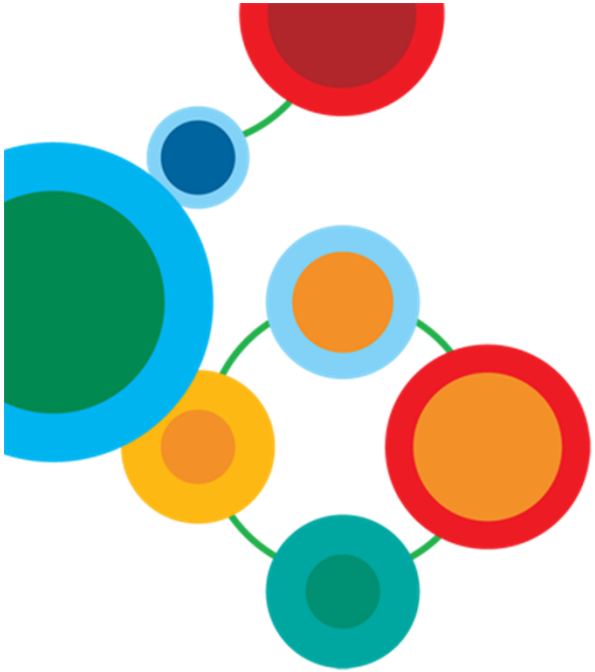


Enabling Multi-pipeline Data Transfer in HDFS for Big Data Applications

Liqiang (Eric) Wang, Hong Zhang
University of Wyoming

Hai Huang
IBM T.J. Watson Research Center

UNIVERSITY OF WYOMING

Background

- Hadoop: Apache Hadoop is a popular open-source implementation of the MapReduce programming model to handle large data sets.
- Hadoop has two main components: MapReduce and Hadoop Distributed File System (HDFS).
- In our research, we focused on how to optimize the performance of uploading data to HDFS.

Motivation

- HDFS is inefficient when handling upload of data files from client local file system due to its synchronous pipeline design.
- Original HDFS transfers data blocks one by one and waiting for ACK (acknowledgement) packets from all datanodes involved in the transmission

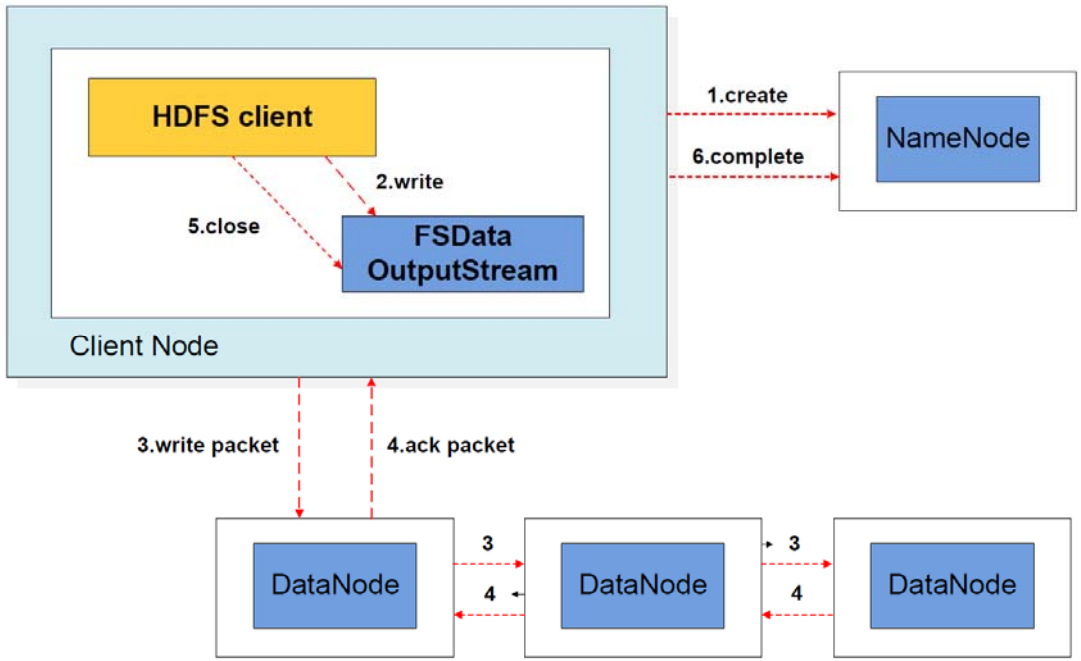
Objectives

- To introduce an innovative asynchronous data transmission approach to greatly improve the write operation's performance in HDFS
- To support flexible sort of datanodes in pipelines based on real-time and historical datanode accessing condition
- To provide a comprehensive fault tolerance mechanism under this asynchronous transmission approach

