Tachyon: Reliable File Sharing at Memory-Speed Across Cluster Frameworks

Haoyuan Li

UC Berkeley

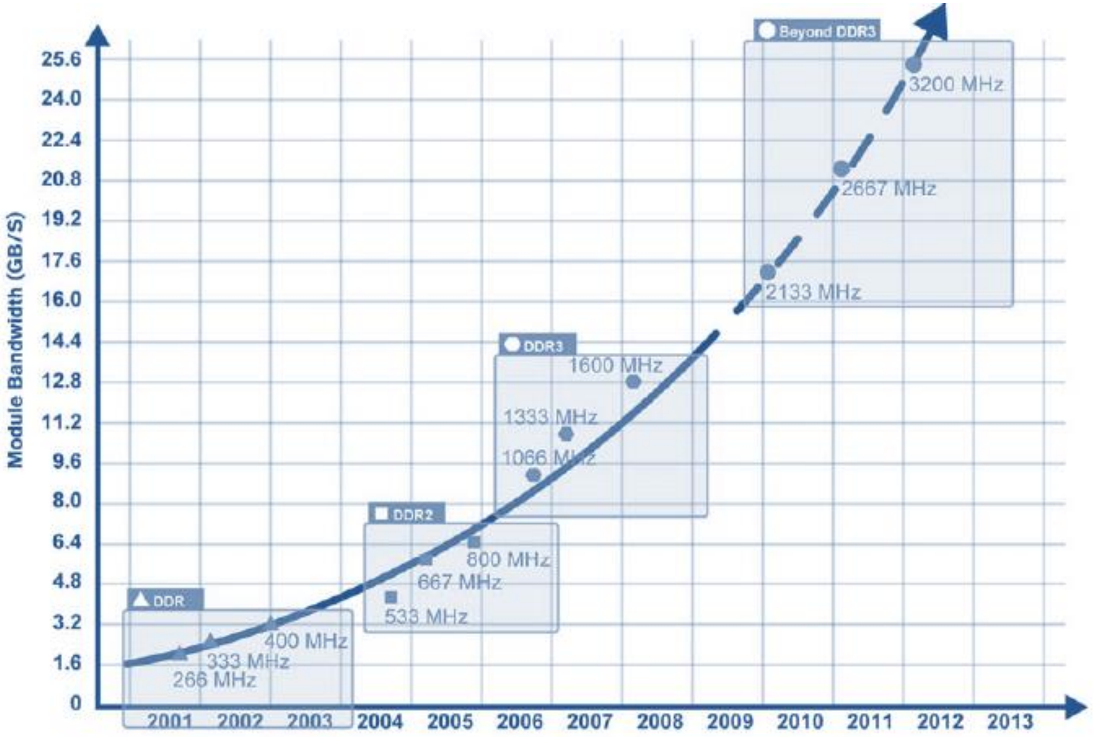
Outline

- Motivation
- System Design
- Evaluation Results
- Release Status
- Future Directions

Memory is King

Memory Trend

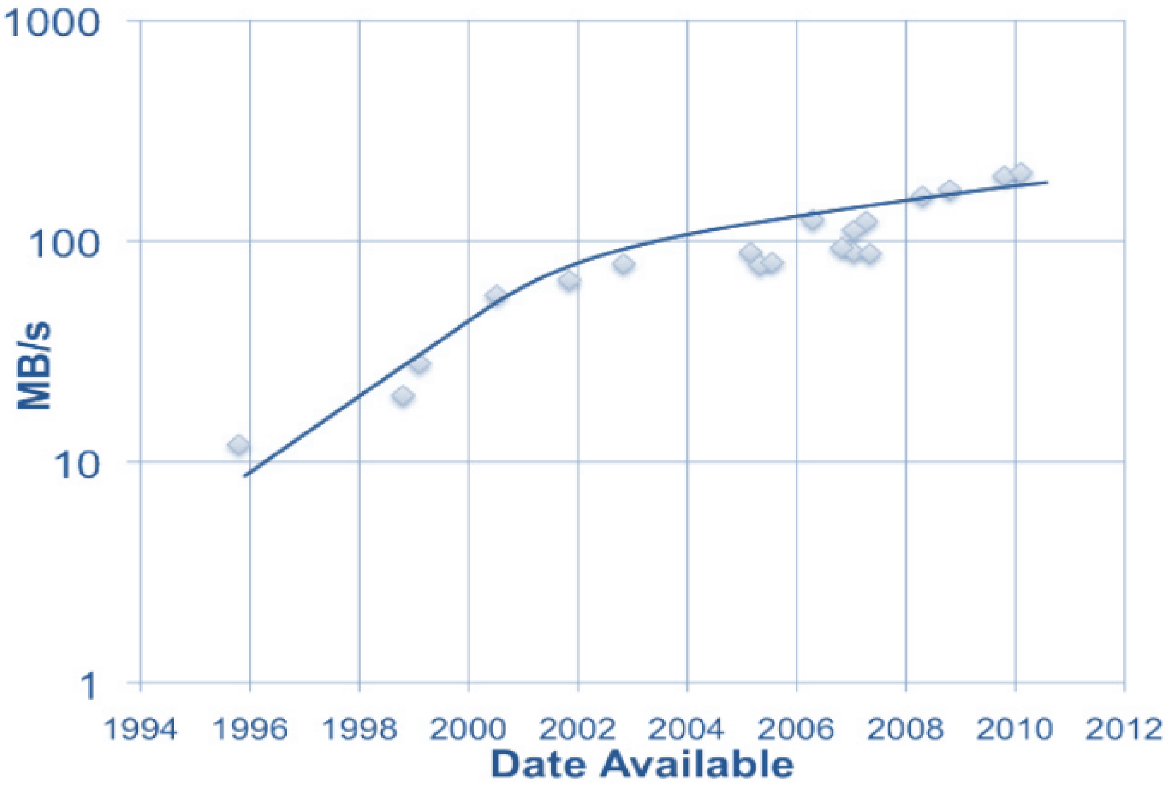
- RAM throughput increasing exponentially



Bandwidths shown for 64-bit memory module. Date indicates approximate industry product introduction.

