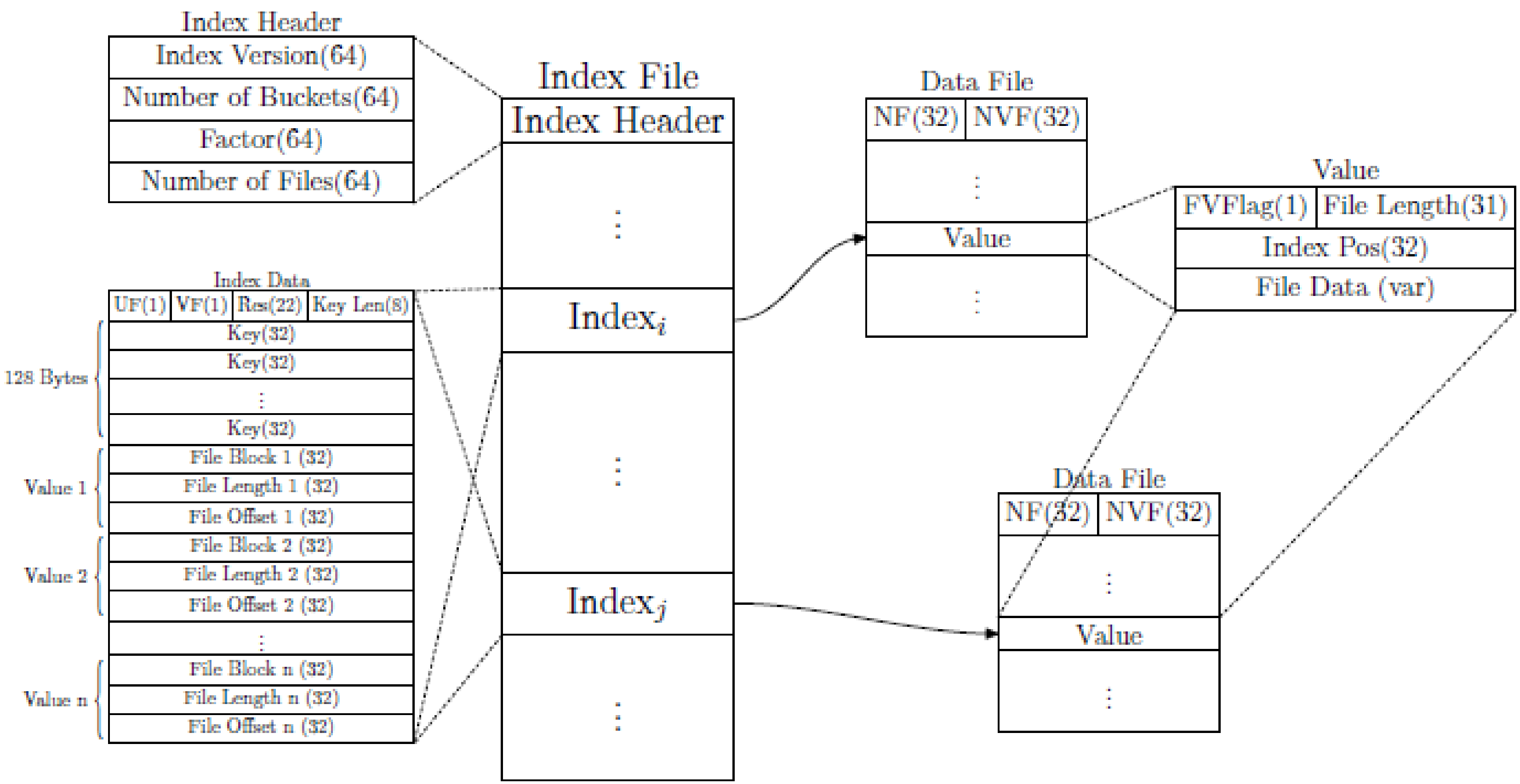
- Some attributes of files' metadata are same: owner, permission

- High spatial locality: when an image file is accessed, its geographic proximal images are likely to be accessed soon.

File organization of WebGIS

- Files are split to fix-size small files (tiles).
- Tiles are located by three variables: (L, (x, y)). L: scale level
- Group $n \times n$ adjacent files to a group.