Optimization of Big Data System

[ICPP 2014]

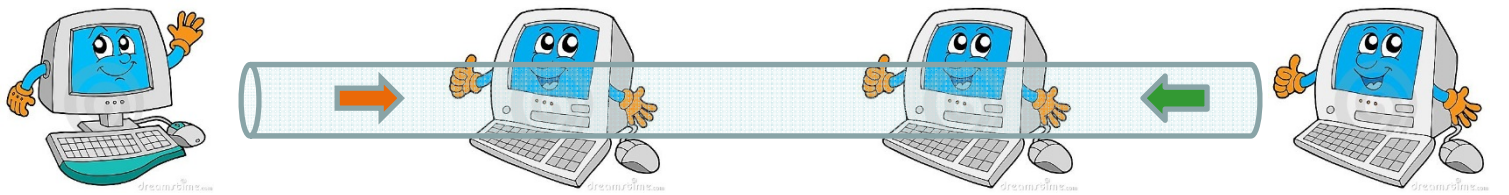
- Hadoop: Open-source implementation of MapReduce
  - MapReduce
  - Hadoop Distributed File System (HDFS).
- Performance Issue with HDFS
  - HDFS write is much slower than SCP.



Problem Scrutinizing

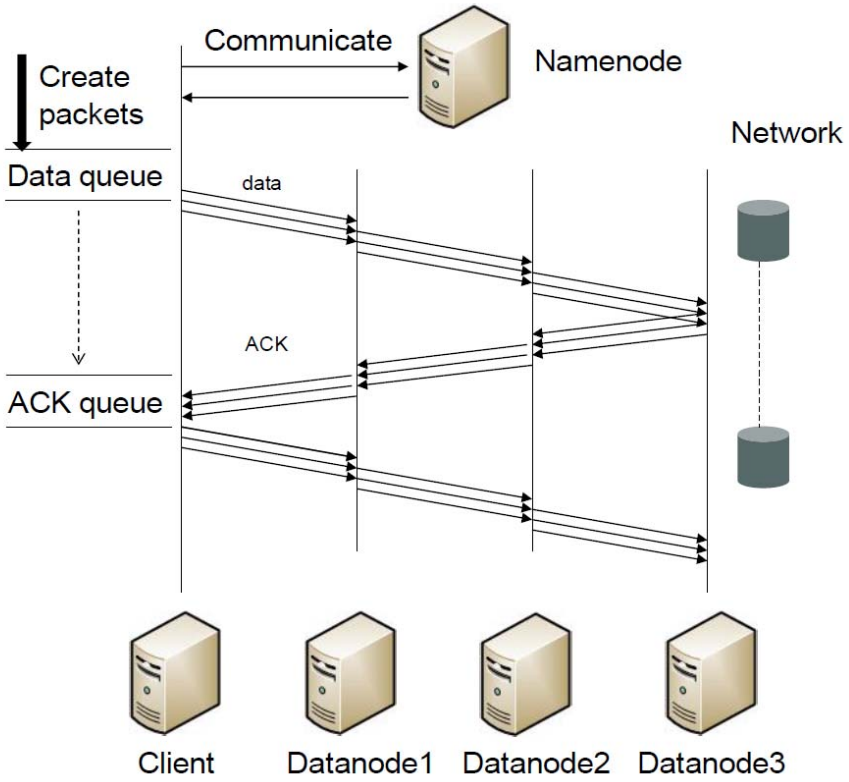
- Finding the problem is not easy
  - No detailed document available.
  - Need to read source code
  - Profiling and Testing
- Reason for slow upload performance
  - data block transmission mechanism :