Disk Trend

- Disk throughput increasing slowly



Consequence

- Memory locality **key** to achieve
  - Interactive queries
  - Fast query response

Current Big Data Eco-system

- Many frameworks already leverage memory
  - e.g. Spark, Shark, and other projects
- File sharing among jobs replicated to disk
  - Replication enables fault-tolerance
- **Problems**
  - Disk scan is slow for read.
  - Synchronous disk replication for write is even slower.

Tachyon Project

- Reliable file sharing at memory-speed across cluster frameworks/jobs

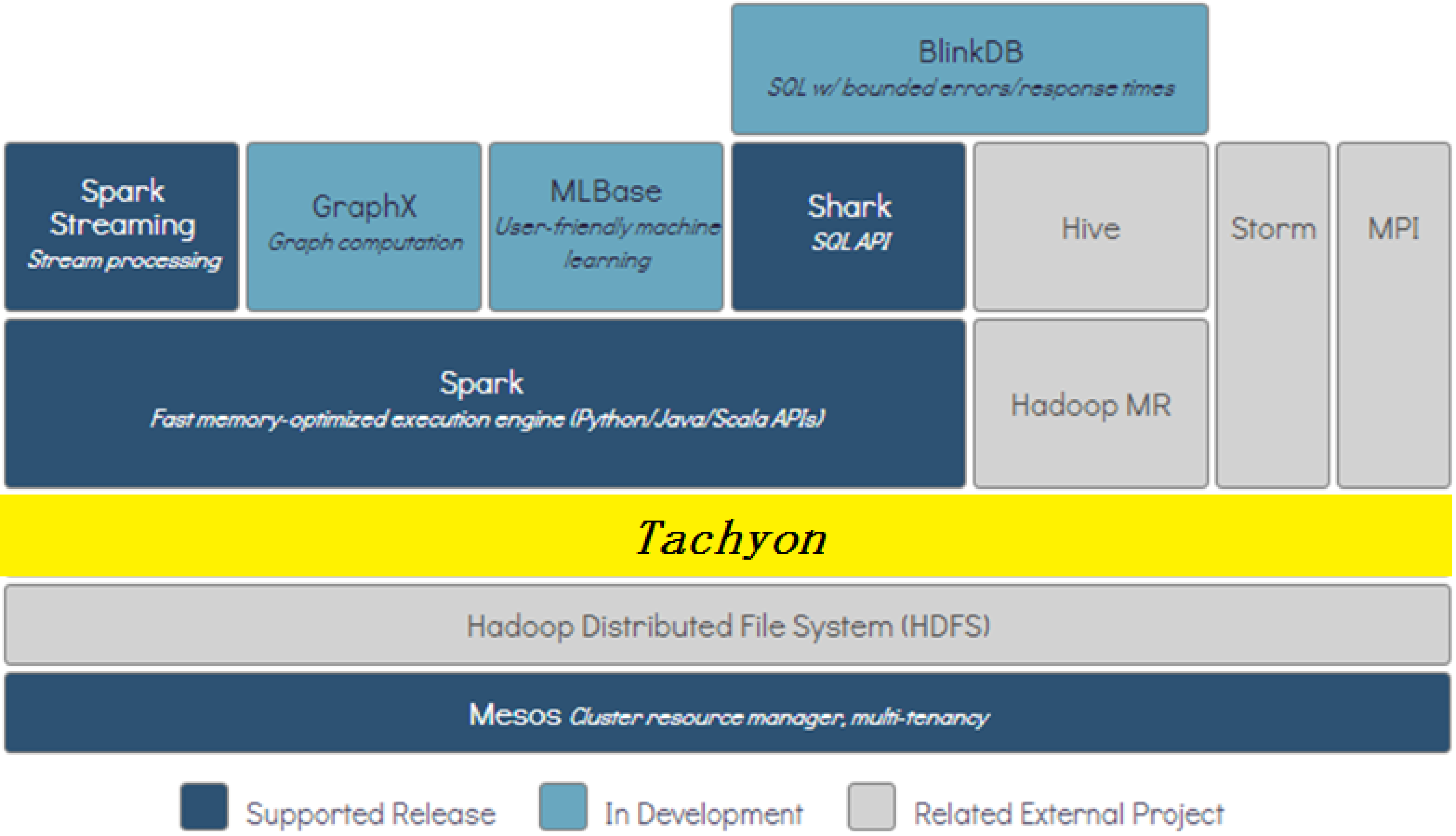
- Challenge
  - How to achieve reliable file sharing without replication?

