Make Tachyon ready for next-gen data center platforms with NVM

Bin Fan
Tachyon Nexus
binfan@tachyonnexus.com

Mingfei Shi
Intel
mingfei.shi@intel.com

Intel Cloud & BigData Engineering Team

- Work with community to optimize Apache Spark and Hadoop on Intel platform
- Improve Spark scalability and reliability
- Deliver better tools for management, benchmarking, tuning
  - e.g., HiBench, HiMeter
- Build Spark based solutions

Tachyon Nexus

- Team consists of Tachyon creators, top contributors
- Series A ($7.5 million) from Andreessen Horowitz
- Committed to Tachyon Open Source Project
- www.tachyonnexus.com

Outline

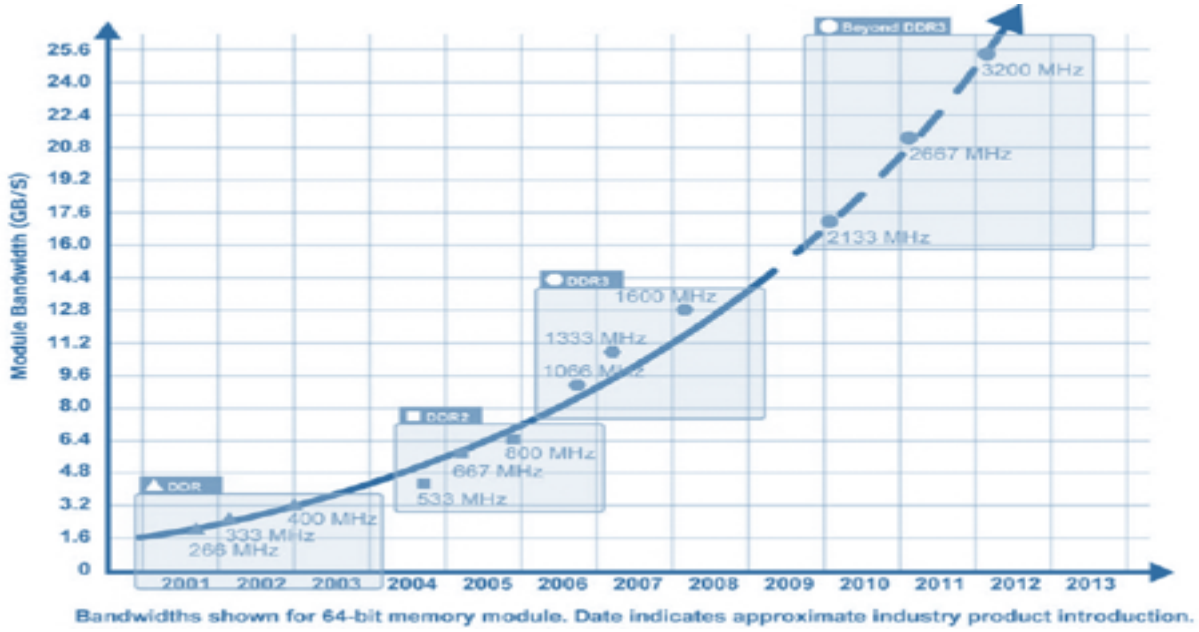
- Memory trend and challenges
- Introduction of Tachyon & NVM
- Performance testing and key learning
- Summary

Outline

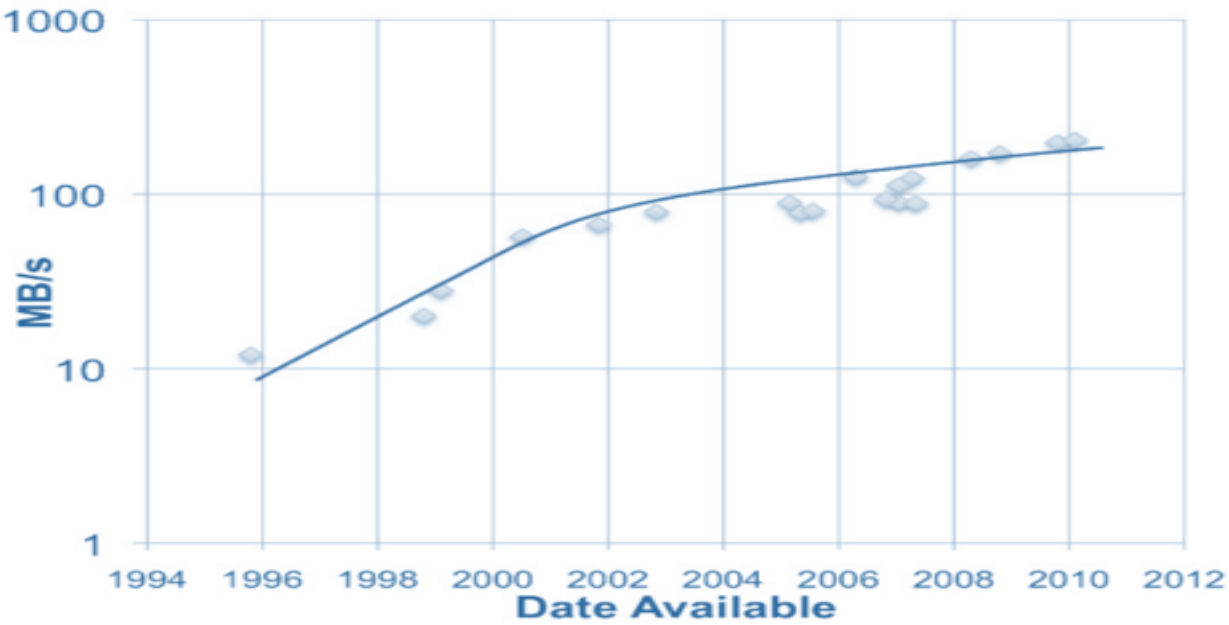
- Memory trend and challenges
- Introduction of Tachyon & NVM
- Performance testing and key learning
- Summary

Performance Trend: Memory is Fast

- RAM throughput increasing **exponentially**

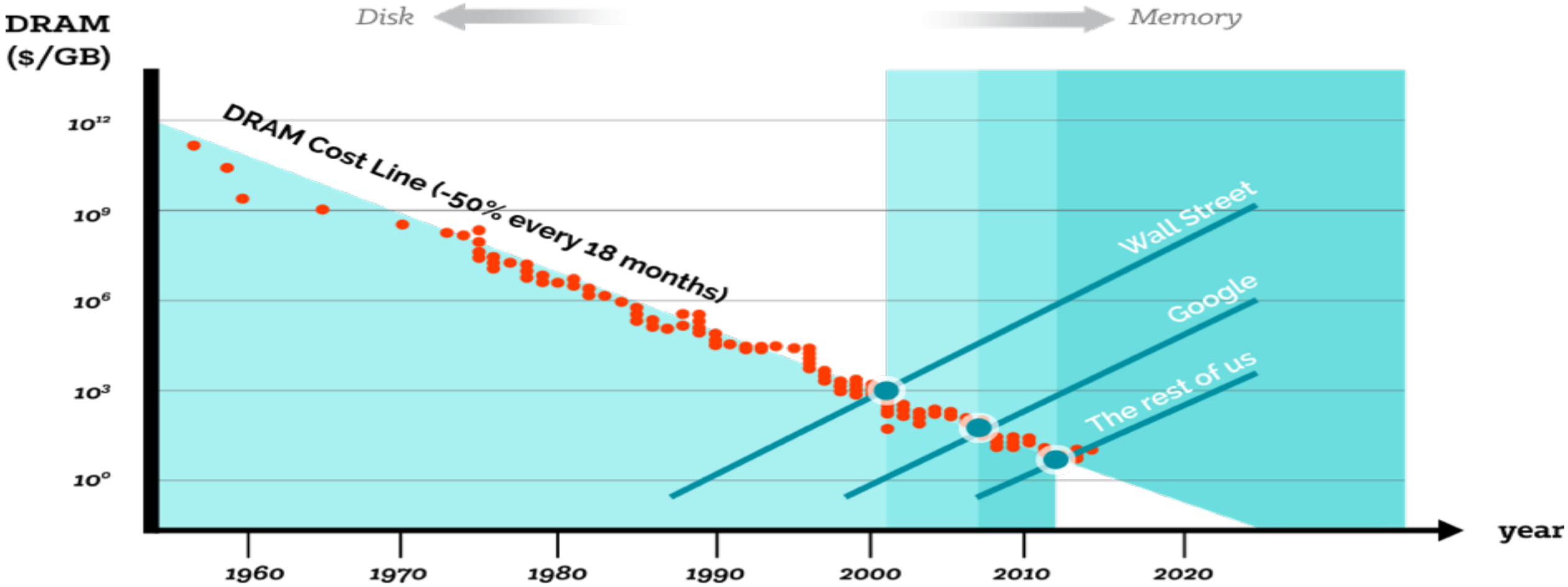


- Disk throughput increasing **slowly**



Memory-locality is important!

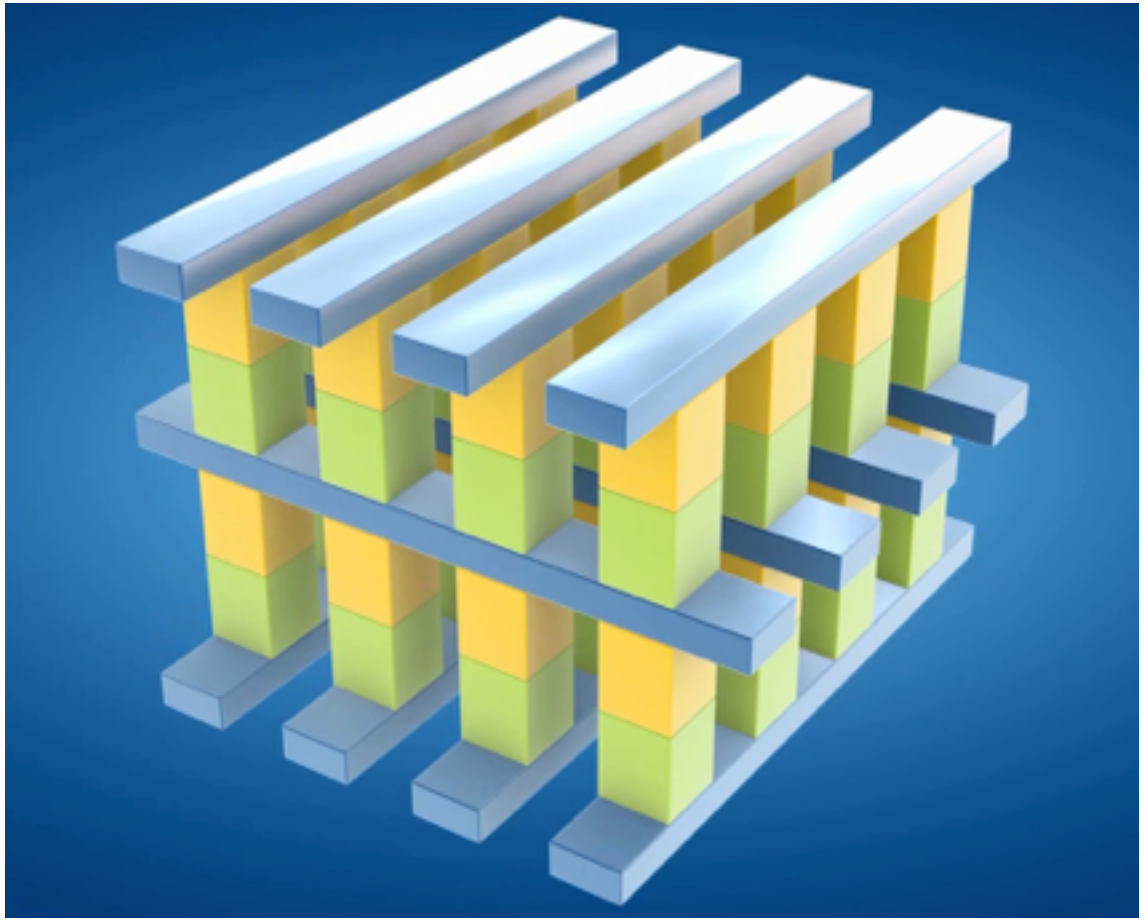
Price Trend: Memory is Cheaper



source: jcmit.com

$/ GB of Memory is lower, possible to handle huge size of data in memory

New Memory Technology: Intel's Crazy-Fast 3D XPoint Memory



- 1,000X faster than NAND
- 10X denser than DRAM
- less costly than DRAM
- Shipments may start in 2017

Realized By Many Frameworks:

cloudera
IMPALA

Challenges

- Effectively share in-memory data among distributed applications.
- GC overhead introduced by in memory caching
- Data set could be larger than memory capacity

Outline

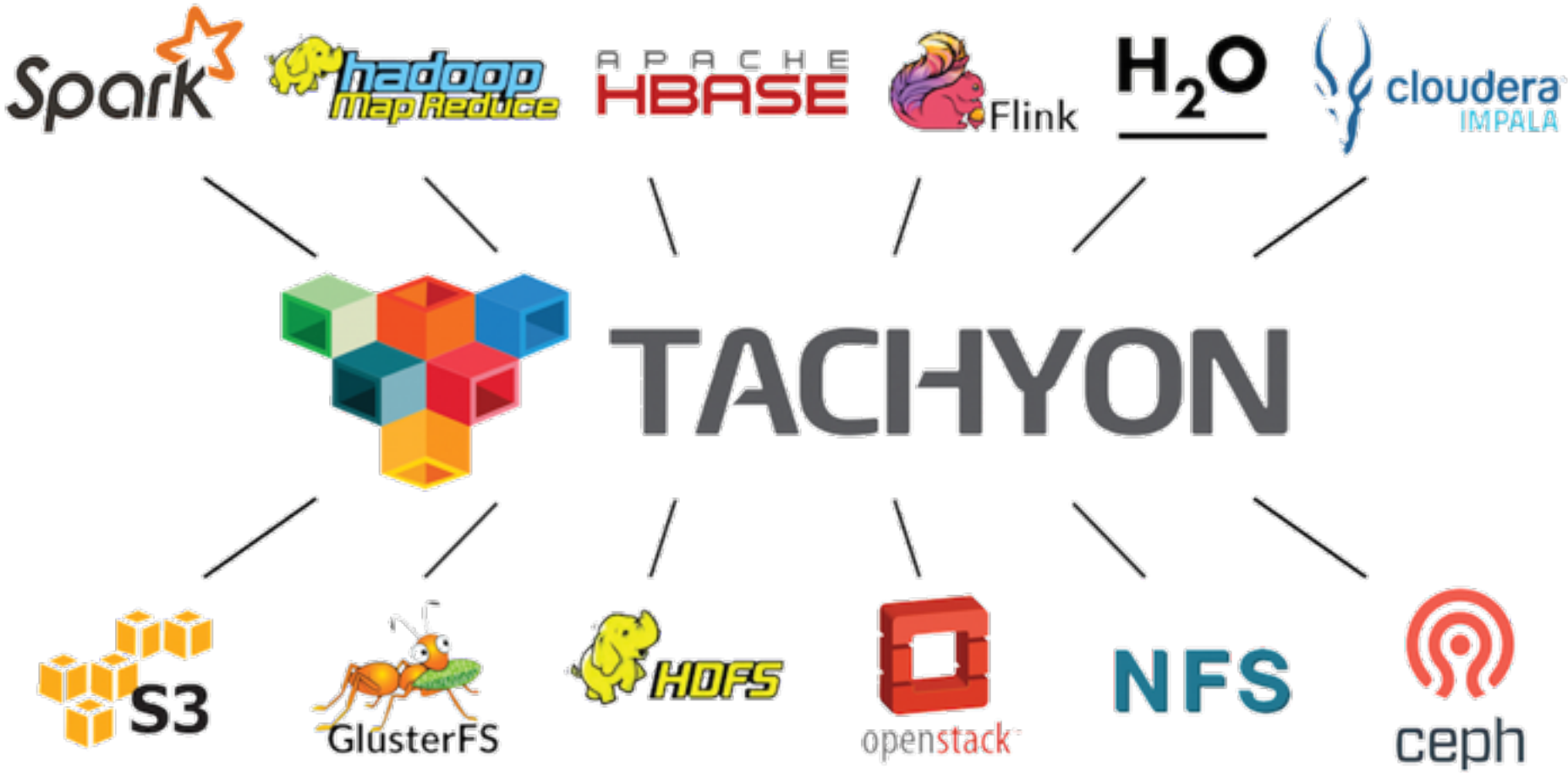
- Memory trend and challenges
- Introduction of Tachyon & NVM
- Performance testing and key learning
- Summary

What is Tachyon?