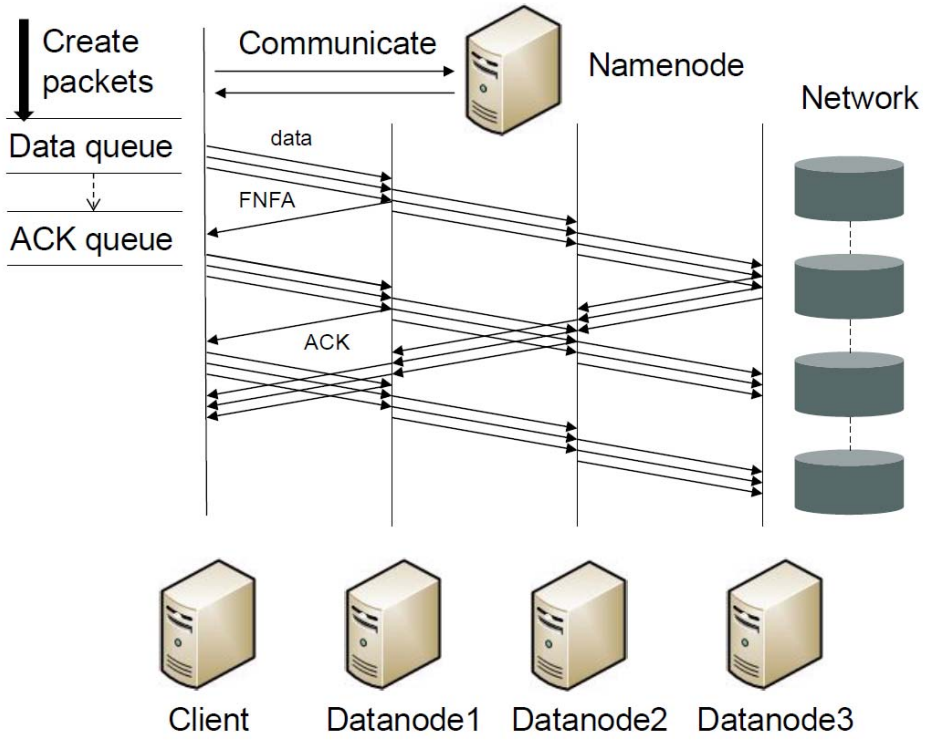
    Synchronous pipelined stop-and-wait



Our Own Approach

- Asynchronous Multi-pipelined Data Transfer





Optimization for Data Transmission

Algorithm 1 Algorithm for global optimization

```
1: num = the number of active datanodes
2: repli = the number of replica factor
3: n = num / repli // the maximum pipeline size
4: if (namenode has transmission records for the client)
   then
5:     TopN = top n datamodes in terms of transfer speed
6:     // the number of datanodes we have choosen
7:     results = 0
8:     while (results != repli) do
9:         if (results == 0) then
10:            targets[0] = randomDatanode(TopN)
11:        else if (results == 1) then
12:            targets[1] = randomRemoteRackNode()
13:        else if (results == 2) then
14:            targets[2] = nodeOnSameRack(targets[1])
15:        else
16:            targets[results] = randomDatanode()
17:        end if
18:        results++
19:    end while
20: else
21:    targets = employ the original HDFS method to
   select datanodes
22: end if
```

To selects a datanode randomly from the n best performing nodes for this client as the first datanode

Randomly choose data nodes for the replicas..

Datanode Exchange

Algorithm 2 Algorithm for local optimization

```
1: repli = the number of replica factor
2: TransSpeedVector = the transmission speed of every
   nodes in targets
3: sort targets in descending order by
   TransSpeedVector
4: r = a random number between 0 to 1
5: if (r > threshold) then
6:     //the target index to switch the first datanode
7:     index = a random integer between 1 to repli − 1
8:     swap(targets[0], targets[index])
9: end if
```

To decide whether to swap the first datanode with another datanode in the pipeline in order to give a chance to all nodes.

Fault tolerance for HDFS

Algorithm 3 Algorithm for fault tolerance of HDFS

```
1: checks the validity of parameters
2: close all streams related to the block
3: moves all packets in ACK queue back to data queue
4: success = false
5: while (!success) do
6:     if (targets is not empty) then
7:         return an exception
8:     else
9:         primaryNode = the first datanode in targets
10:        add new datanodes to replace error nodes in
   targets
11:        success=recoverBlock(primaryNode, targets)
12:        if (!success) then
13:            remove primaryNode from targets
14:        end if
15:        recreate block streams
16:    end if
17: end while
18: recreate ResponseProcessor thread
```

- To check the validity of parameters, close all streams related to the block. move all packets in ACK queue back to data Queue
- To pick the primary datanode from active datanodes in pipeline, and use it to recover the other datanodes.