Our design



Index Header
Index Version(64)
Number of Buckets(64)
Factor(64)
Number of Files(64)
Index Data
UF(1)
VF(1)
Res(22)
Key Len(8)
128 Bytes
Key(32)
Key(32)
Key(32)
Value 1
File Block 1 (32)
File Length 1 (32)
File Offset 1 (32)
Value 2
File Block 2 (32)
File Length 2 (32)
File Offset 2 (32)
Value n
File Block n (32)
File Length n (32)
File Offset n (32)
Index File
Index Header
Index$_i$
Index$_j$
Data File
NF(32)
NVF(32)
Value
Value
FVFlag(1)
File Length(31)
Index Pos(32)
File Data (var)
Data File
NF(32)
NVF(32)
Value

Index File

- Index Header

Index Version(64)	Number of Buckets(64)	Factor(64)	Number of Files(64)

- Index Block

UFlag(1)	VFlag(1)	Reserve(22)	Key Length(8)
Key(64)			
⋮			
Key(64)			
File Block 1 (32)	File Length 1 (32)	File Offset 1 (32)	
⋮			
File Block n (32)	File Length n (32)	File Offset n (32)	

Data File

- Data header

Number of Files(32)	Number of Valid Files(32)

- Data block

FVFlag(1)	File Length(31)	Index Position(64)	File Data(var)

File Operations

- **Write Files**
  1. Collect and group files
  2. Calculate index for file to be stored,
  3. Append data of small file to data file
  4. Update index

- **Read a file**
  1. Get read request.
  2. Calculate and look up index.
  3. Retrieve data block in data file according to index.

File Operations (Contd.)

- Delete a file
  - Lazy delete scheme is used. Deleted file will not be removed from system immediately instead of being marked as being deleted.
  - A space reclamation operation will be performed to a data file if *Number of Valid Files* field is less than half value of *Number of Files* field in data file.
- Update a file
  - A combination of file delete and write.
  - Update-out-of-place.

Performance Evaluation -Experiment Setup

- Hardware:
  - Five Dell Power Edge SC430 nodes: Intel Pentium 4 CPU of 2.8GHz,1GB or 2GB memory, 80GB or 160GB SATA disk.
- Software:
  - OS: Red Hat AS4.4 (kernel 2.6.20).
  - Hadoop: 0.16.1
  - Java: 1.6.0.
- One master/four data nodes. Number of replications is set to 2 in tests.