TACHYON

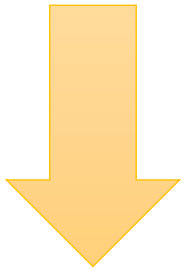
An Open Source Memory-centric Distributed Storage System

Tachyon Stack



How Easy to Use Tachyon in Spark

```
scala> val file = sc.textFile(“hdfs://foo”)
```

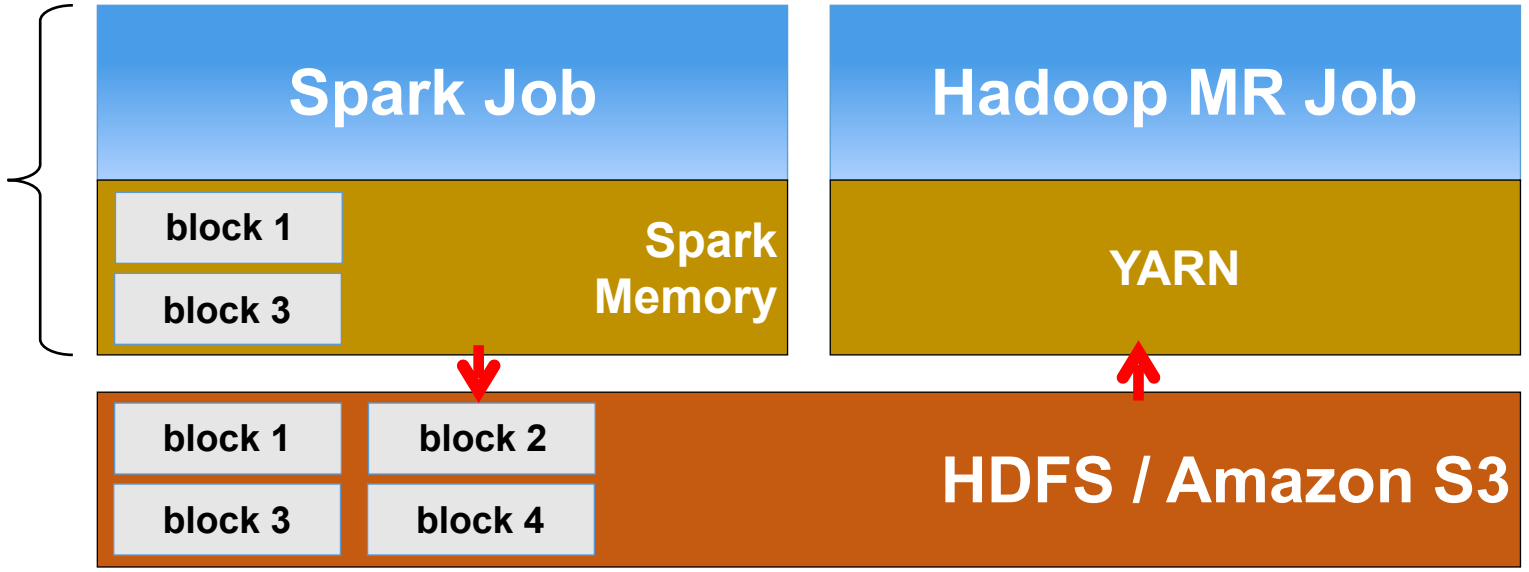


```
scala> val file = sc.textFile(“tachyon://foo”)
```

Issue 1

Data Sharing bottleneck in analytics pipeline: Slow writes to disk

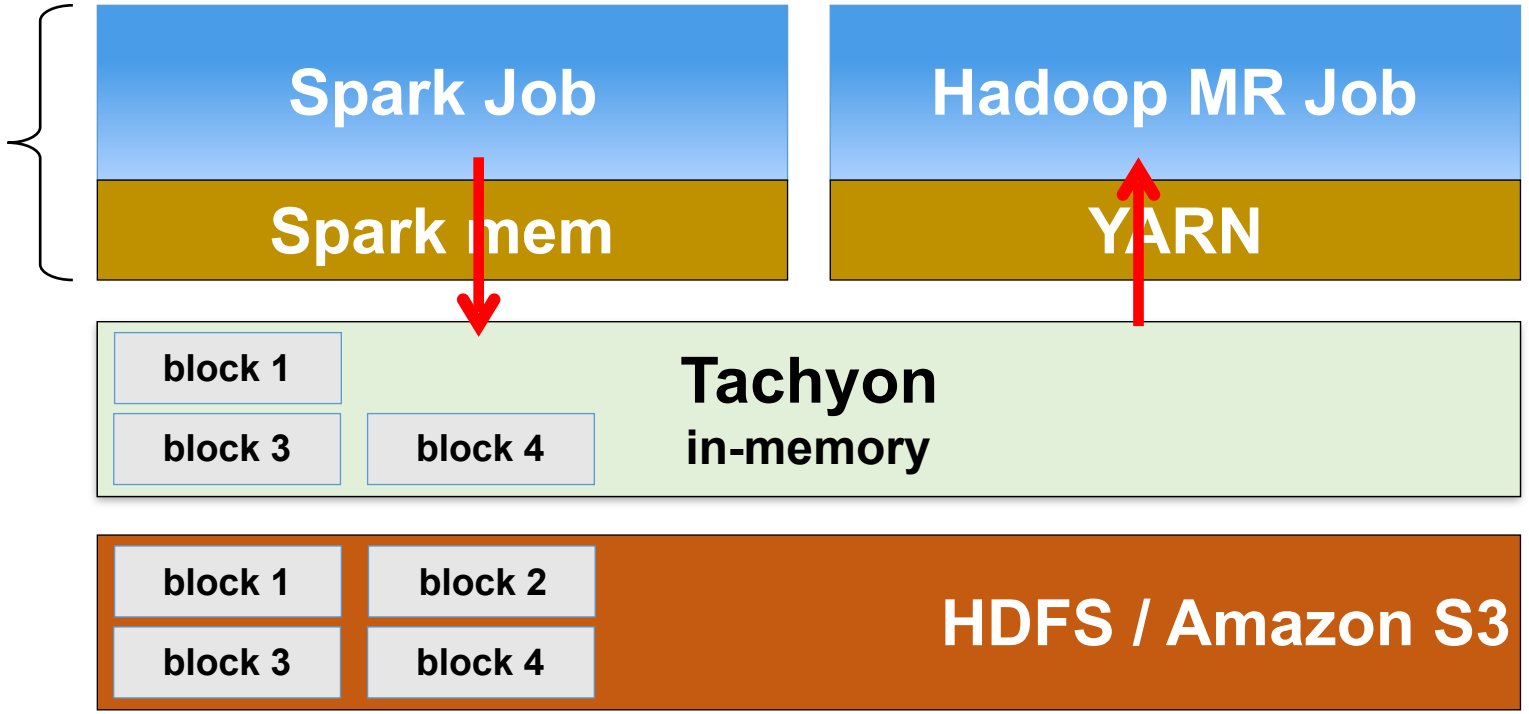
storage engine &
execution engine
same process