Fault tolerance for SMARTH

Algorithm 4 Algorithm for fault tolerance of SMARTH

```
1: stop the current block transfer
2: moves all packets in ACK queue back to data queue
3: while (errorPipelineSet is not empty) do
4:     recover one error pipeline as Algorithm 3
5:     remove the error pipeline from errorPipelineSet
6: end while
7: start transferring the interrupted block
```

To stops the current block sending, and starts a recovery process as Alg. 3 to recover error pipelines in error pipeline set.

Buffer Overflow Problem

- Two Conditions
  - Limit the pipeline size to a maximum number (the cluster size / the number of replica),
  - And if a datanode is already in a pipeline, it cannot be added into other pipelines created by the same client.
- Result:
  - Each datanode belongs to only one pipeline

Data imbalance problem

- Conditions
  - Always choose a random datanode from Top N as the first datanode (N = the cluster size / the number of replica),
  - And select other datanodes from left active datanodes.
- Hence there is no imbalance problem.

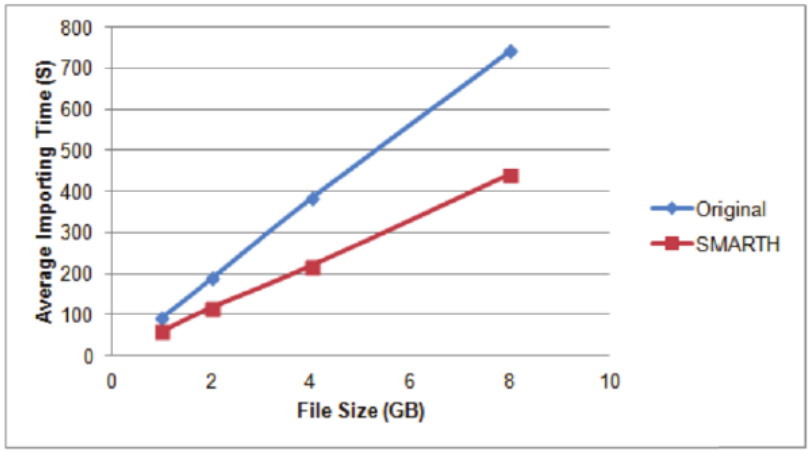
Experiments – Setup

Instance Type	Memory	ECUs	Network
Small	1.7 GB	1	$\approx 216Mbps$
Medium	3.75 GB	2	$\approx 376Mbps$
Large	7.5 GB	4	$\approx 376Mbps$

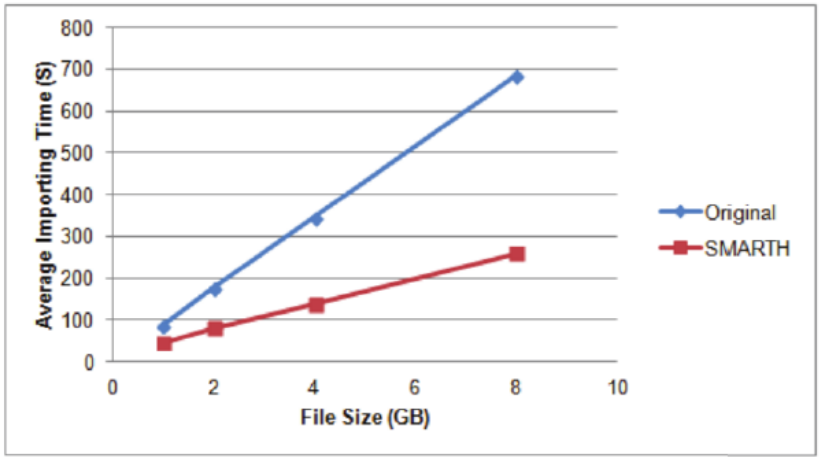
Table I
AMAZON EC2 INSTANCE TYPES

We use four different clusters in our evaluations. Three of the clusters are homogeneous consisted of one namenode and nine datanodes, i.e., of small, medium, or large instances. The other cluster is heterogeneous consisted of 3 small, 4 medium, and 3 large instance nodes