Performance Evaluation -Data Set

- # of files: 558,726
- Dataset: 3.6GB
- Avg size: ~7KB

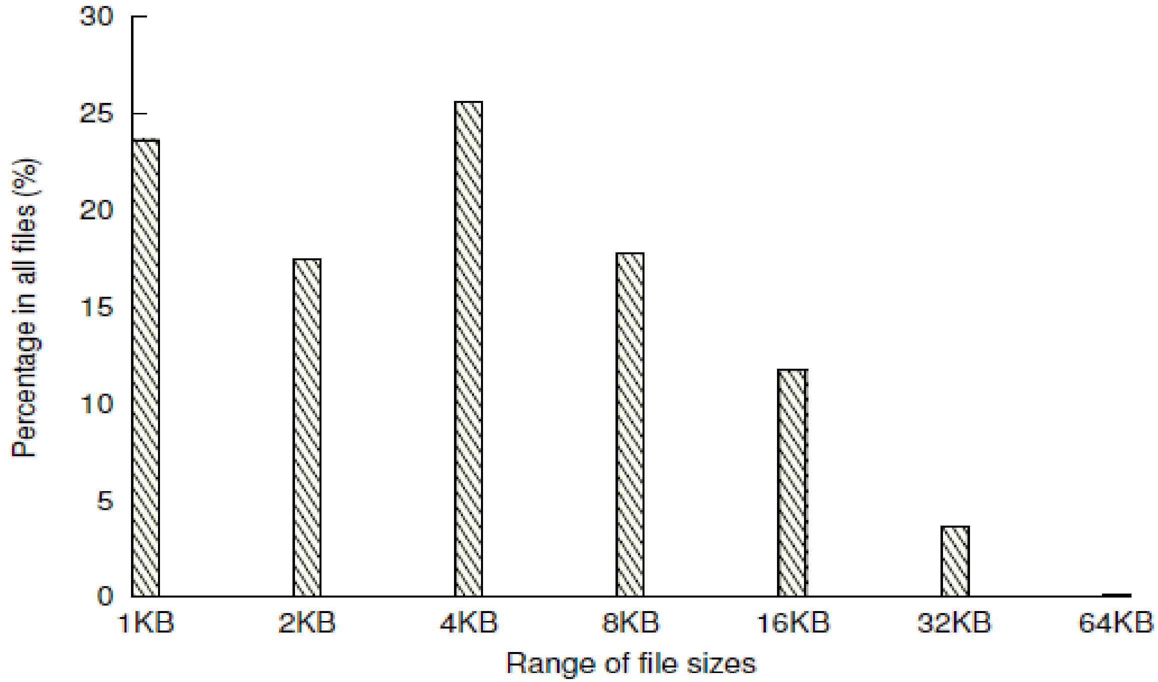


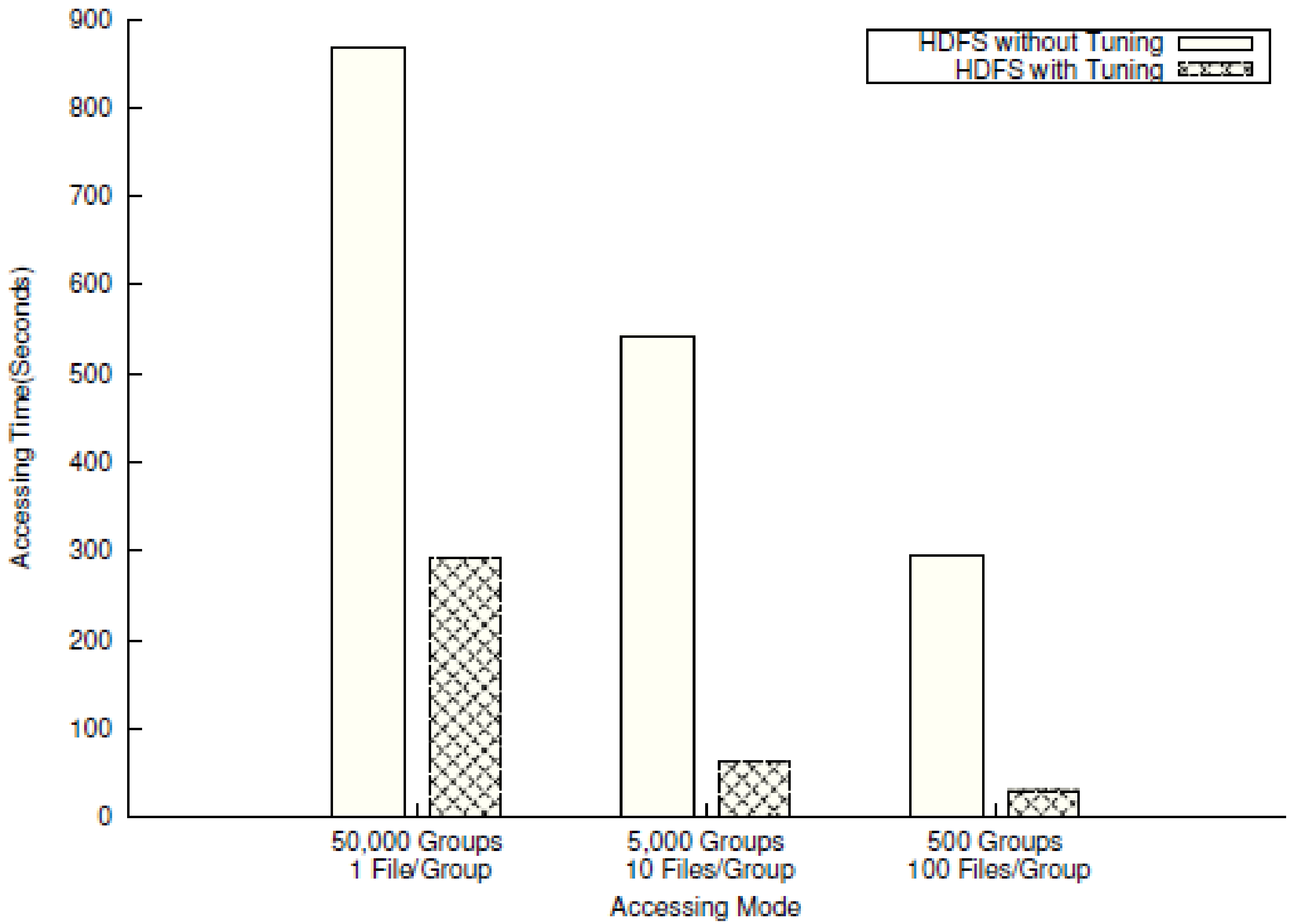
Fig. 6. Distribution of small files' sizes