Issue 1 resolved with Tachyon

Memory-speed data sharing among different jobs and different frameworks

storage engine &
execution engine
same process

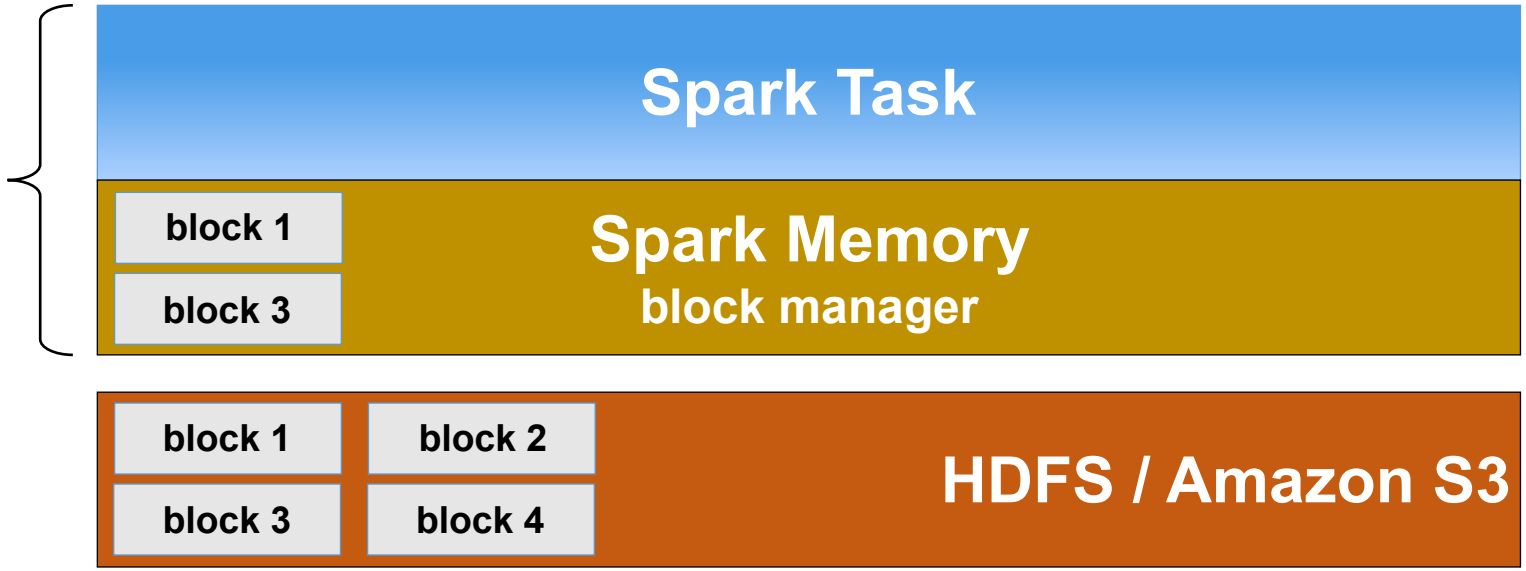


TACHYON
NEXUS

Issue 2

In-Memory data loss when computation crashes

storage engine &
execution engine
same process

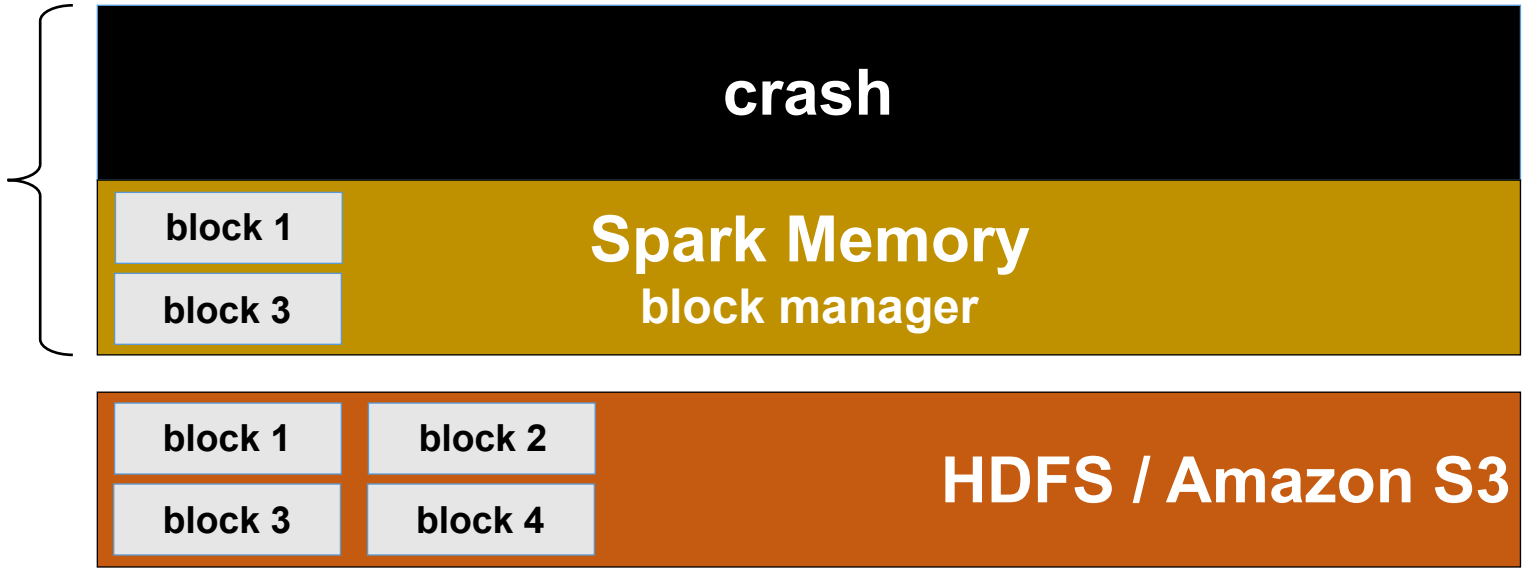


TACHYON
NEXUS

Issue 2

In-Memory data loss when computation crashes

storage engine &
execution engine
same process

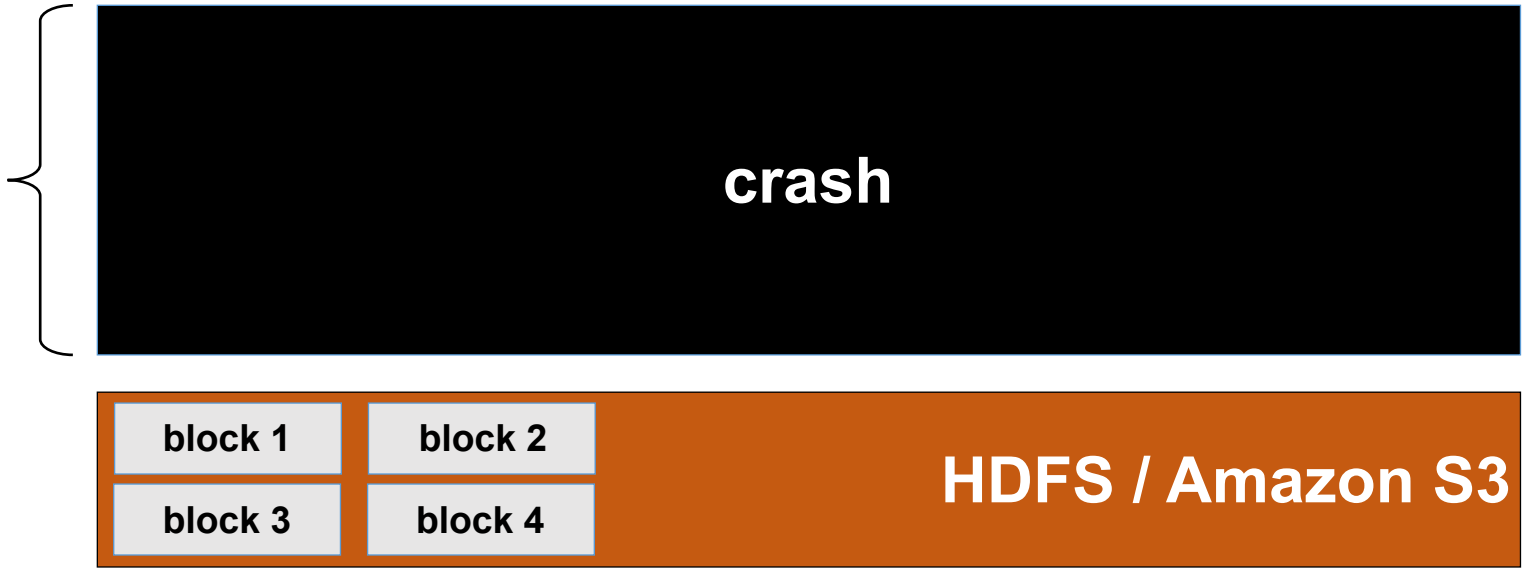


TACHYON
NEXUS

Issue 2

In-Memory data loss when computation crashes

storage engine &
execution engine
same process

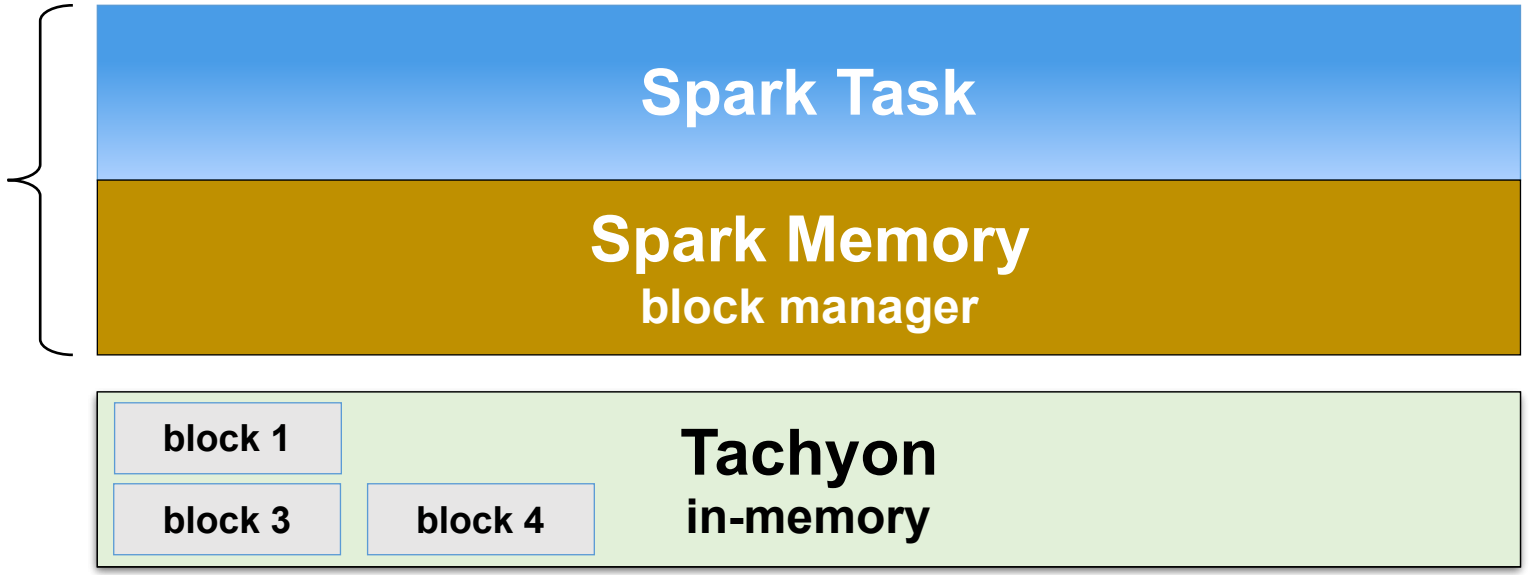


TACHYON NEXUS

Issue 2 resolved with Tachyon

Keep in-memory data safe, even when computation crashes

storage engine &
execution engine
same process

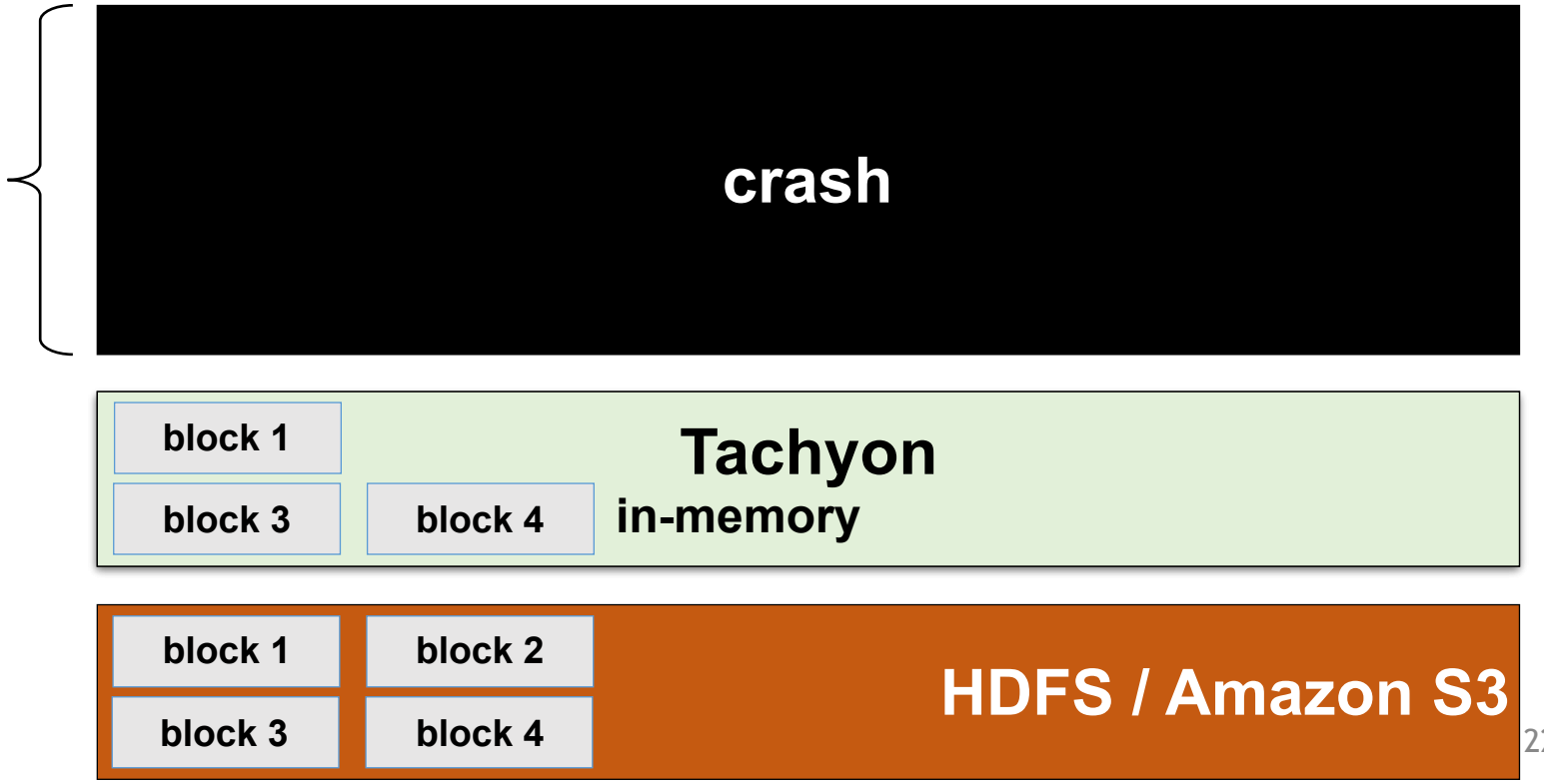


TACHYON
NEXUS

Issue 2 resolved with Tachyon

Keep in-memory data safe, even when computation crashes

storage engine &
execution engine
same process

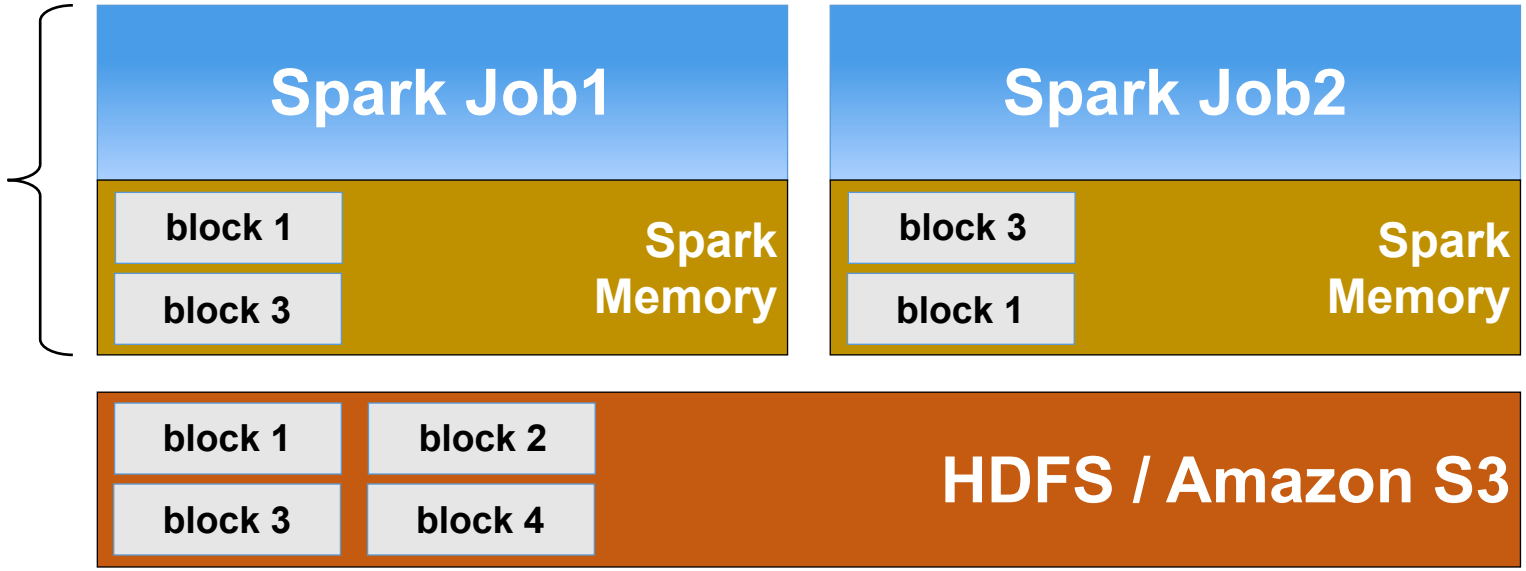


TACHYON
NEXUS

Issue 3

In-memory Data Duplication & Java Garbage Collection

storage engine &
execution engine
same process

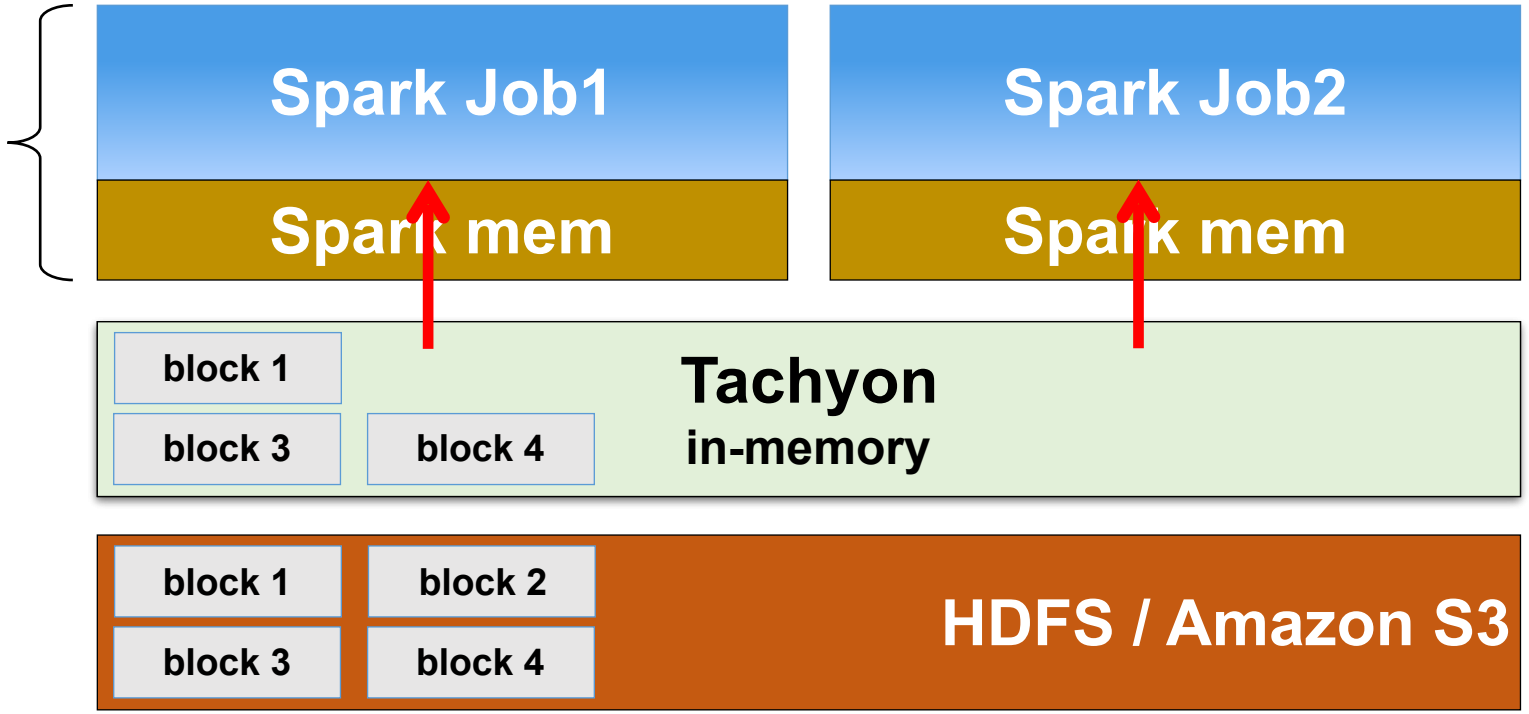


TACHYON NEXUS

Issue 3 resolved with Tachyon

No in-memory data duplication, much less GC

storage engine &
execution engine
same process



TACHYON
NEXUS

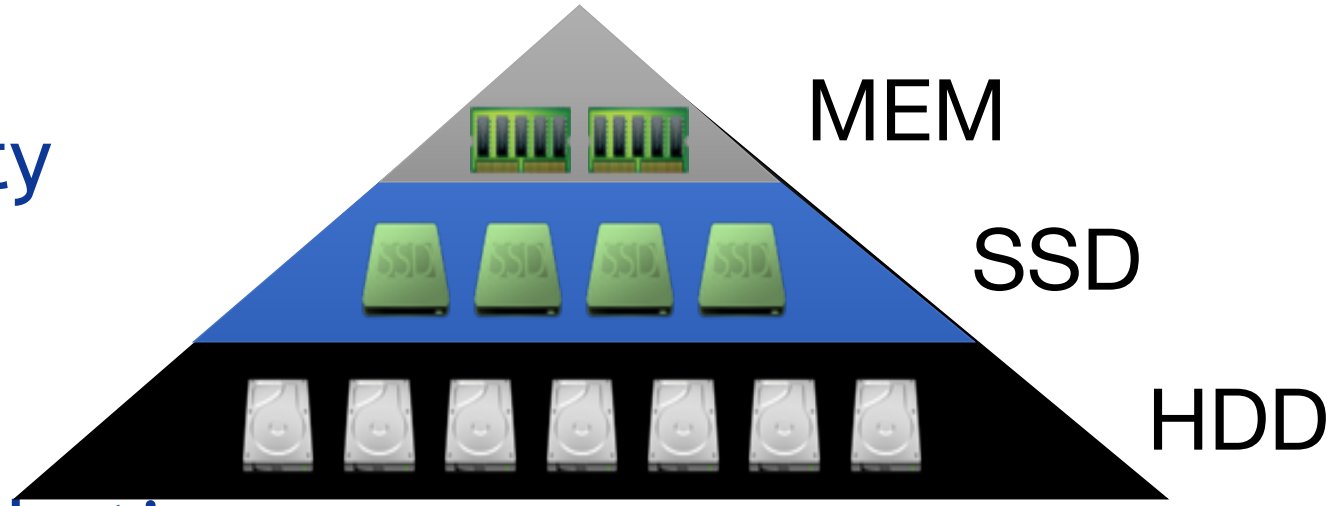
Challenges

- Effectively share in-memory data sharing among distributed applications.
  => HDFS compatible, Spark/MapReduce can access Tachyon in-memory data with no modification
- GC overhead introduced by in memory caching
  => No GC overhead as working set files are in memory but off-heap
- Data set could be larger than memory capacity
  => Tiered storage

Tiered block storage: extend Tachyon space with external storage (e.g., SSD HDD etc.)