Experiments – Two-Rack Cluster



(b) bandwidth throttling in small cluster



(d) bandwidth throttling in medium clustere

Experiments – Two-Rack Cluster

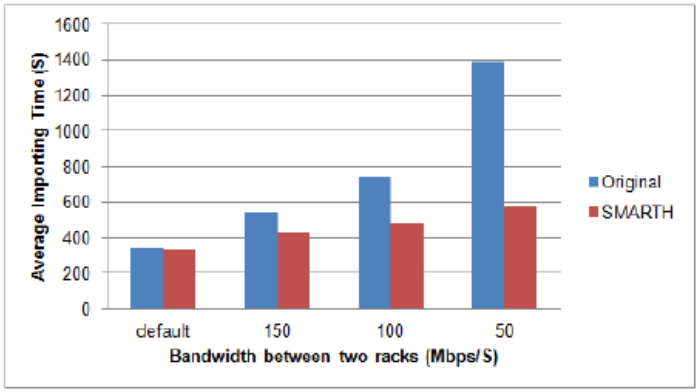


Figure 6. Comparison of small instances' uploading time when throttled bandwidth between two racks varies.

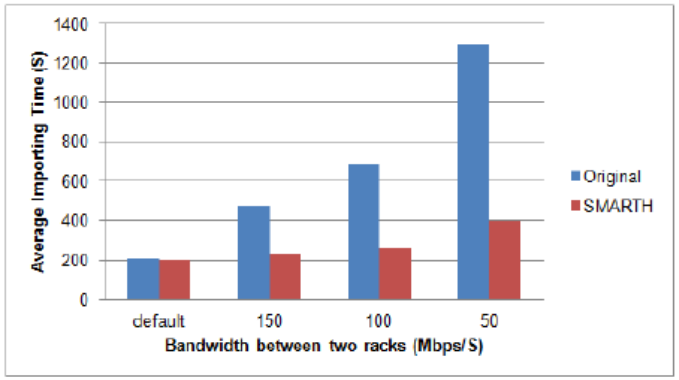


Figure 7. Comparison of medium instances' uploading time when throttled bandwidth between two racks varies.

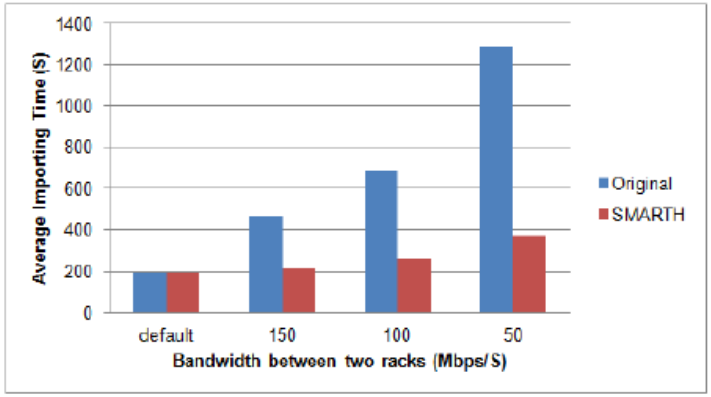


Figure 8. Comparison of large instances' uploading time when throttled bandwidth between two racks varies