Idea

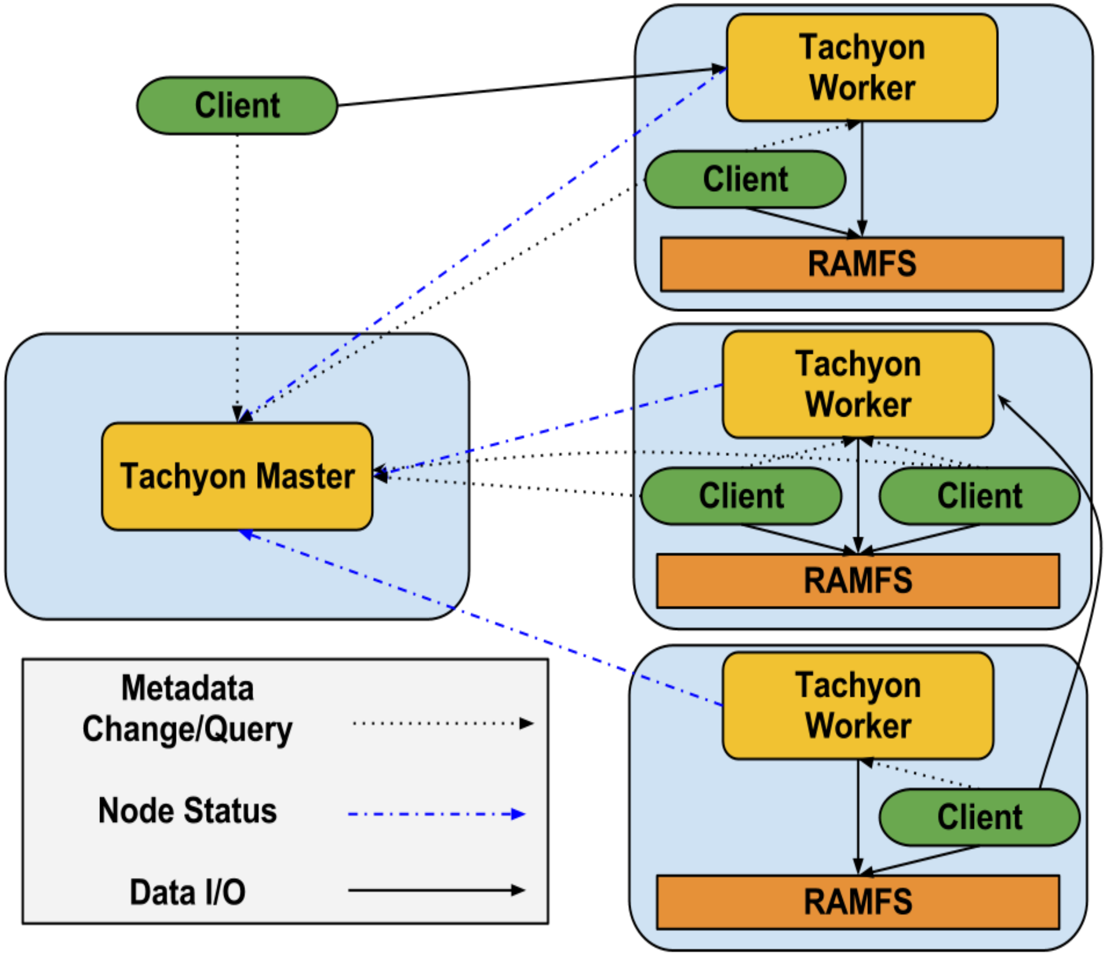
Re-computation (Lineage) based storage using memory aggressively.

1. One copy of data in memory (Fast)
2. Upon failure, re-compute data using ***lineage*** (Fault tolerant)

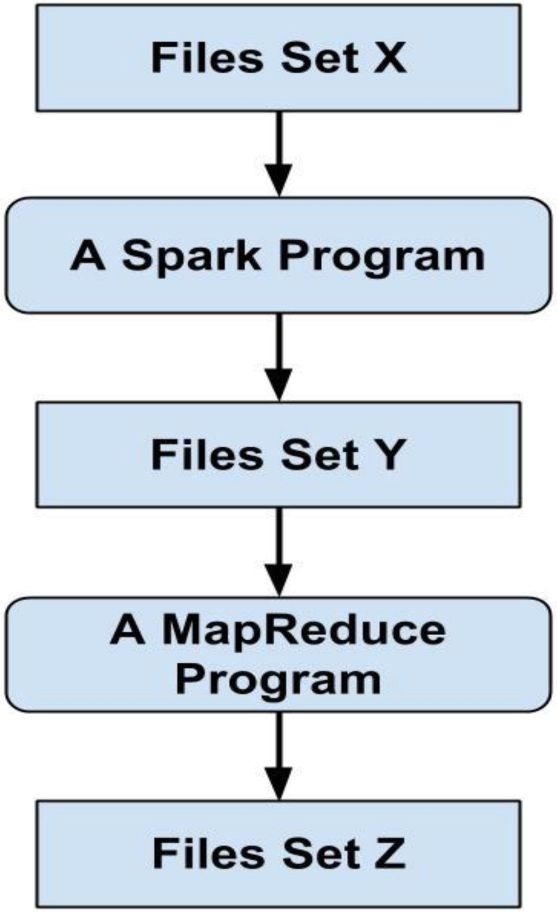
Stack


BlinkDB
SQL w/ bounded errors/response times
Spark Streaming
Stream processing
GraphX
Graph computation
MLBase
User-friendly machine learning
Shark
SQL API
Hive
Storm
MPI
Spark
Fast memory-optimized execution engine (Python/Java/Scala APIs)
Hadoop MR
Tachyon
Hadoop Distributed File System (HDFS)
Mesos Cluster resource manager, multi-tenancy
Supported Release
In Development
Related External Project