- It is available since Tachyon 0.6 Release
  - The JIRA for tiered storage: TACHYON-33

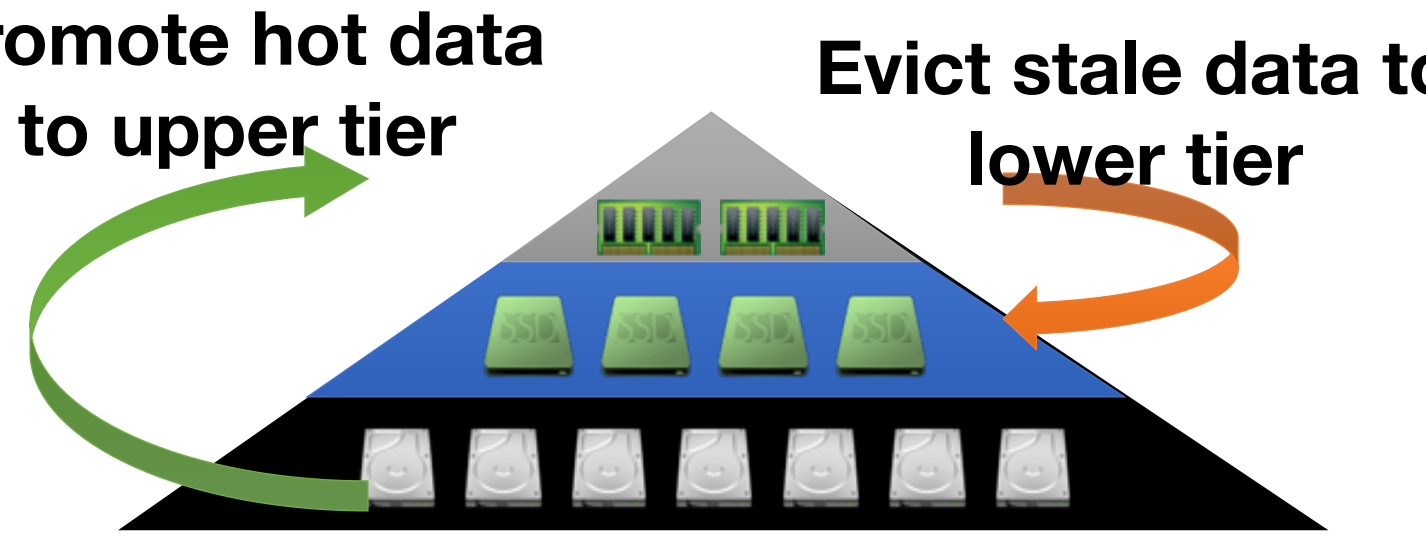
- Different tiers have different speed and priority
  - "hot" data in upper tier
  - "warm" data in lower tier



- Multiple directories in single tier

Data migration among tiers

- Data can be evicted to lower layer if it is less “hot”
- Data can be promote to upper layer if it is “hot” again.
- Pluggable Eviction Policy
  - LRU, LRFU pre-defined



Non-Volatile Memory (NVM)

- NVM is the next era of computer memory
  - Including 3D NAND, solid-state drives, and Intel 3D XPoint™ technology.
- Advantages of NVM
  - Fast, comes to memory performance, low latency
  - Non-Volatile, Capable of retrieving stored data even after a power outage.
  - Inexpensive

When Tachyon Meets NVM

- Data caching & sharing at memory speed
  - Local cache for remote data
  - Data share in job chain / among different applications
- Eliminate GC overhead
  - Storing data off heap on NVM
- Efficient cache management
  - Storage hierarchy knowledge for different storage media, MEM SSD HDD etc
  - Quota management for different storage

Outline

- Memory trend and challenges
- Introduction of Tachyon & NVM
- Performance testing and key learning
- Summary

PCI-E SSD(P3700) SPEC

Specification	Unit	Intel SSD DC P3700 Series			
		400GB	800GB	1.6TB	2TB
Sequential Read (up to)[1]	MB/s	2,700	2,800	2,800	2,800
Sequential Write (up to)[1]	MB/s	1,080	1,900	1,900	2,000

- Due to availability of NVM hardware, use high speed SSD to simulate the usage of NVM
  - The NVM can be just used as block device like SSD, with lower latency and higher bandwidth
  - The performance of NVM will be better than PCI-E SSD, with similar behavior

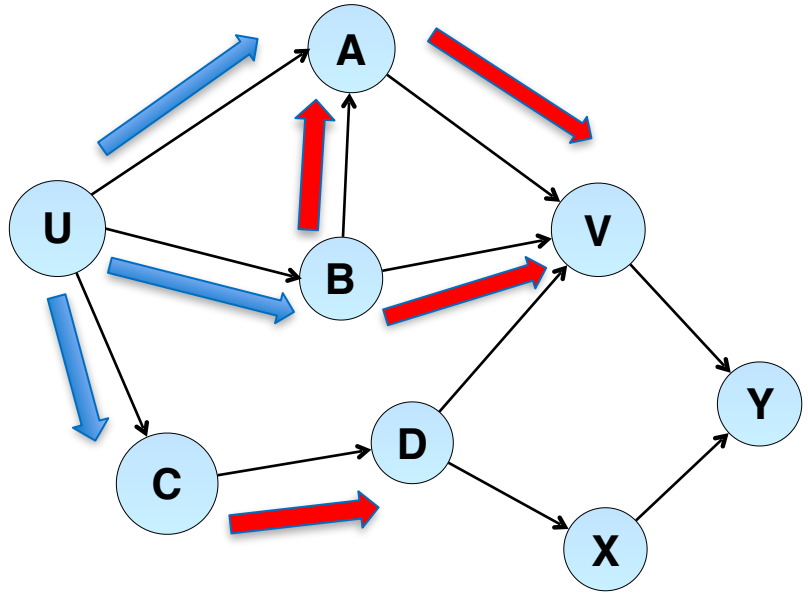
Test Bed

- One Tachyon master node
- Four Tachyon worker nodes

Worker node Configuration	
CPU	IVB E5-2680 @ 2.8G 40 logical cores
Memory	192GB DDR3 @ 1066 MHZ
Disk	P3700 SSD * 1 + HDD * 11
OS Distribution	Redhat 6.2 (kernel 2.6.32-220.el6.x86_64)
JDK version	JDK1.8.0_60 (64 bit server)
Spark version	1.4.1 release
Tachyon version	0.8.1 release
Hadoop version	2.5.0

Experiment 1: Big Graph Computation

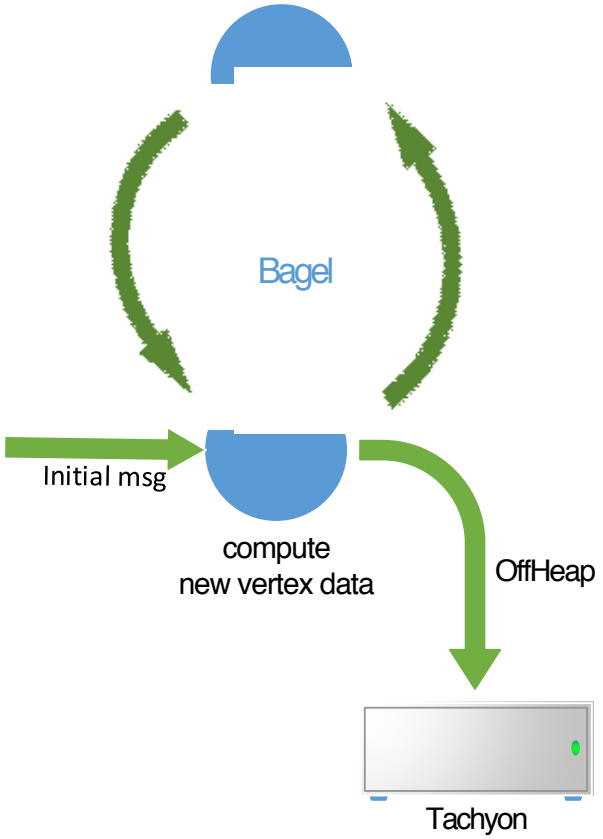
- Background: Calculate similarity between videos
- A case of Graph Analysis
  - Friends-of-friend
  - Iteratively compute association between nodes



- Challenge
  - Generating huge Intermediate data after each iter

Architecture: Bagel over Tachyon

- Bagel is a graph processing framework.
  - Spark implementation of Google Pregel
  - jobs run as a sequence of iterations.
  - In each iteration, each vertex runs a user-specified function
    - update state associated with the vertex and
    - send messages to other vertices for use in the next iteration.
  - The message generated and new vertex data will be cached after each iteration
- Intermediate data can be cache in Tachyon
  - Tachyon can make full use of high speed device for data caching.

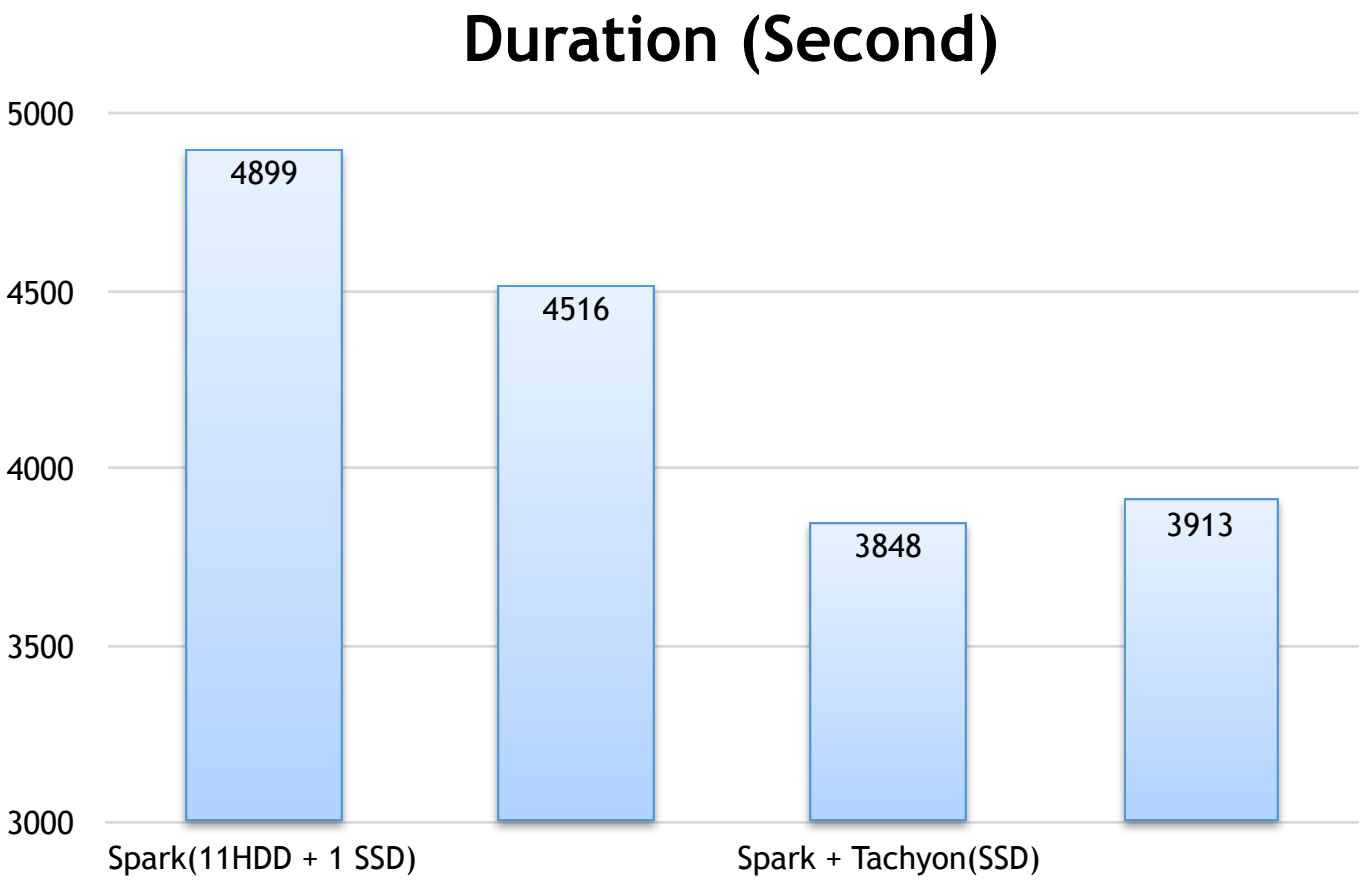


Configuration

- Input data size: 22G
- Caching data size: 1.5TB
- Four configuration to test

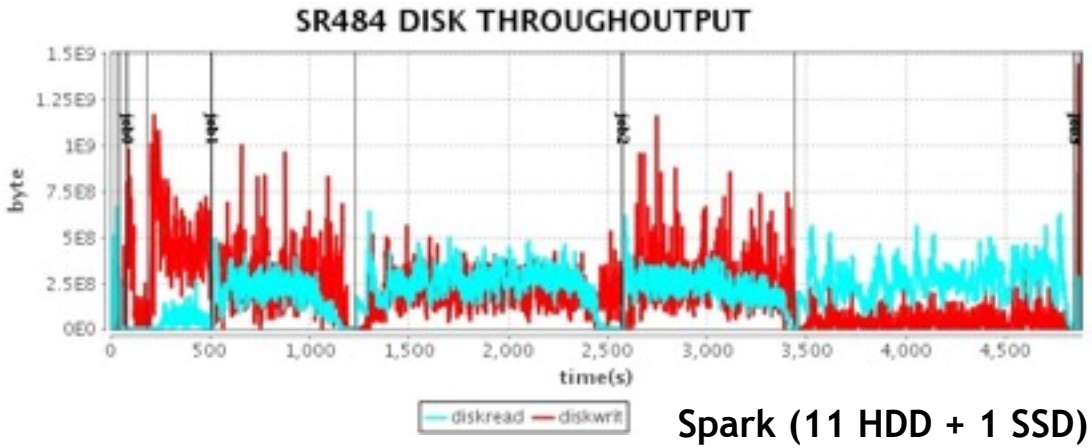
Framework	Storage Configuration
Spark	Spark: 11HDD + 1 SSD
Spark	Spark: 11HDD + 4 SSD
Spark + Tachyon	Spark: 11 HDD Tachyon (1 tier): 1 SSD (500GB)
Spark + Tachyon	Spark: 11 HDD Tachyon (2 tiers): 1 SSD (250GB) + 11 HDD (100GB per disk) Eviction strategy: LRU