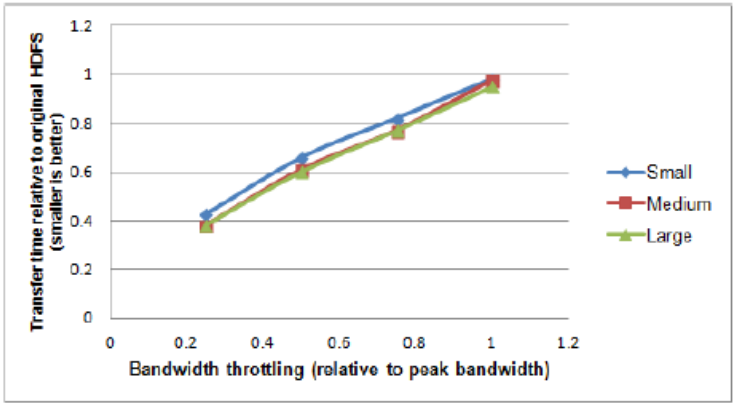
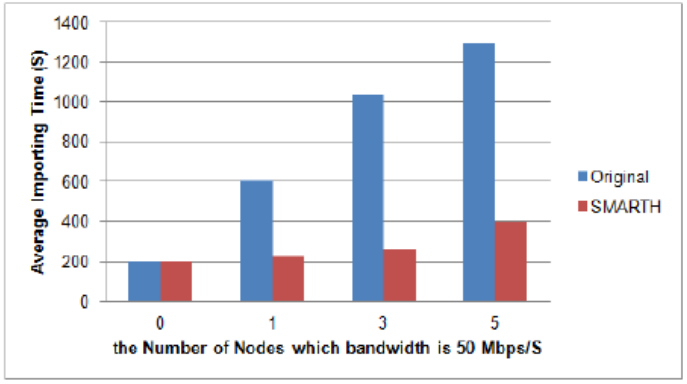
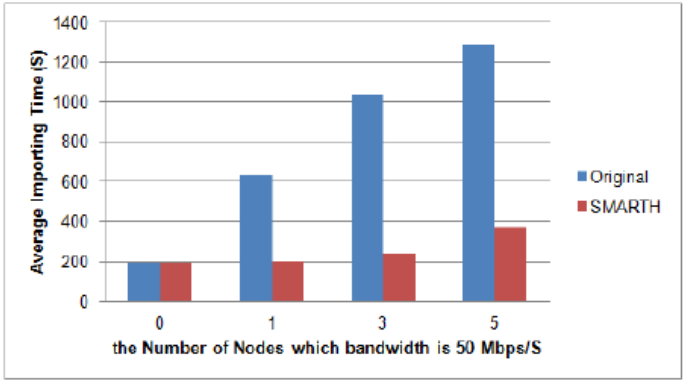


Figure 9. Relationship between bandwidth throttling and performance improvement.

Experiments – Bandwidth Contention

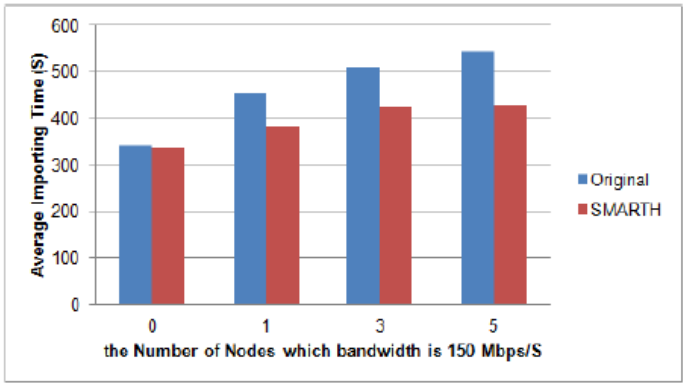


(a) medium cluster

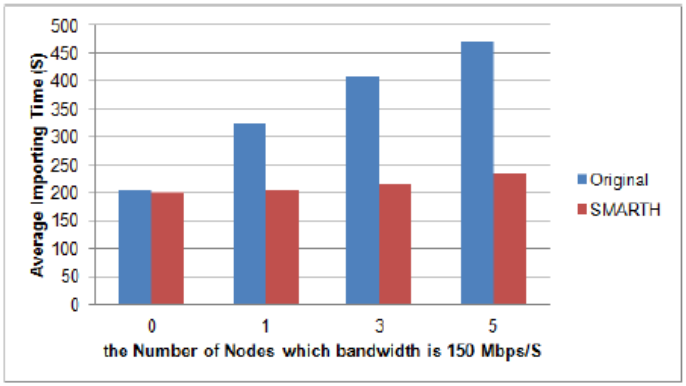


(b) large cluster

Figure 11. Comparison of uploading time for medium and large clusters when the number of nodes with 50Mbps throttling varies.



(a) small cluster



(b) medium cluster

Figure 12. Comparison of uploading time for small and medium clusters when the number of nodes with 150Mbps throttling varies.