System Architecture



Lineage



Files Set X

A Spark Program

Files Set Y

A MapReduce Program

Files Set Z

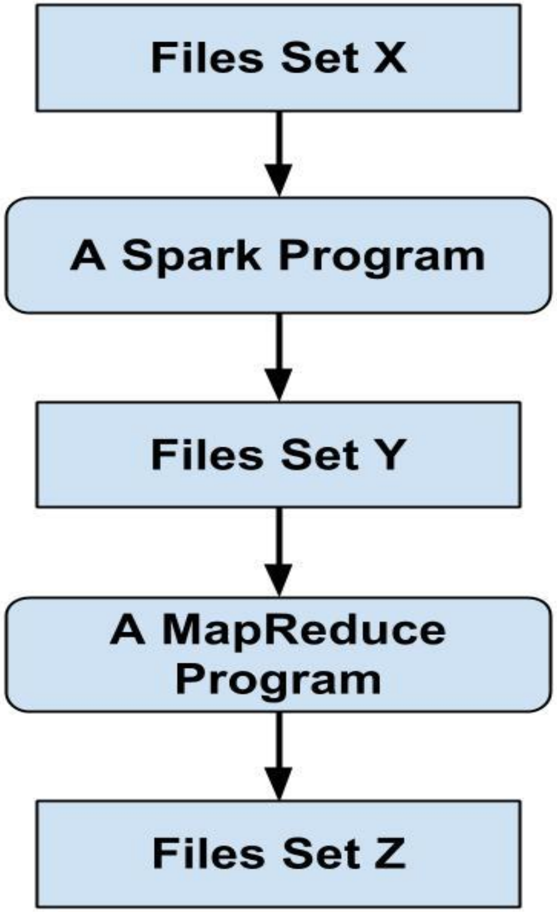
Lineage Information

- Binary program
- Configuration
- Input Files List
- Output Files List
- Dependency Type

Fault Recovery Time

Re-computation Cost?

Example


Files Set X

A Spark Program

Files Set Y

A MapReduce Program

Files Set Z

Asynchronous Checkpoint

1. Better than using existing solutions even under failure.
2. Bounded recovery time (Naïve and Snapshot asynchronous checkpointing).

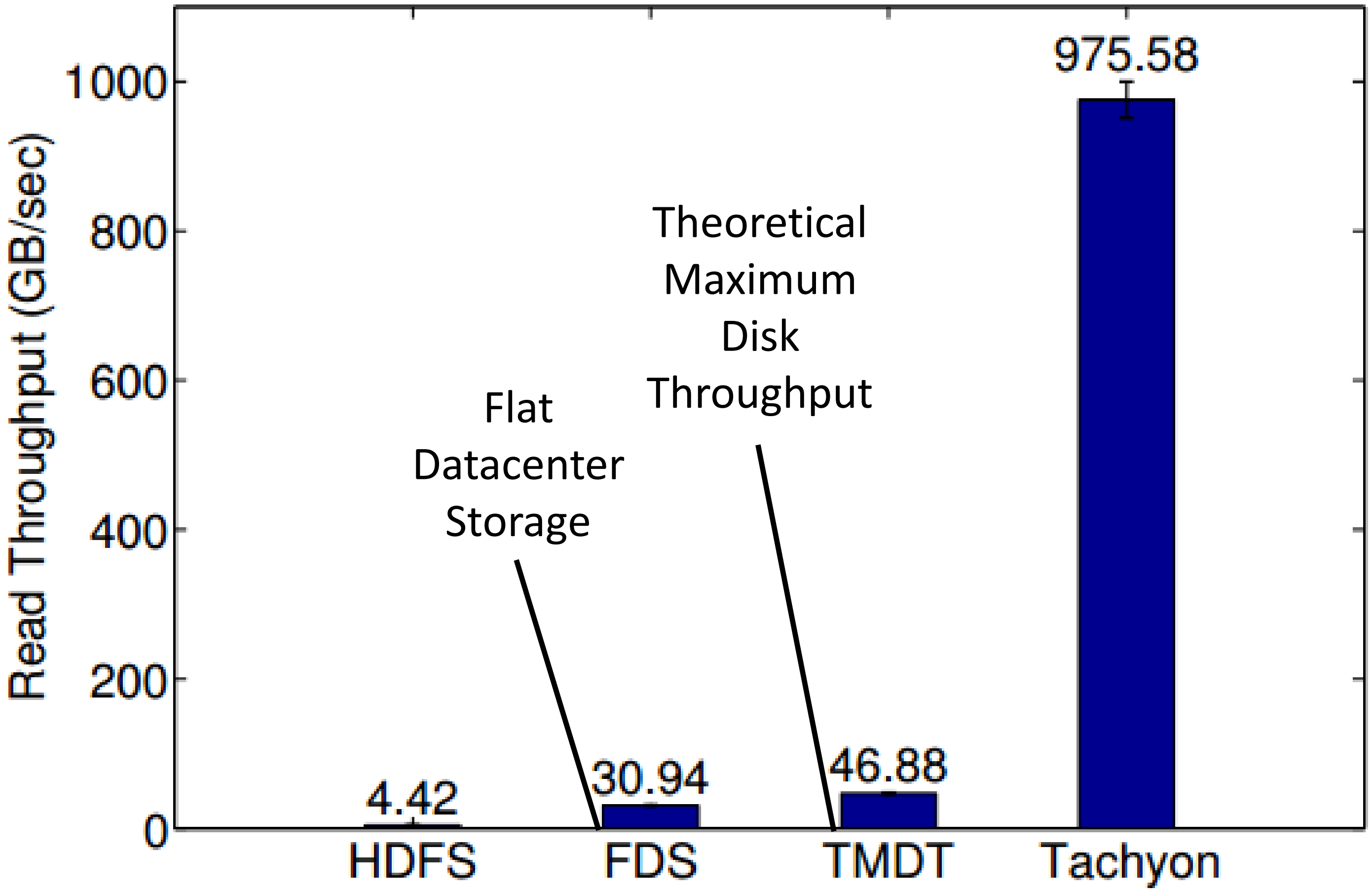
Master Fault Tolerance

- Multiple masters
  - Use ZooKeeper to elect a leader
- After crash workers contact new leader
  - Update the state of leader with contents of caches