Test Result

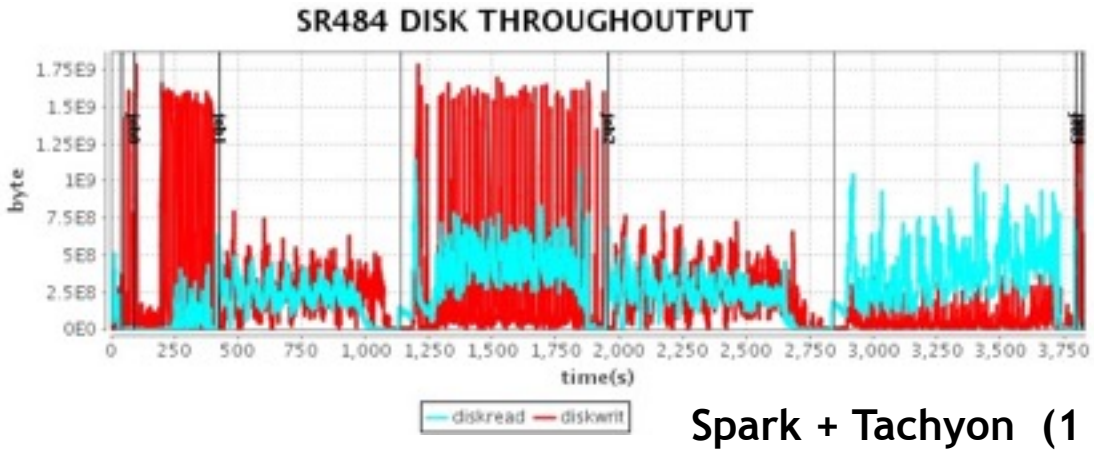


- Tachyon with SSD & SSD + HDD outperforms original Spark, gets about 20% performance gain
- In Spark + Tachyon(2 tiers), 2/3 of caching data is putted on SSD and 1/3 on Disk
- Spark + Tachyon(1 tier) and Spark + Tachyon(2 tiers) have very similar performance

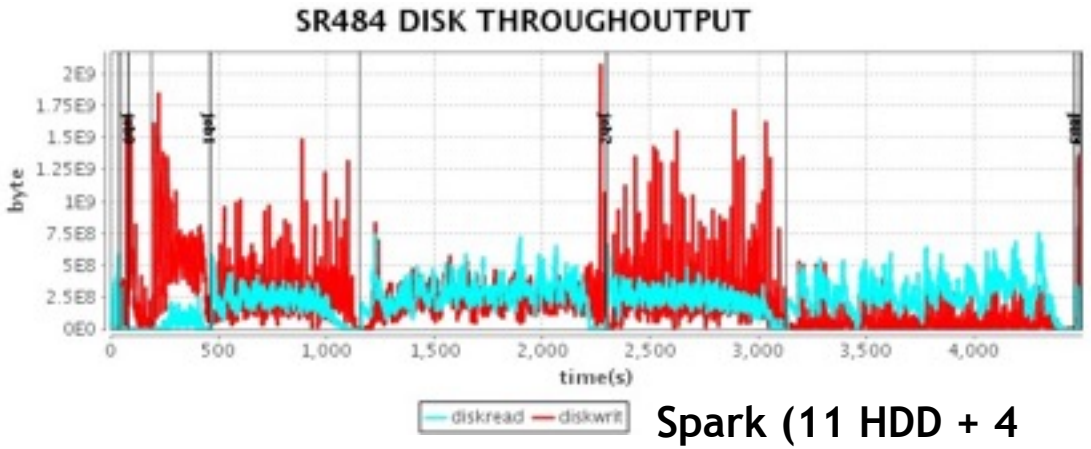
Performance Analysis – Disk utilization



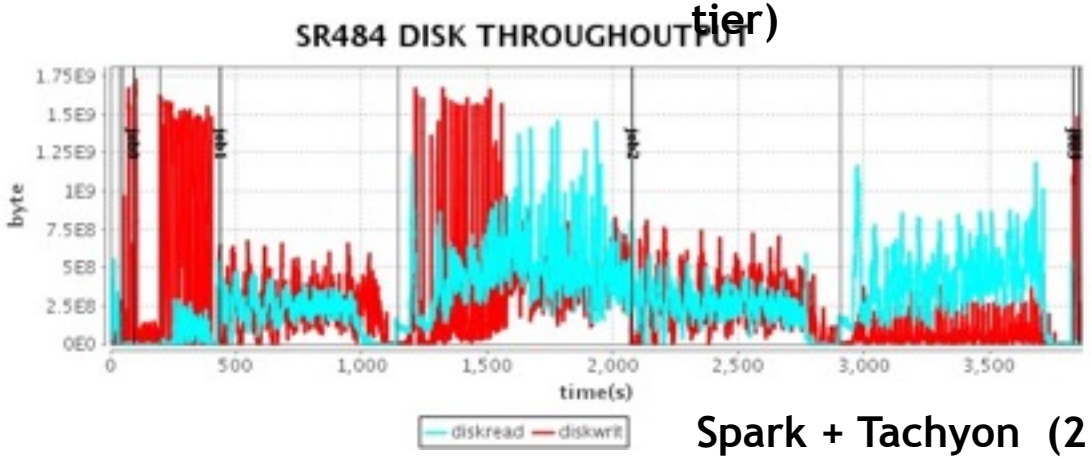
Spark (11 HDD + 1 SSD)



Spark + Tachyon (1 tier)



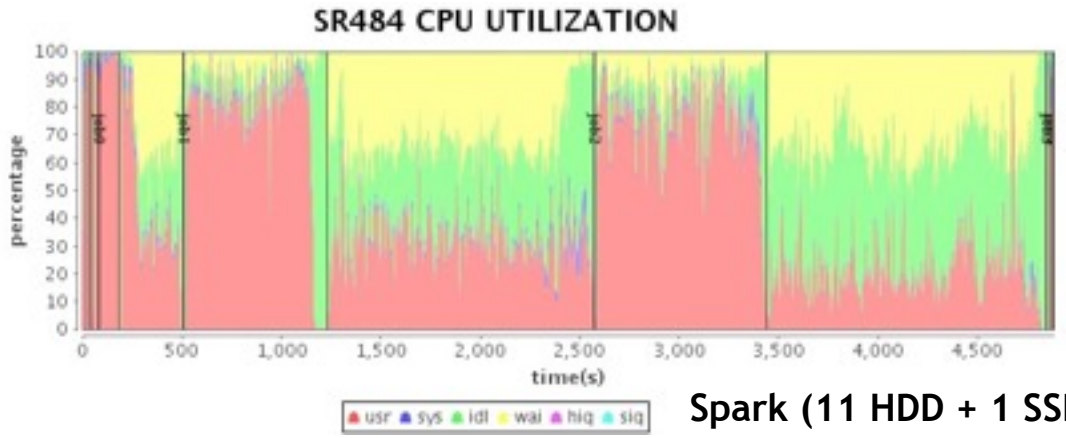
Spark (11 HDD + 4 SSD)



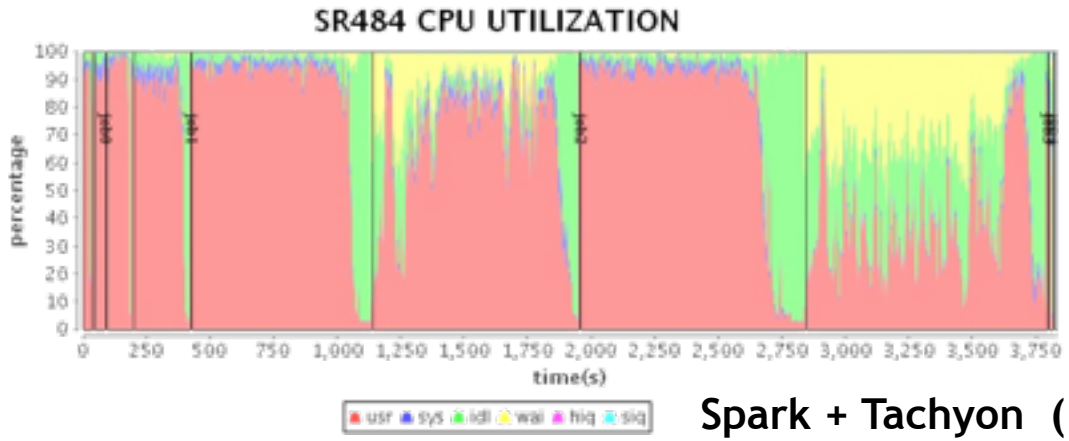
Spark + Tachyon (2 tiers)

Tachyon helps achieve higher disk utilization

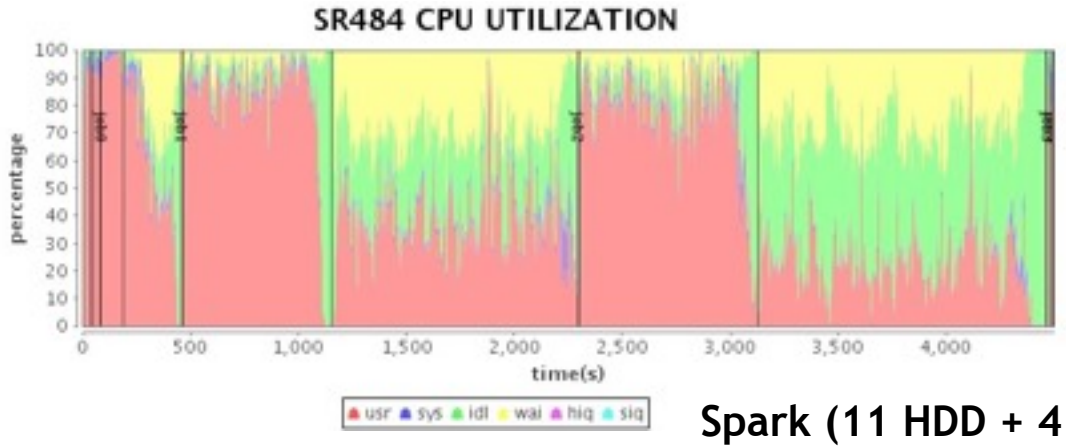
Performance Analysis – CPU utilization



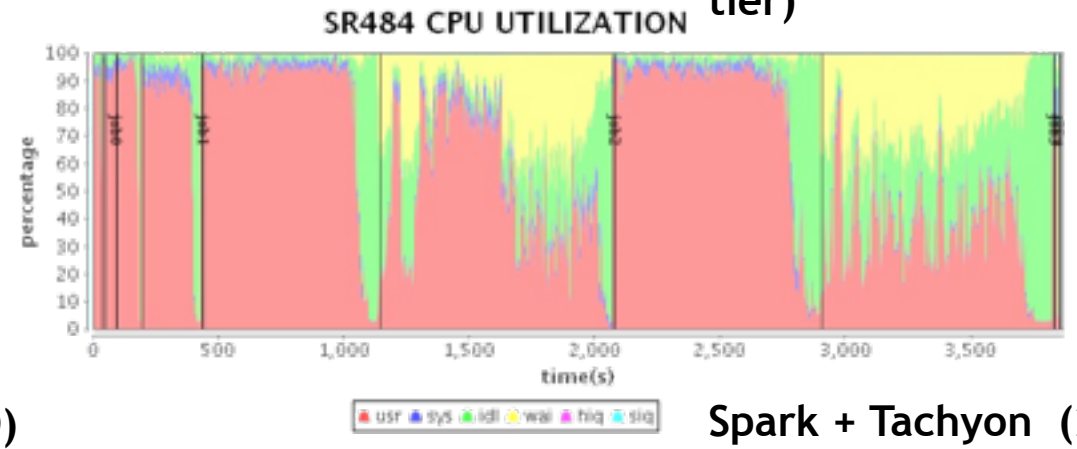
Spark (11 HDD + 1 SSD)



Spark + Tachyon (1 tier)



Spark (11 HDD + 4 SSD)



Spark + Tachyon (2 tiers)

Tachyon helps achieve higher CPU utilization due to less time in IO

Experiment 1: Summary

- Tachyon significantly speeds up Spark
  - Spark takes all kinds of storage as the same storage media, it randomly allocates space and access to all local directories configured.

- With tiered storage and smarter cache management, Tachyon achieves comparable performance while requiring less fast memory resource
  - Tachyon tries to use the fast device as much as possible, and has efficient data cache management for “hot” and “warm” data.

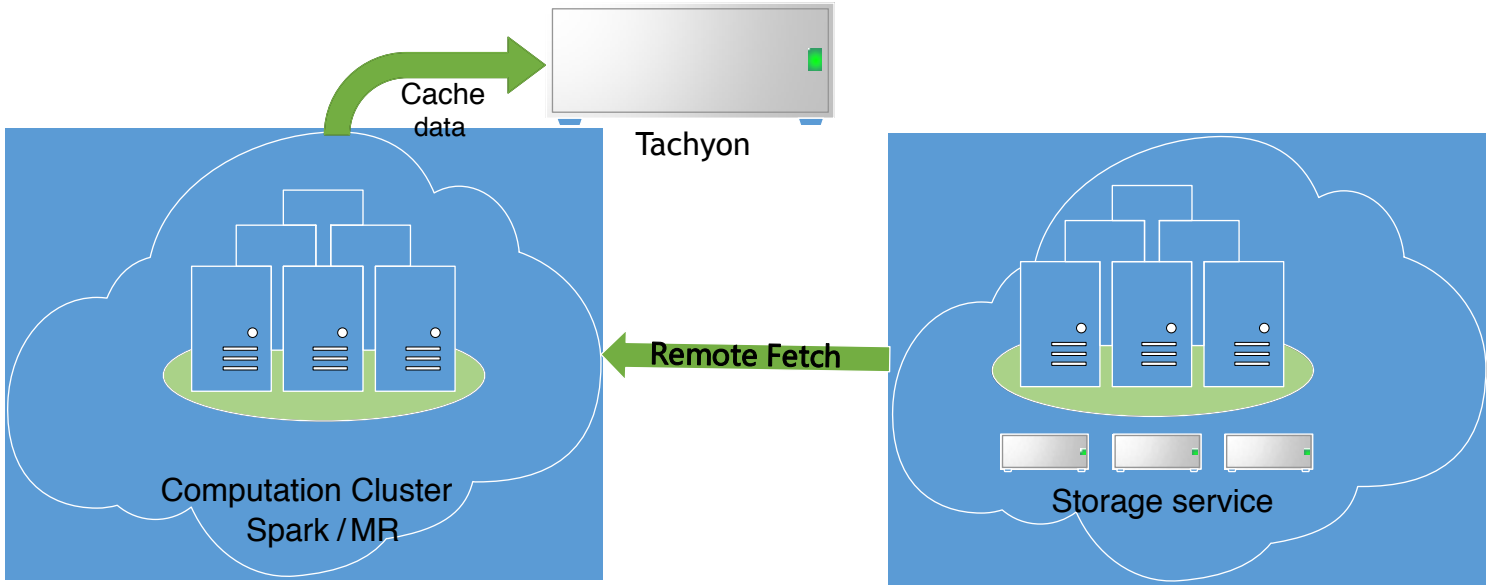
Experiment 2: Recommendation System

- Background: data analysis of user event logs
  - Job1: Top-K online videos
    - IO Bound
  - Job2: Visits of advertisement by unique users in one day/ week
    - CPU Bound

- Challenges:
  - storage and computation cluster are separate.
    - Data must be transferred from remote data cluster during computation
  - Same Input data is used by different applications
    - causing redundant network I/O.