Experiments – Heterogeneous Cluster

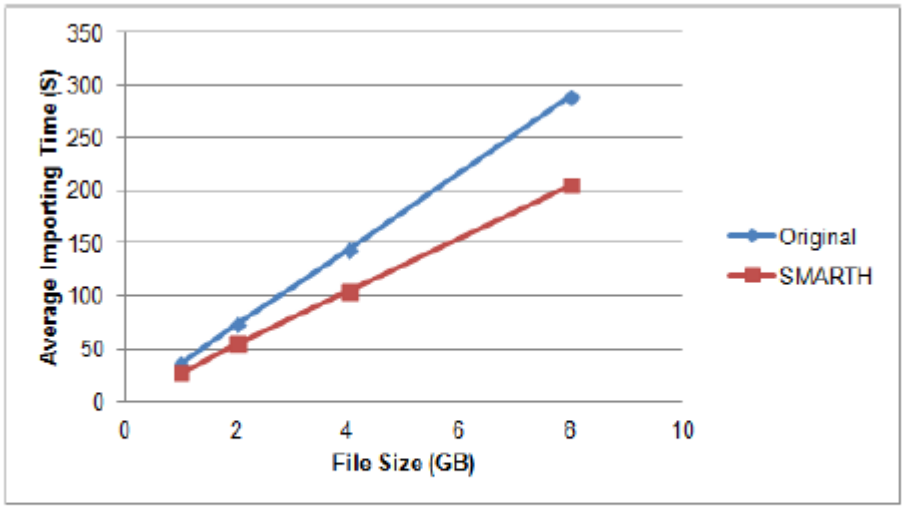


Figure 13. Comparison of uploading time of different data size in a heterogeneous cluster.

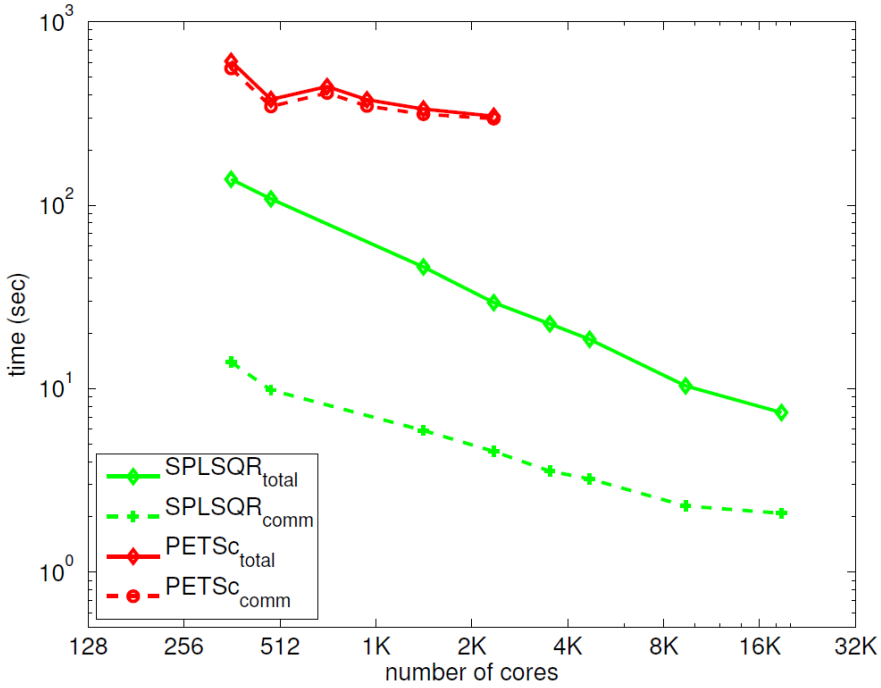
Without any network throttling, Figure 13 shows that it takes 289 seconds to upload an 8 GB file in HDFS, but SMARTH only takes 205 seconds, which is 41% faster.

Conclusion

- To introduce an asynchronous multi-pipeline file transfer protocol with a revised fault tolerance mechanism instead of the HDFS's default stop-and-wait single-pipeline protocol.
- To employ global and local optimization techniques to sort datanodes in pipelines based on the historical data transfer speed.