More than 96% files are smaller than 16KB

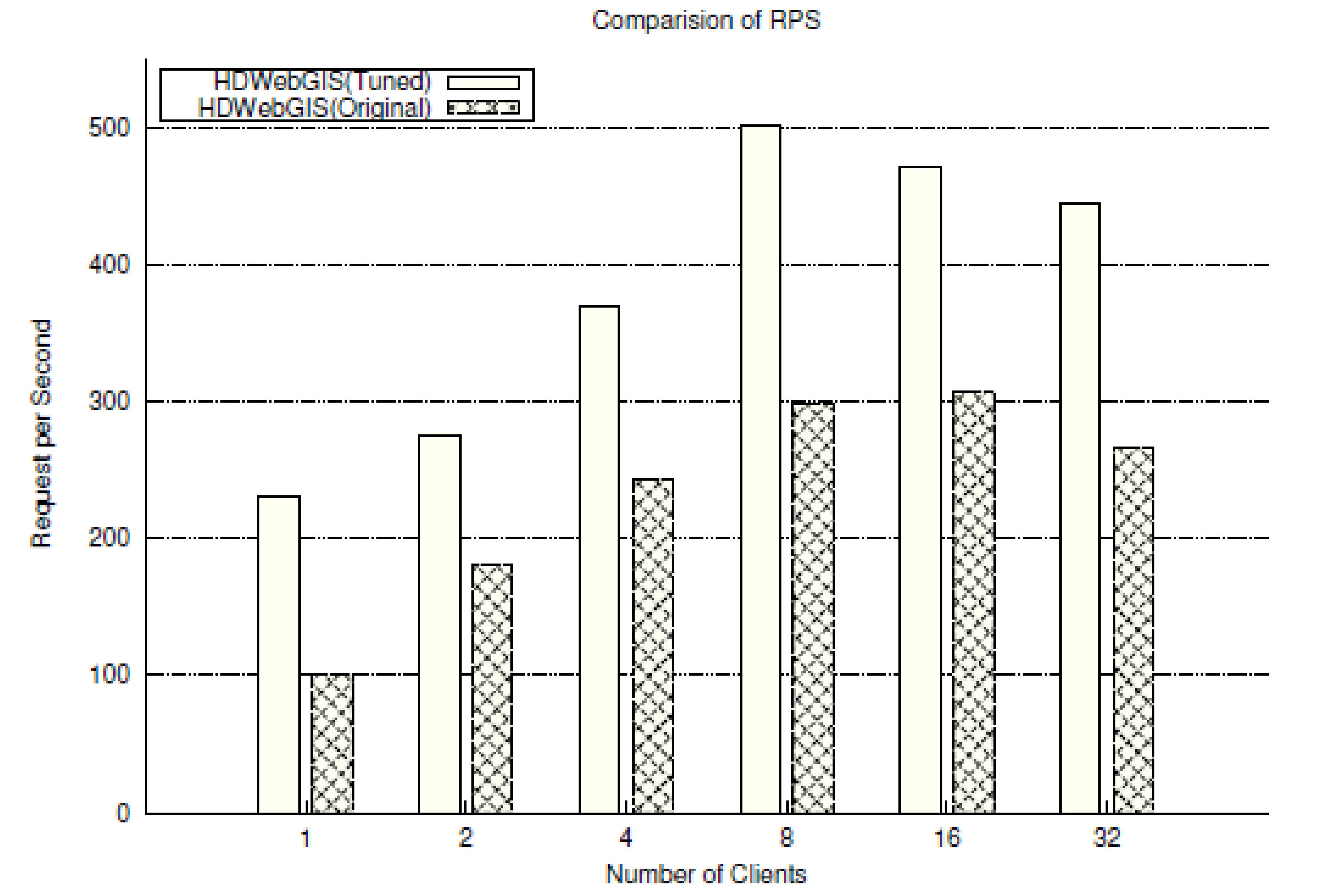
Performance Evaluation -File Write Operation Results

Original HDFS	Tuned HDFS
22,719 seconds	431 seconds

Performance Evaluation -File Read Operation Results



Performance Evaluation -System Perspective



Conclusions

- Improvements of HDFS
  - Write: 27,719 seconds vs. 431 seconds.
  - Read: 867 seconds vs. 292 seconds.
  - Memory usage: 55.78% vs. 18.36% on average.
- Tuning method for HDFS is feasible and efficient for applications which are sensitive to small file I/O performance.