Architecture: Tachyon as local cache

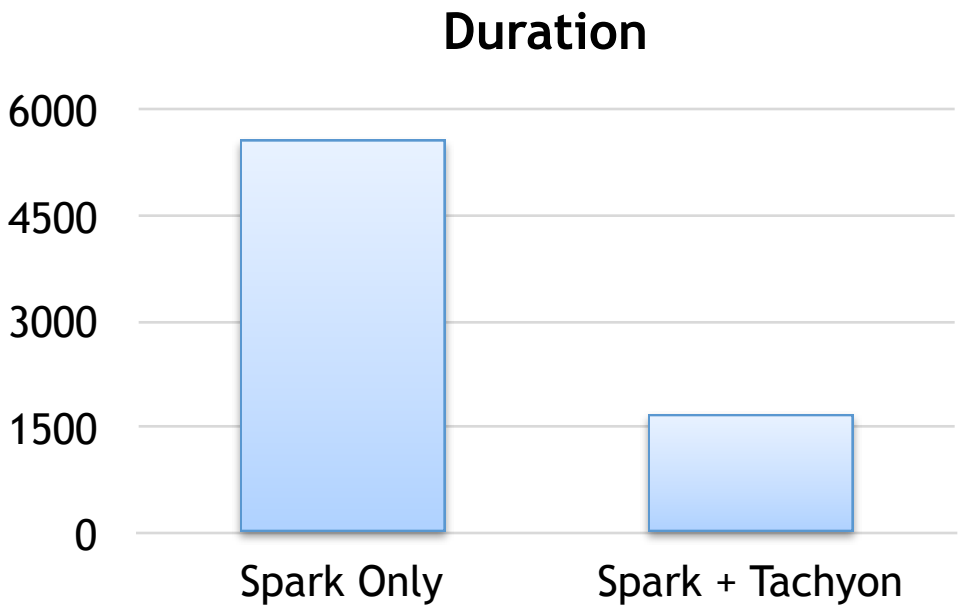
- Deploy Tachyon in compute cluster to avoid the duplicated network I/O



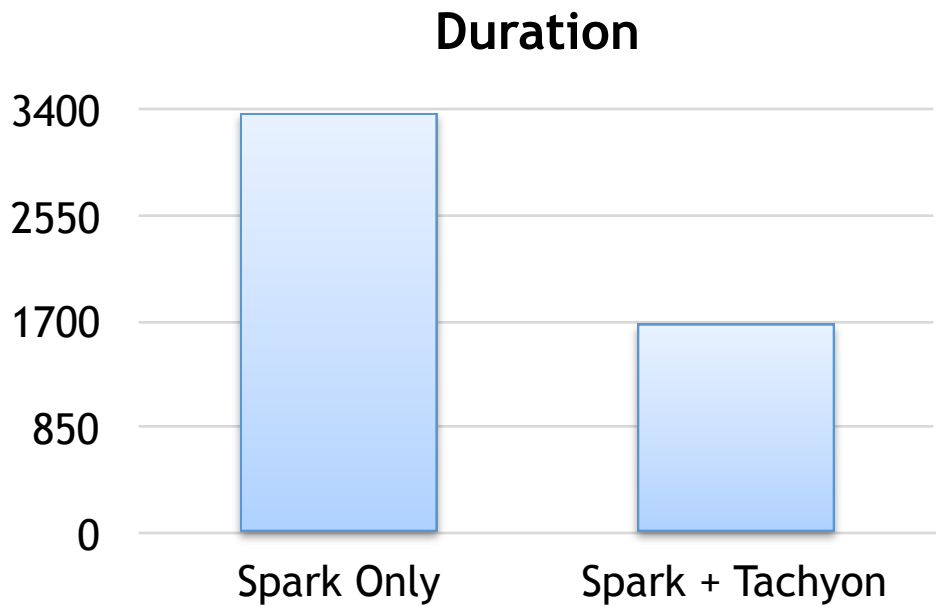
Configuration

- Input data size: 1.2TB
- Tachyon
  - 1 tier with 500GB SSD on each worker
- Network
  - 1 Gb connection between computation cluster and data service cluster
  - 10 Gb connection inside computation cluster

Test Result

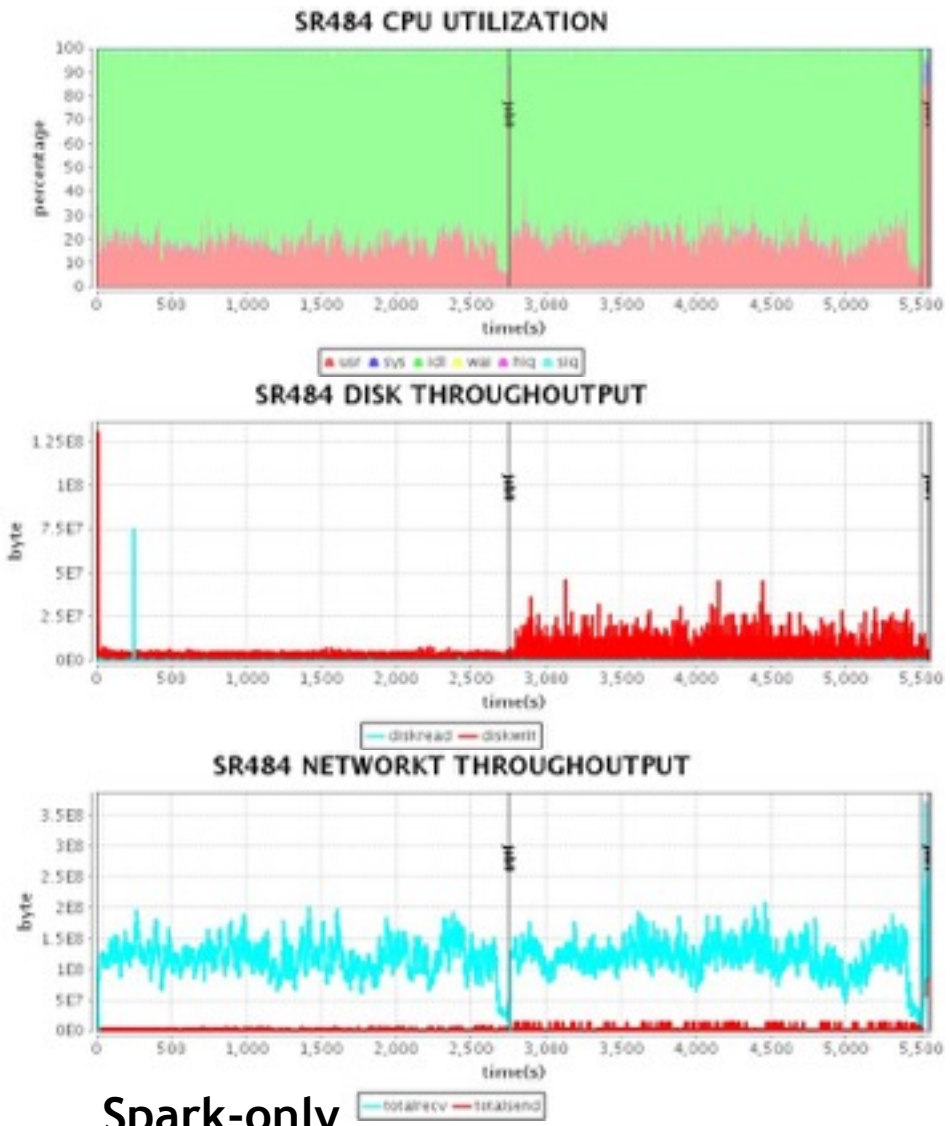


Job1 (IO Bound)

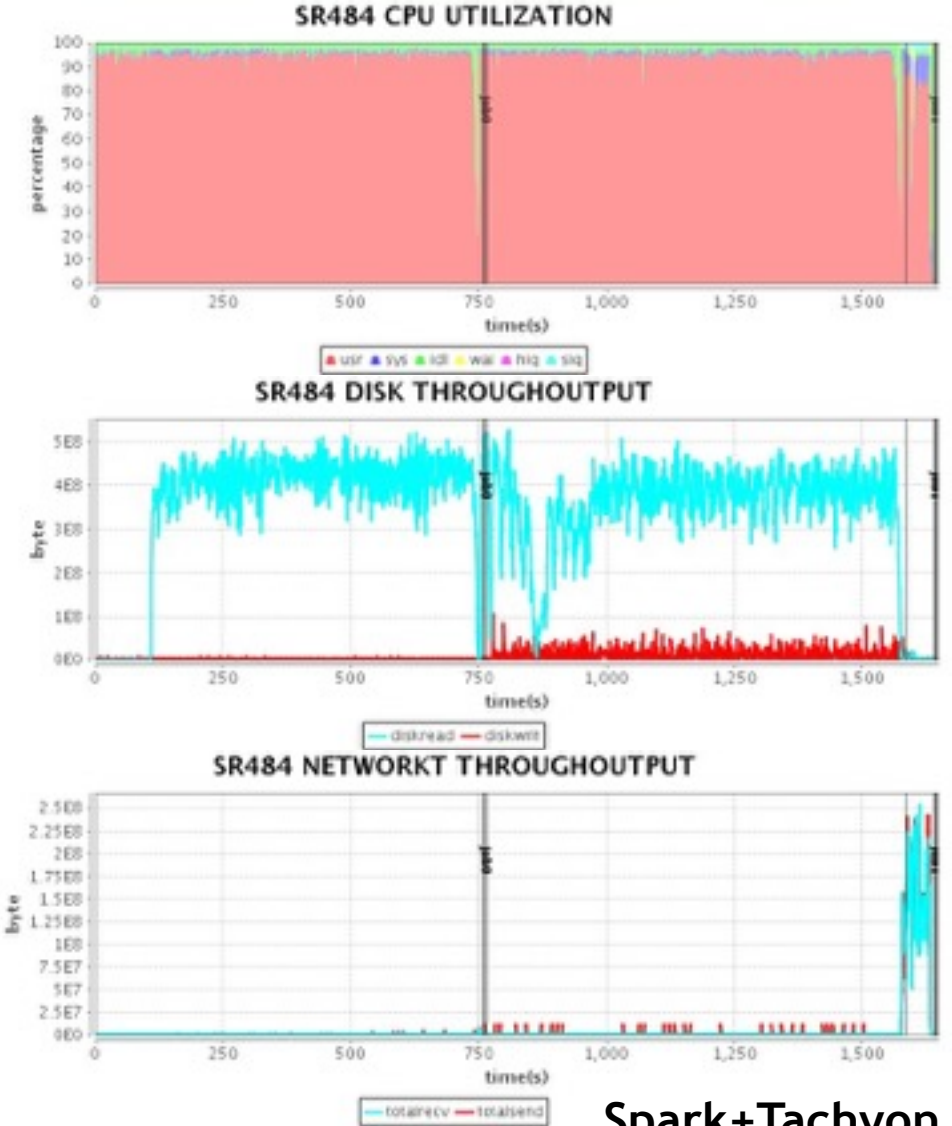


Job2 (CPU Bound)