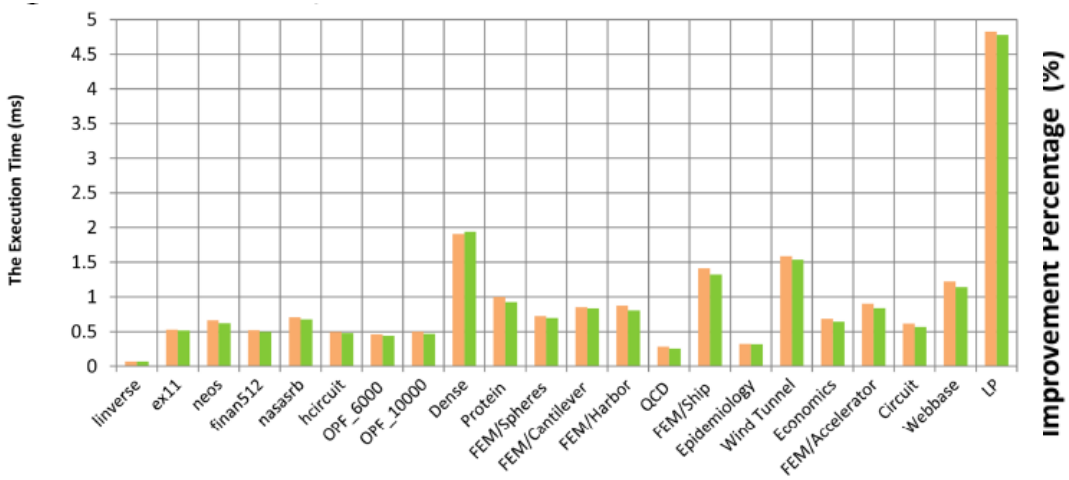
SPLSQR: Optimizing HPC Performance and Scalability (ICCS 2012, 2013, Collaborative Project with Dr. John Dennis, NCAR)

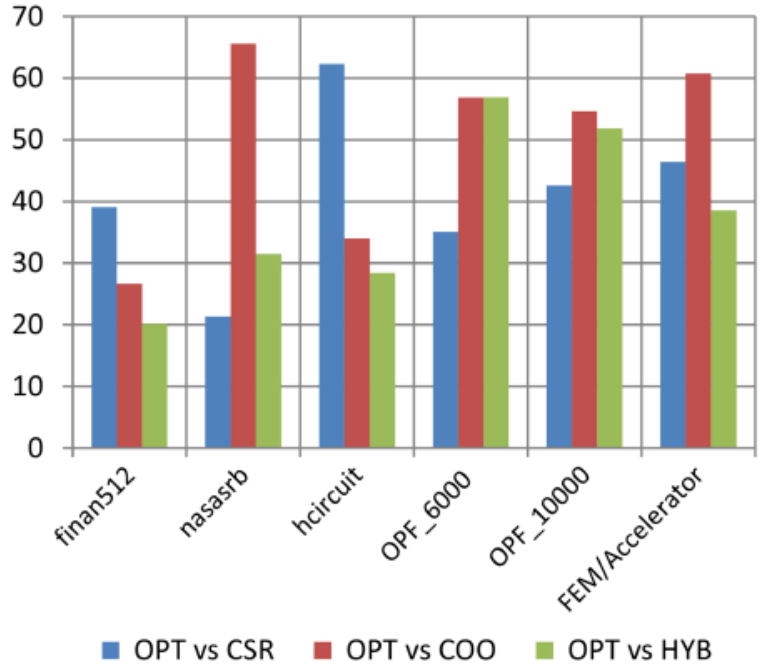
- A Scalable Parallel LSQR (SPLSQR) algorithm for solving large-scale linear system in seismic tomography and reservoir simulation.
  - Main idea: optimize partition strategy to significantly reduce the communication cost and improve the overall performance.
  - Much faster (17-74x) than the widely-used PETSc
  - Reported by NCSA magazine & NSF (http://www.nsf.gov/news/news_summ.jsp?cntn_id=128020&org=NSF&from=news)



GPU Acceleration (TeraGrid11, HPCS12, IEEE TPDS)

- Based on our accurate GPU performance modeling, optimize the performance of SpMV (Sparse Matrix-Vector Multiplication).
- Main idea: a sparse matrix can be partitioned into blocks with optimal storage formats, which dramatically affect performance.





(Left) Accuracy of Performance model is around 9%
(Right) Performance Improvement are around
41%, 50%, 38%, respectively.

Thank you!