Implementation Details

- 15,000+ lines of JAVA
- Thrift for data transport
- Underlayer file system supports HDFS, S3, localFS, GlusterFS
- Maven, Jenkins

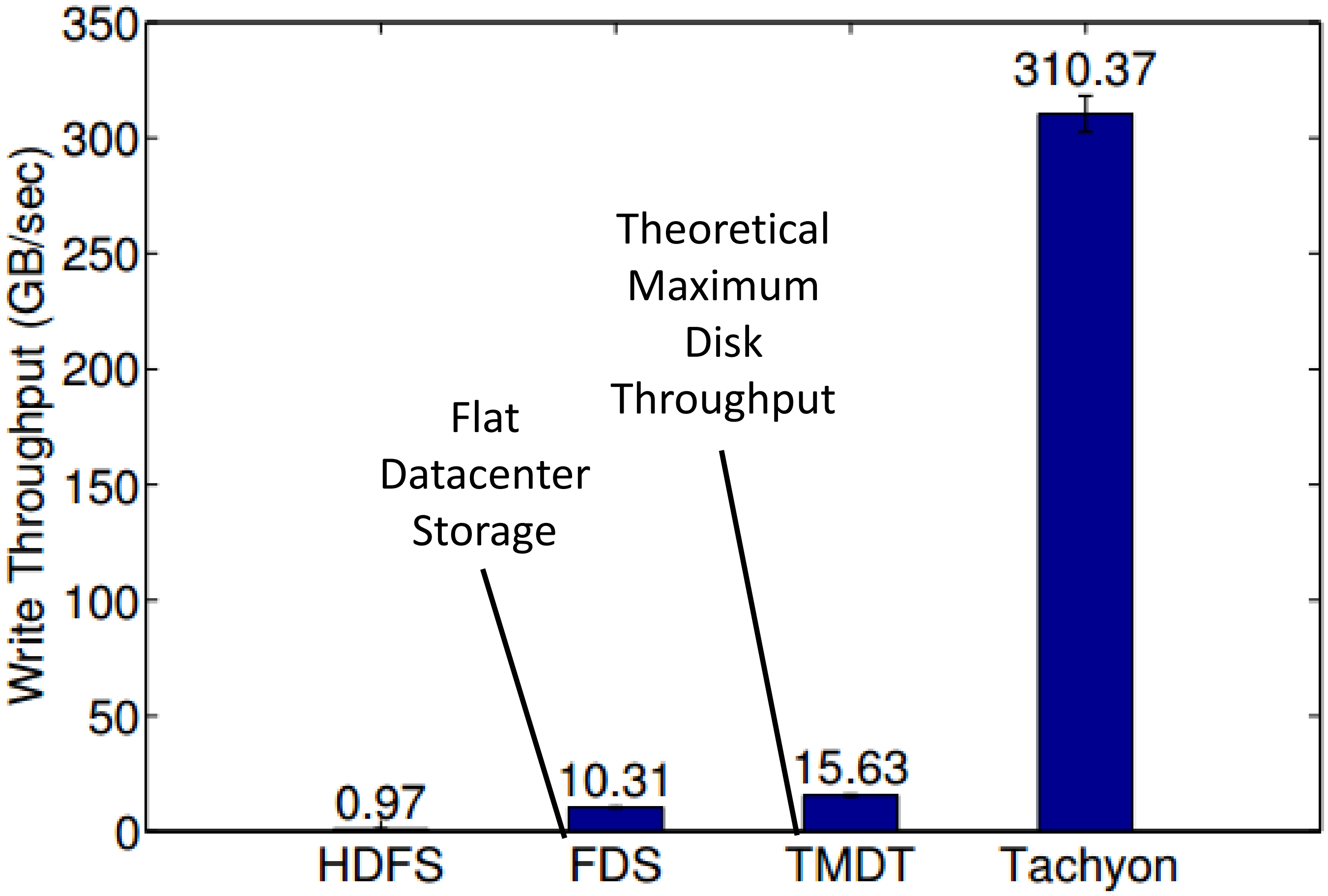
Sequential Read using Spark



Flat Datacenter Storage

Theoretical Maximum Disk Throughput

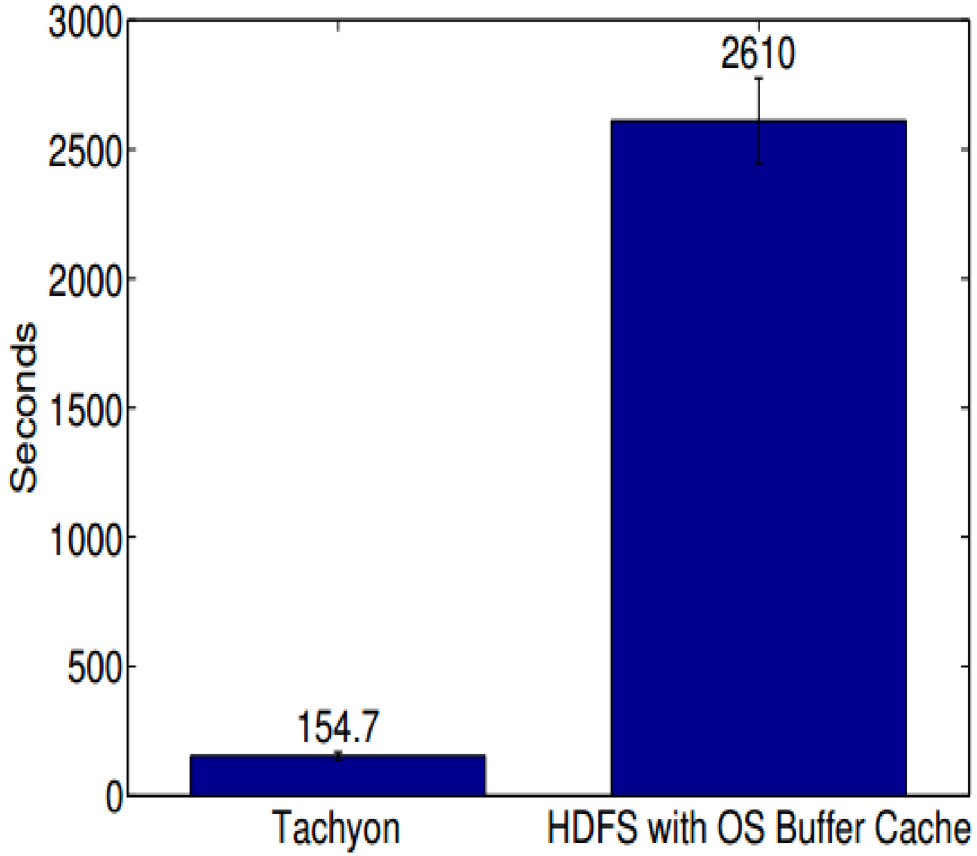
Sequential Write using Spark



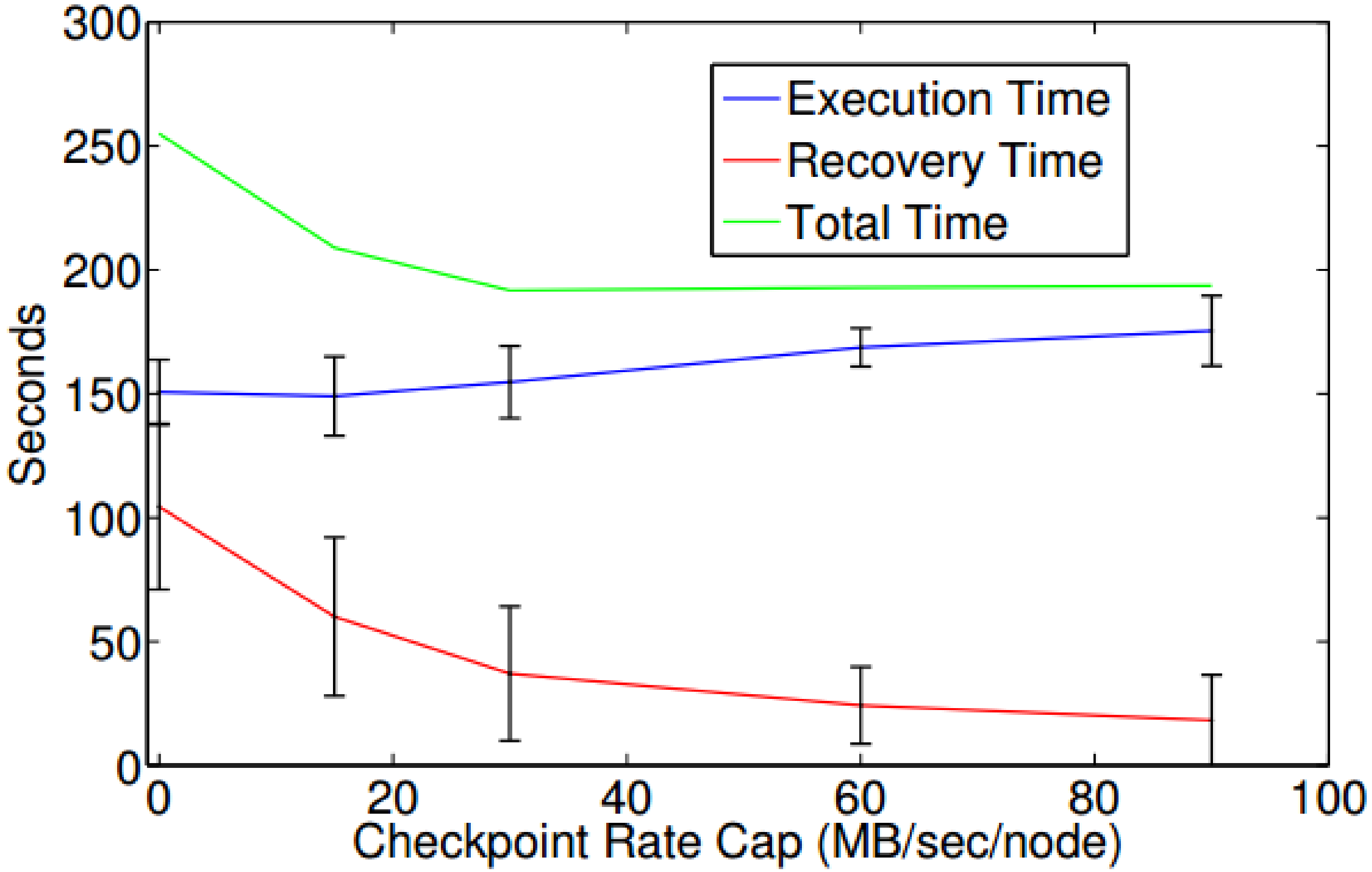
Flat Datacenter Storage

Theoretical Maximum Disk Throughput

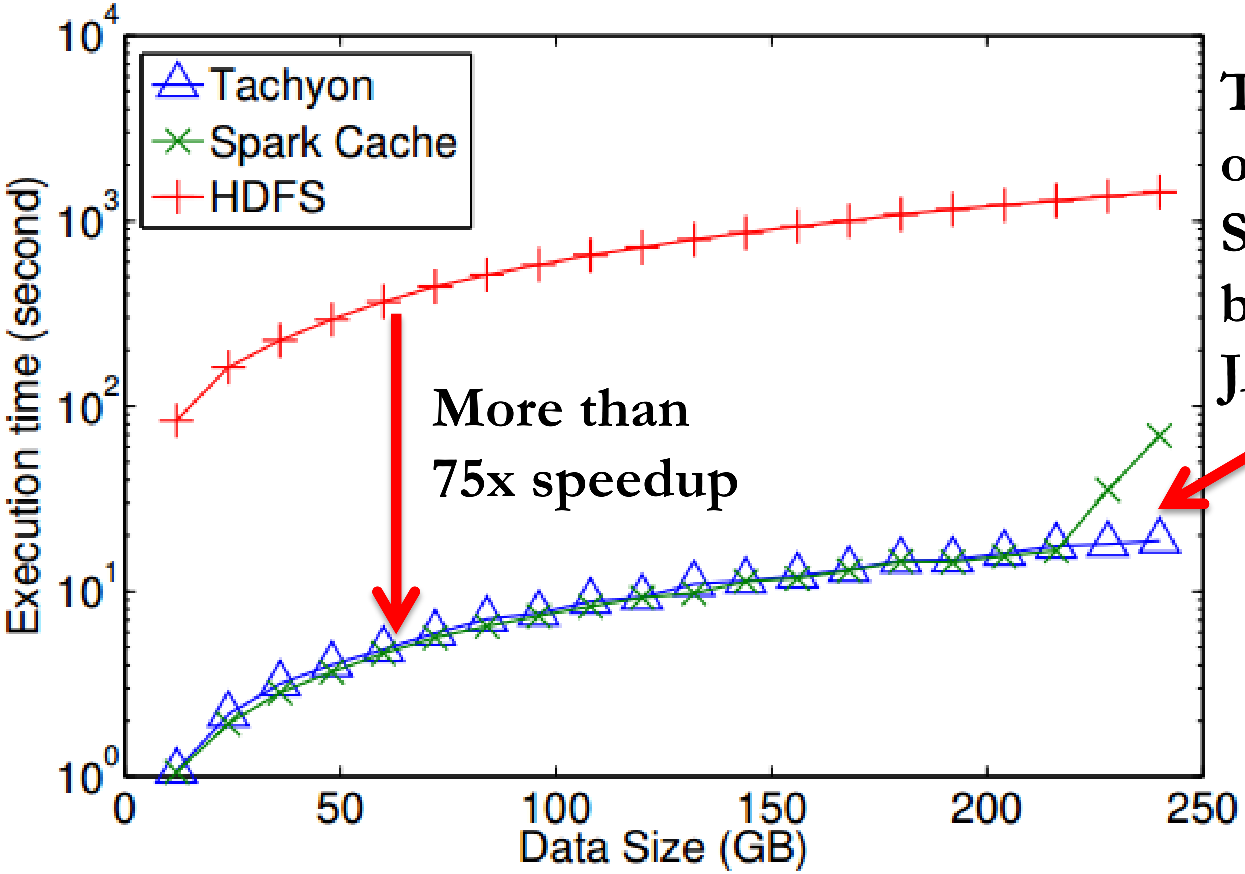
Realistic Workflow using Spark



Realistic Workflow Under Failure



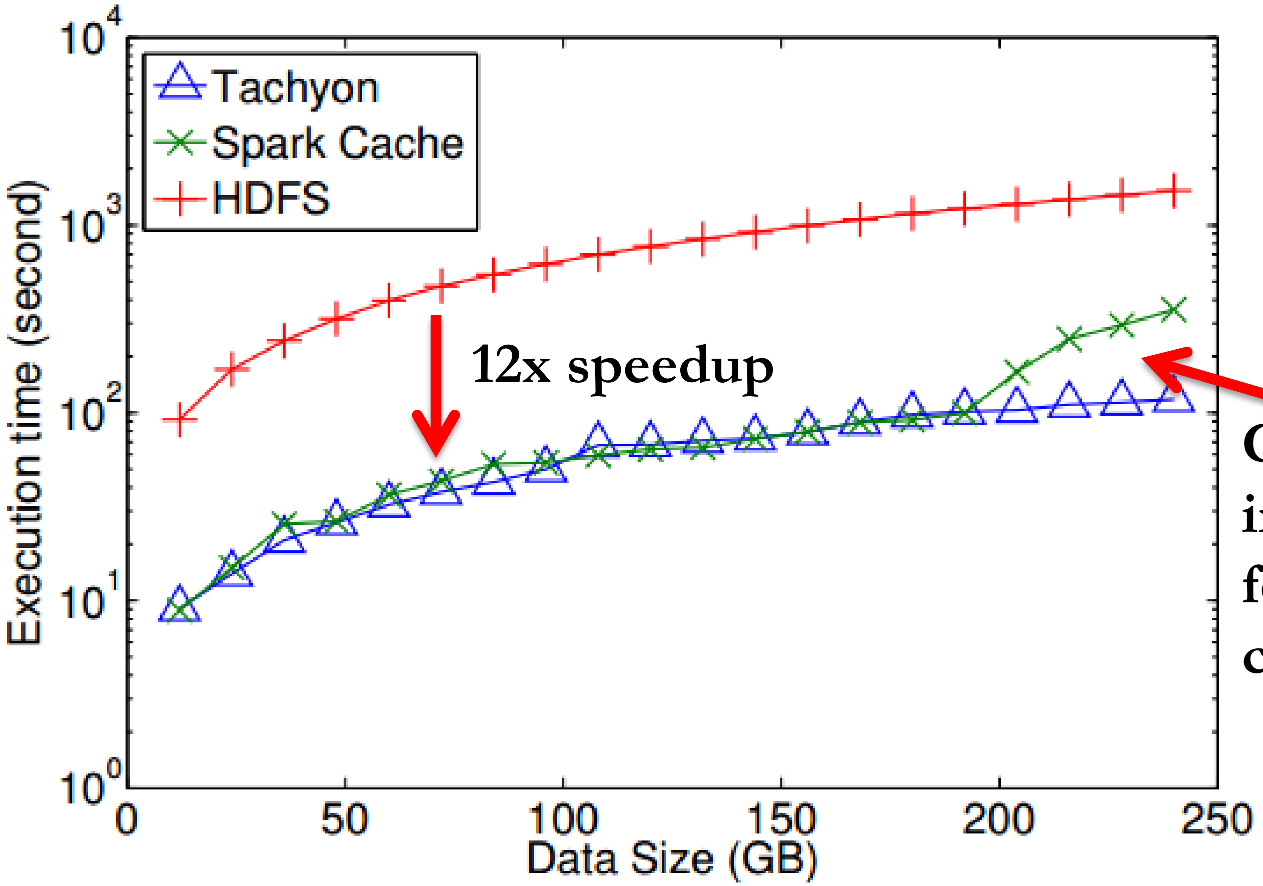
Conviva Spark Query (I/O intensive)



More than 75x speedup

Tachyon outperforms Spark cache because of JAVA GC