Performance Analysis – Job1

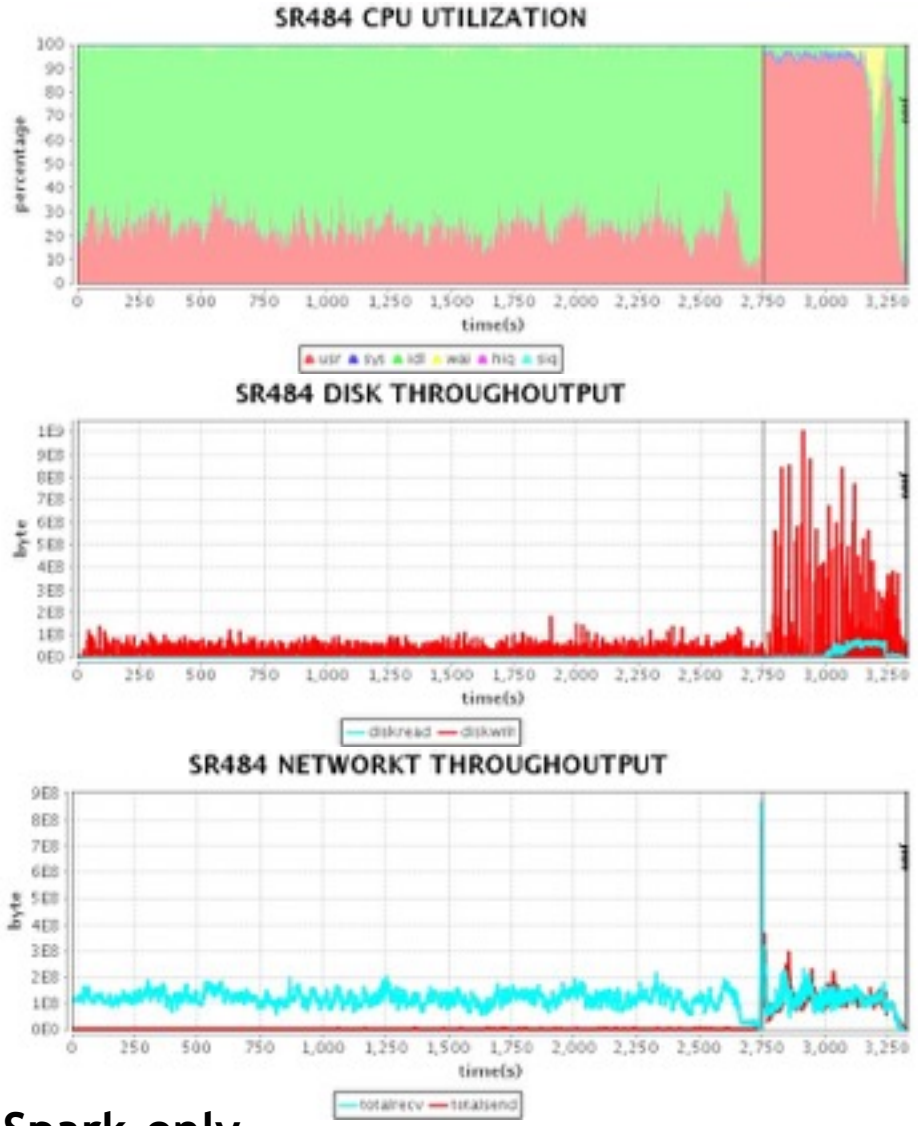


Spark-only

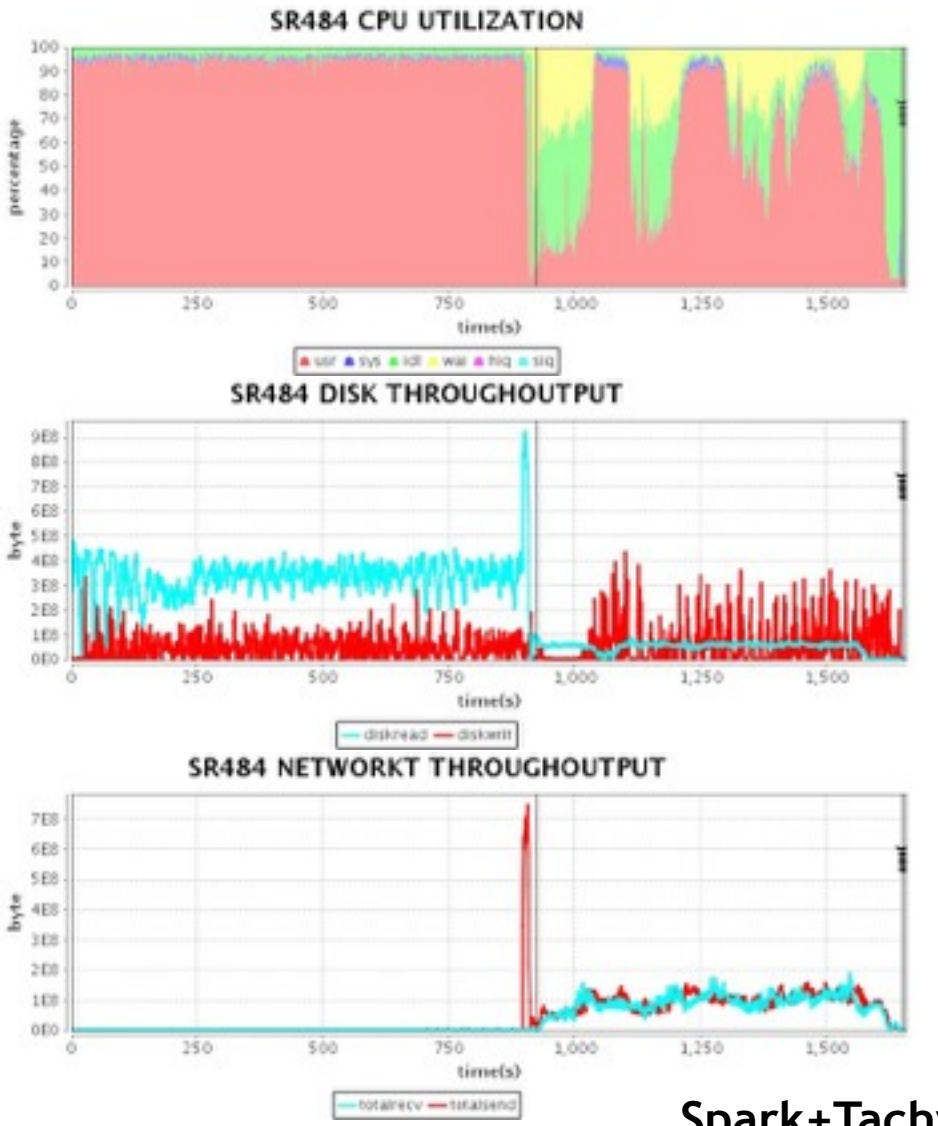


Spark+Tachyon

Performance Analysis – Job2



Spark-only



Spark+Tachyon

Experiment 2: Summary

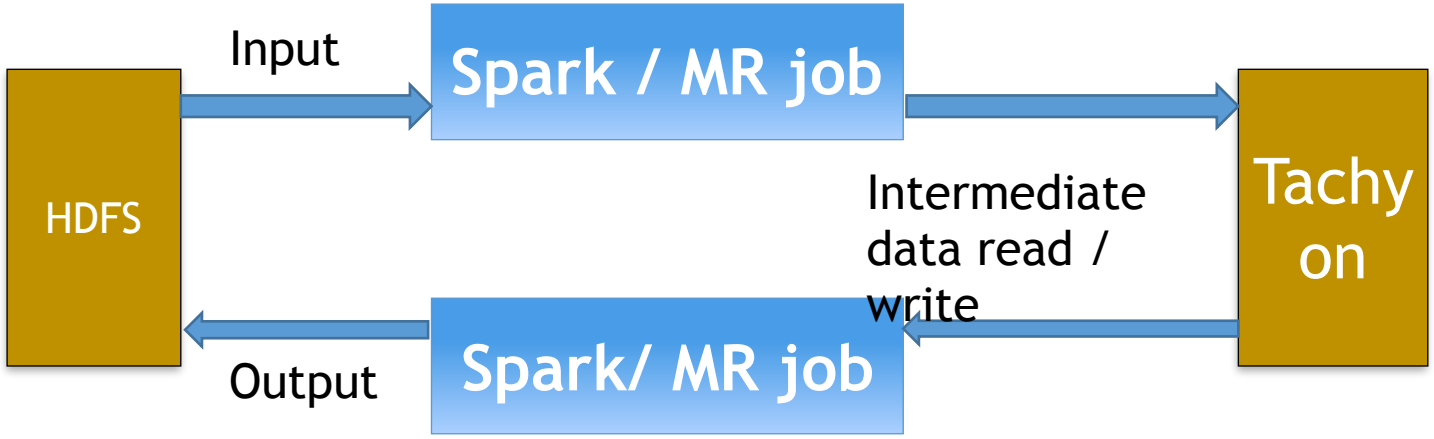
- Using Tachyon as local cache accelerates the application
  - For the application is I/O bound, Tachyon greatly improves the performance(3X)
  - For the application is CPU bound, Tachyon still brings much performance gain(2X)

Experiment 3: Data Sharing within a Pipeline

- Background:
  - Intermediate data is shared in a job chain , the output of one job is the input of another job
- Challenge:
  - Without Tachyon, output of the previous job needs to be written into HDFS, and read by another job

Architecture

•With Tachyon, the output can be cached in Tachyon caching space, and makes the jobs share data without heavy network & HDD I/O



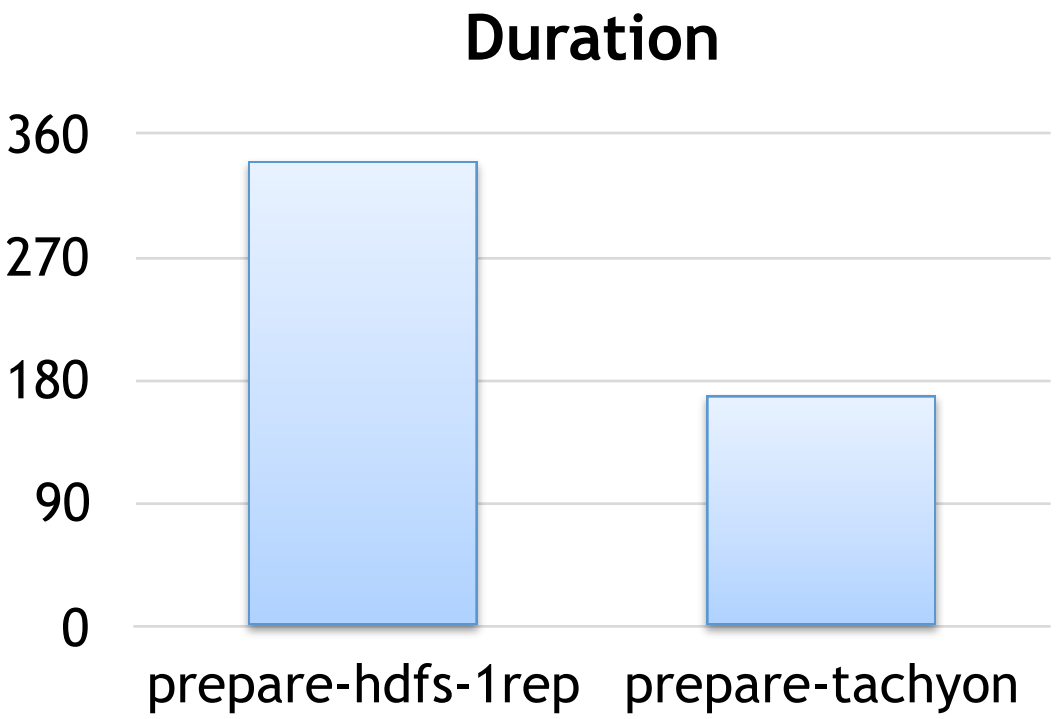
Data share in job chain – Terasort

- Two jobs in Terasort：
  - Data-preparation
  - Data-processing
- Data sharing between two jobs:
  - Written into HDFS, with 1 replication
  - Written into Tachyon

Configuration

- Input data size: 1TB
- Tachyon caching space
  - 1 layer with 500G SSD on each worker
- Two rounds of testing for the second phase:
  - Data processing-round1:
    - Write result into HDFS with 3 replications
  - Data processing-round2:
    - Write result into HDFS with 1 replication / tachyon