Conviva Spark Query (less I/O intensive)



12x speedup

GC kicks in earlier for Spark cache

Alpha Status

- Releases
  - Developer Preview: V0.2.1 (4/25/2013)
  - Contributions from:

YAHOO!

ClearStory DATA
Now You See It™

CONVIVA®

NANJING UNIVERSITY 1902

THE UNIVERSITY OF CALIFORNIA BERKELEY 1868 LET THERE BE LIGHT

Alpha Status

- First read of files cached in-memory
- Writes go synchronously to HDFS (No lineage information in Developer Preview release)
- MapReduce and Spark can run without any code change (ser/de becomes the new bottleneck)

Current Features

- Java-like file API
- Compatible with Hadoop
- Master fault tolerance
- Native support for raw tables
- WhiteList, PinList
- Command line interaction
- Web user interface

Spark without Tachyon

```
val file = sc.textFile(“hdfs://ip:port/path”)
```

Spark with Tachyon

```
val file = sc.textFile(“tachyon:// ip:port/path”)
```

Shark without Tachyon

```
CREATE TABLE orders_cached AS SELECT * FROM orders;
```

Shark with Tachyon

```
CREATE TABLE orders_tachyon AS SELECT * FROM orders;
```

Experiments on Shark

- Shark (from 0.7) can store tables in Tachyon with fast columnar Ser/De

20 GB data / 5 machines	Spark Cache	Tachyon
Table Full Scan	1.4 sec	1.5 sec
GroupBys (10 GB Shark Memory)	50 – 90 sec	45 – 50 sec
GroupBys (15 GB Shark Memory)	44 – 48 sec	37 – 45 sec

Experiments on Shark

- Shark (from 0.7) can store tables in Tachyon with fast columnar Ser/De

20 GB data / 5 machines	Spark Cache	Tachyon
Table Full Scan	1.4 sec	1.5 sec
GroupBys (10 GB Shark Memory)	50 – 90 sec	45 – 50 sec
GroupBys (15 GB Shark Memory)	44 – 48 sec	37 – 45 sec

4 * 100 GB TPC-H data / 17 machines	Spark Cache	Tachyon
TPC-H Q1	65.68 sec	24.75 sec
TPC-H Q2	438.49 sec	139.25 sec
TPC-H Q3	467.79 sec	55.99 sec
TPC-H Q4	457.50 sec	111.65 sec

Future

- Efficient Ser/De support
- Fair sharing for memory
- Full support for lineage
- Next release is coming soon

Acknowledgment

Research Team: Haoyuan Li, Ali Ghodsi, Matei Zaharia, Eric Baldeschwieler , Scott Shenker, Ion Stoica

Code Contributors: Haoyuan Li, Calvin Jia, Bill Zhao, Mark Hamstra, Rong Gu, Hobin Yoon, Vamsi Chitters, Reynold Xin, Srinivas Parayya, Dilip Joseph

Questions?

http://tachyon-project.org

https://github.com/amplab/tachyon