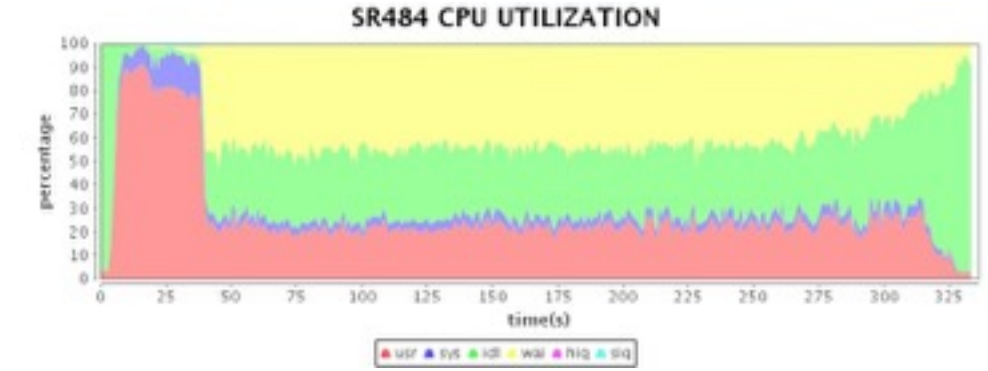
Test Result

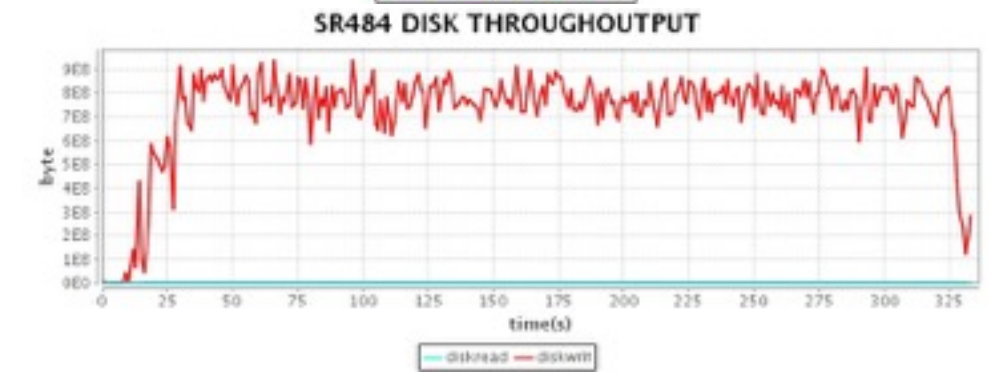


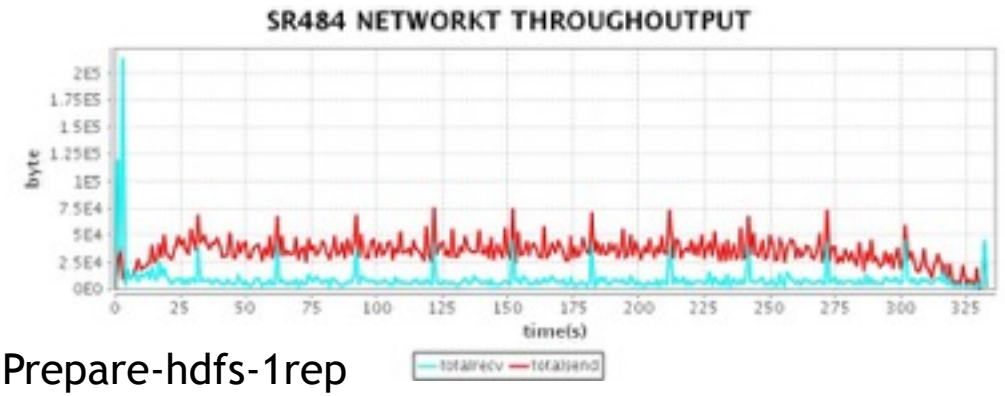
Data preparation

Data processing

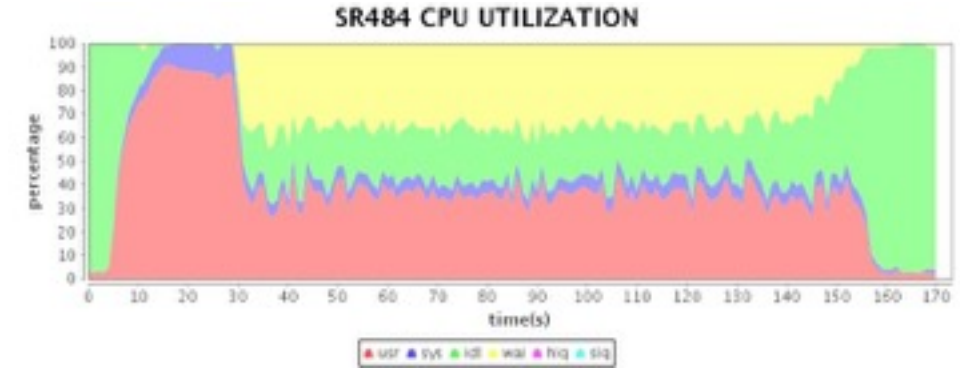
Performance Analysis – Data preparation

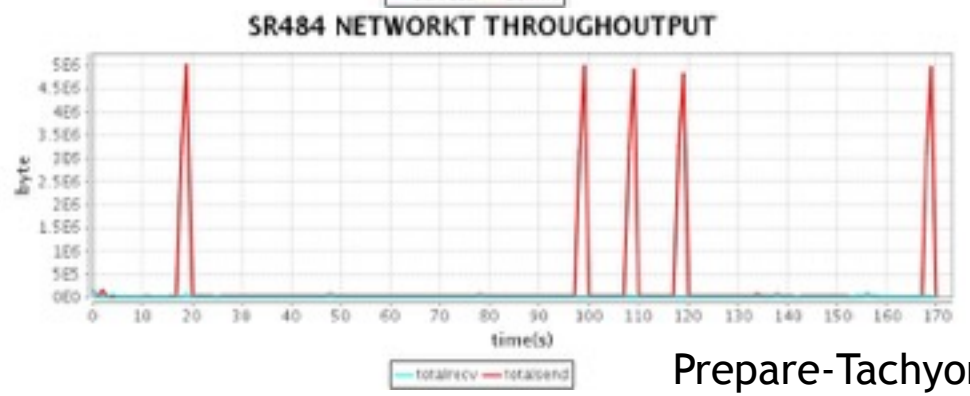






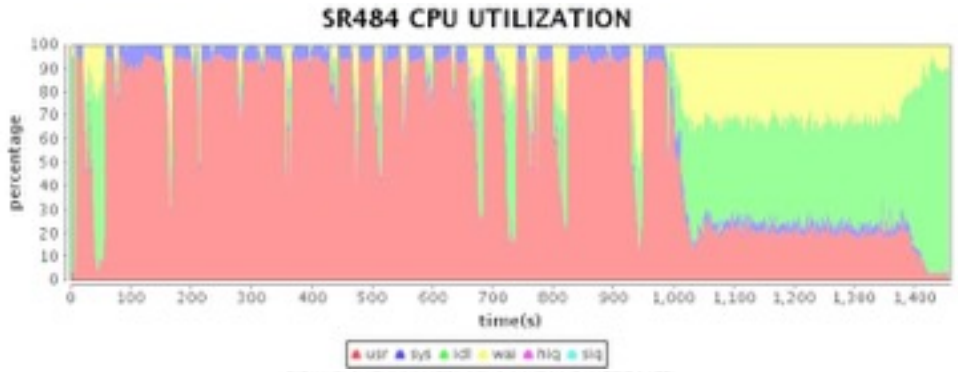
Prepare-hdfs-1rep

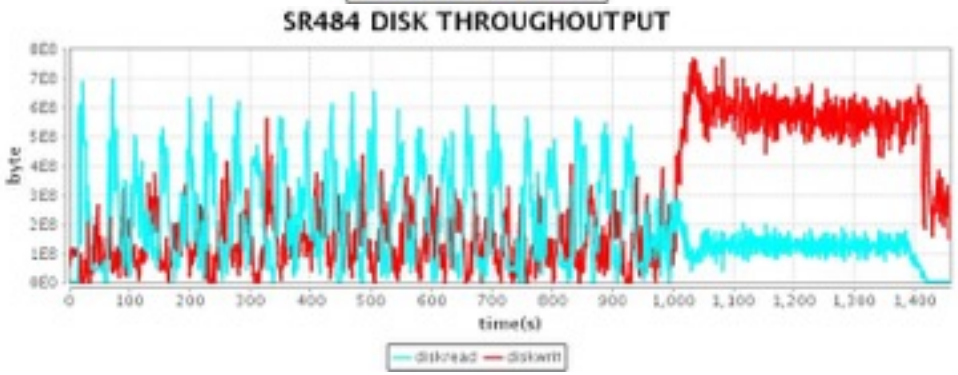


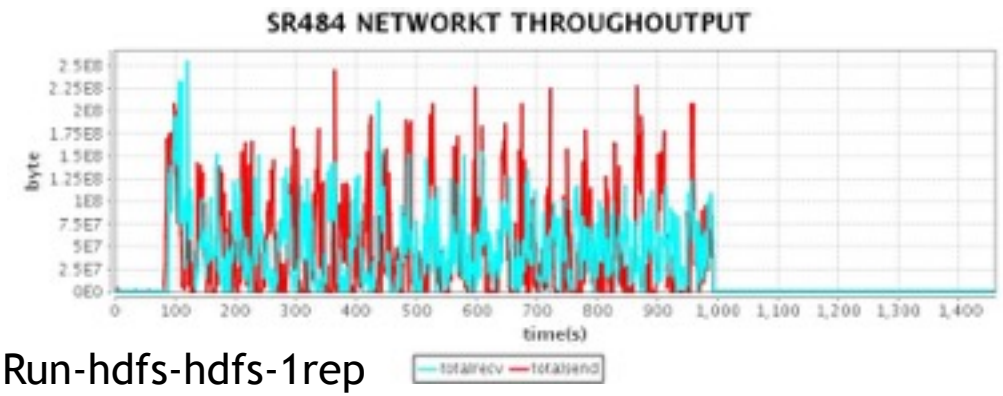


Prepare-Tachyon

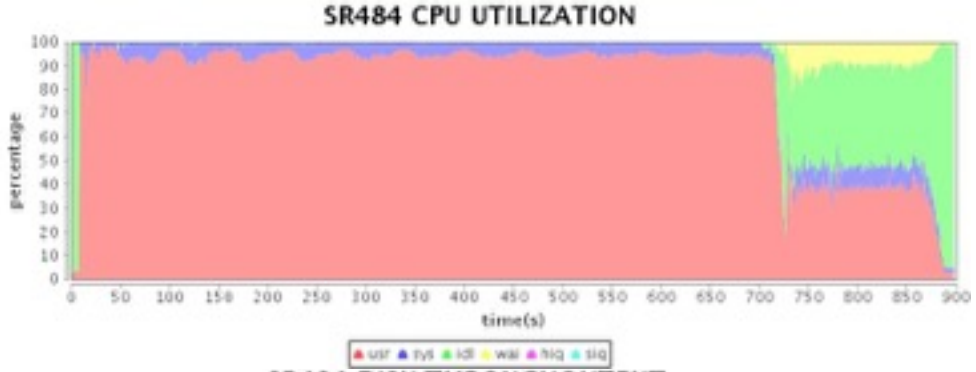
Performance Analysis – Data processing-round1

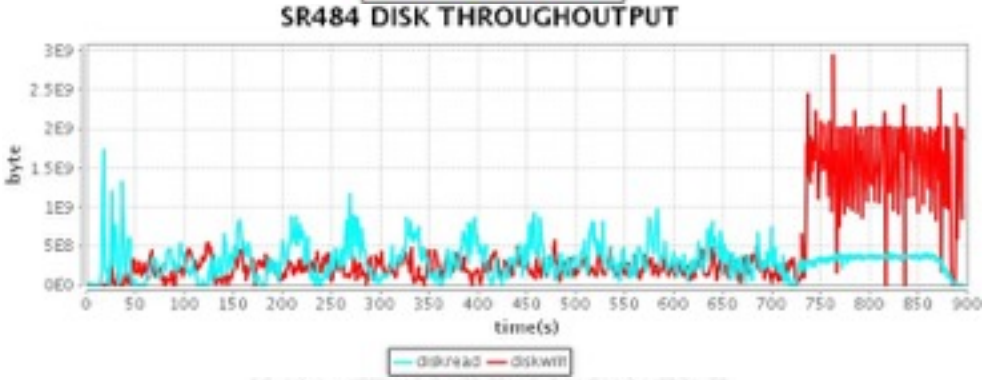


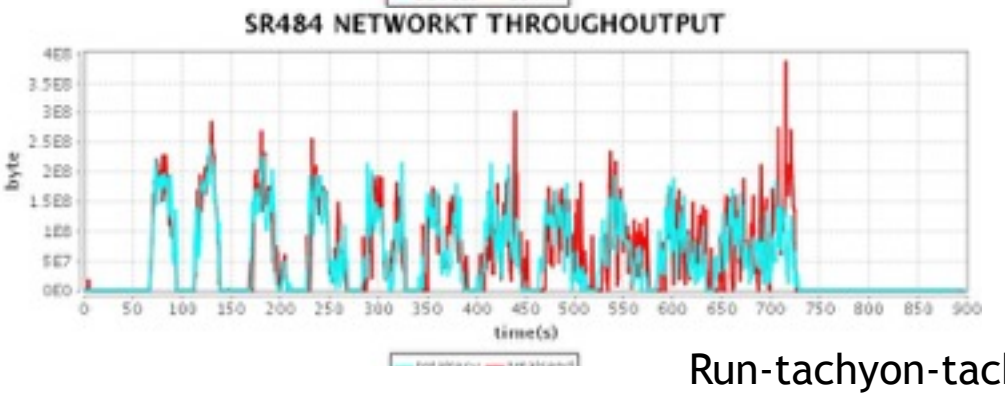




Run-hdfs-hdfs-1rep

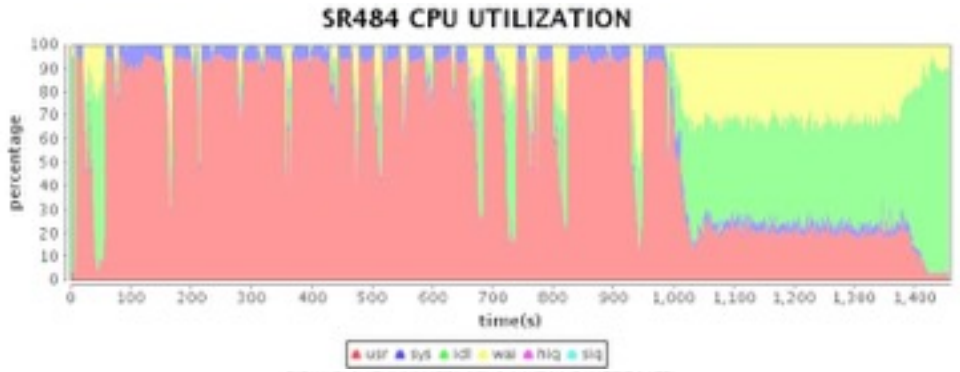


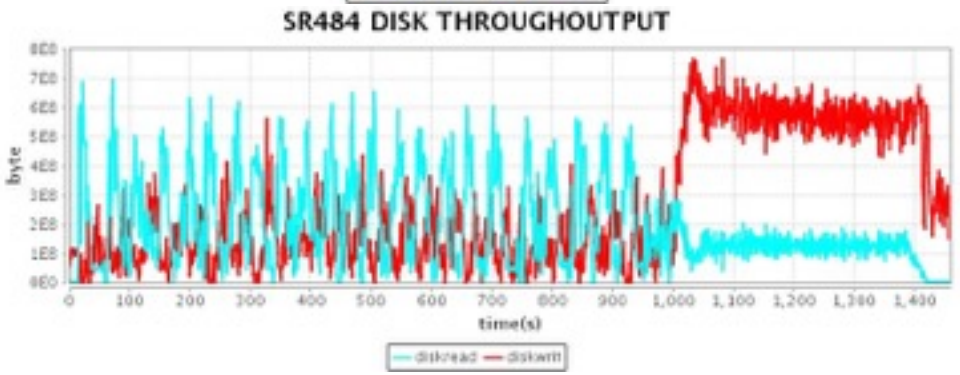


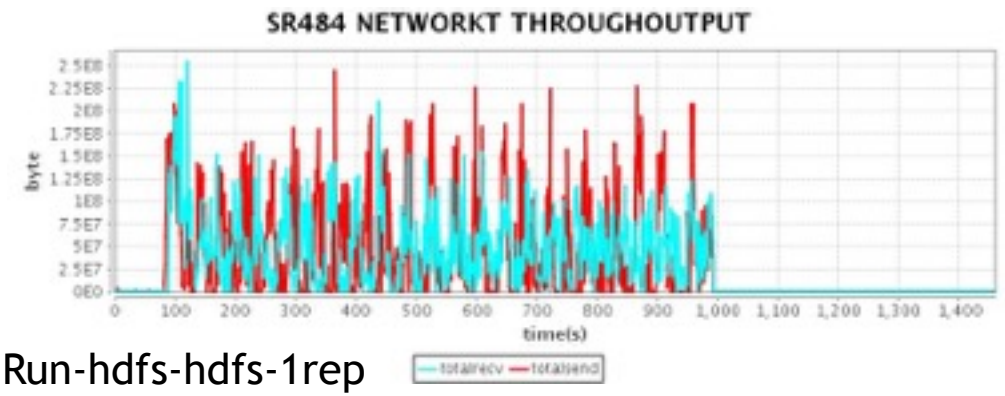


Run-tachyon-tachyon

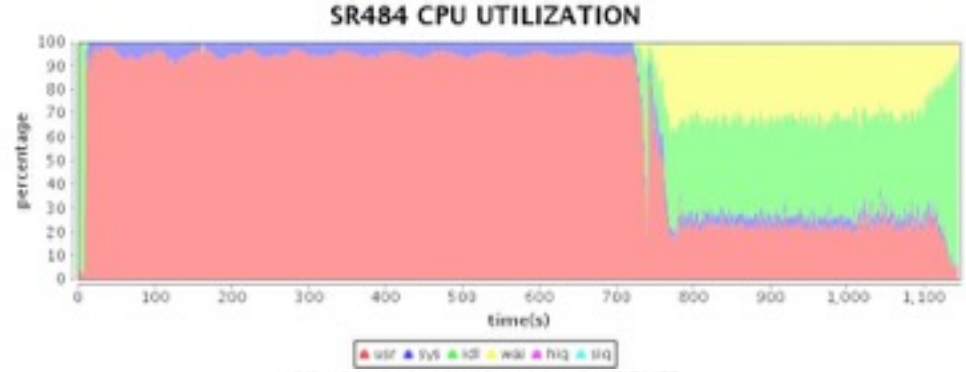
Performance Analysis – Data processing-round2

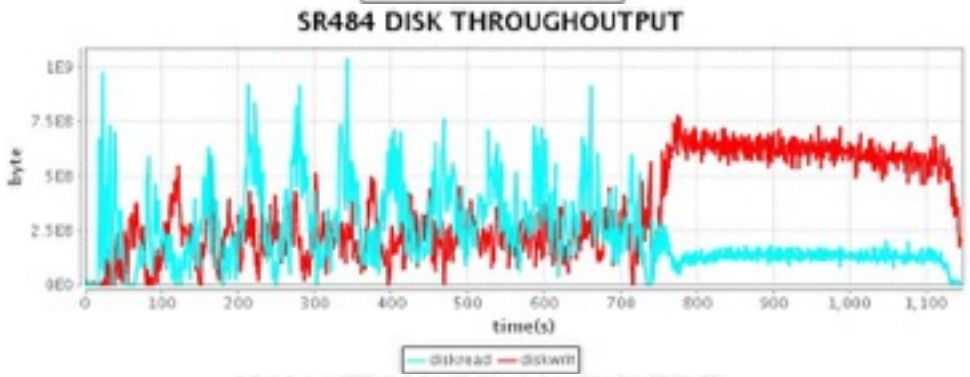


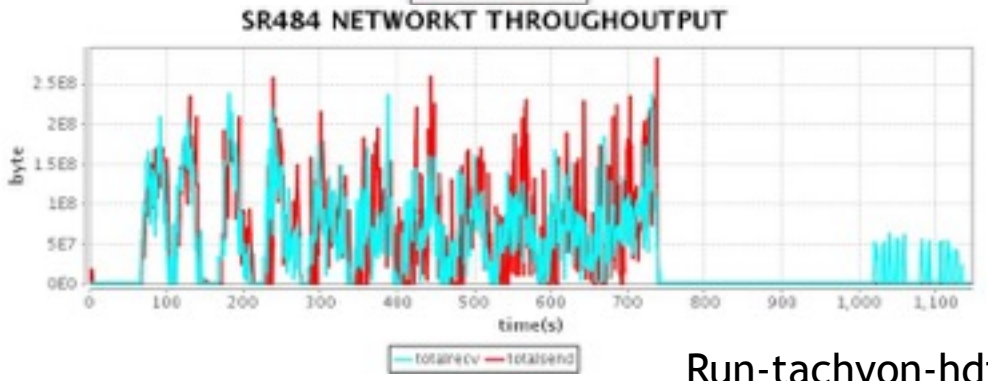




Run-hdfs-hdfs-1rep







Run-tachyon-hdfs-1rep

Experiment 3: Summary

- Sharing Intermediate data via Tachyon with NVM boost the execution
  - For the first phase, which is purely I/O intensive, it brings (2X) performance gain
  - For the second phase, it brings 1.2X ~1.6X performance gain

Outline

- Introduction and problem statement
- Introduction of Tachyon & NVM
- Performance testing and key learning
- Summary

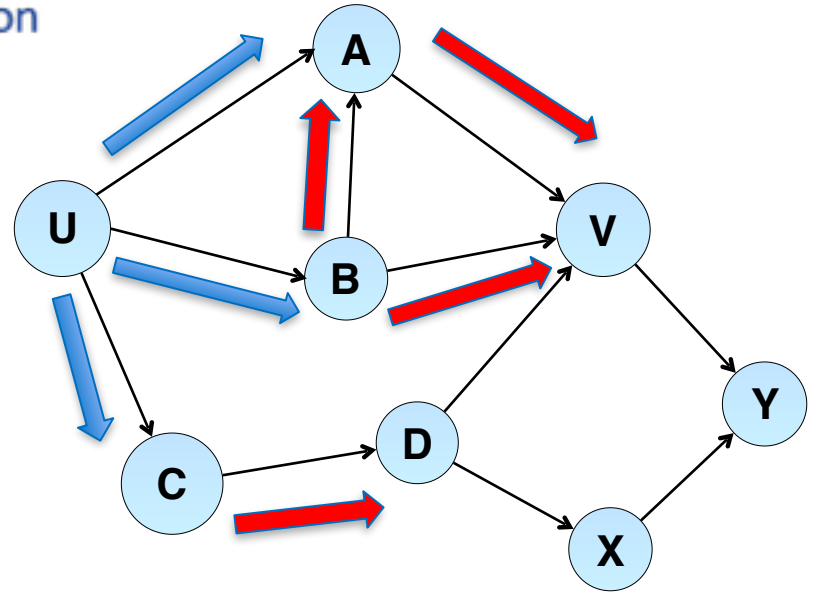
Conclusion

- Memory is the new disk
- Tachyon resolves the challenges in in-memory data management
- NVM will offer great performance for data access in Tachyon

Q & A

Test 1: Friends-of-friend

- Compute associations between two vertices n-hops away in a Graph
  - Getting weights between Vertices that have N degree association
  - Weightn(u, v) = $\sum k=1 \uparrow M \blacksquare Wp(U,V)k$ (M is the number of paths that have exactly n edges)
  - $Wp(U,V)k = \prod e=1 \uparrow n \blacksquare We$ (We is the weight of edge)
- A Graph Analysis case
  - E.g., friends of friend in social network
- Graph-parallel implementation
  - Bagel (Pregel on Spark)



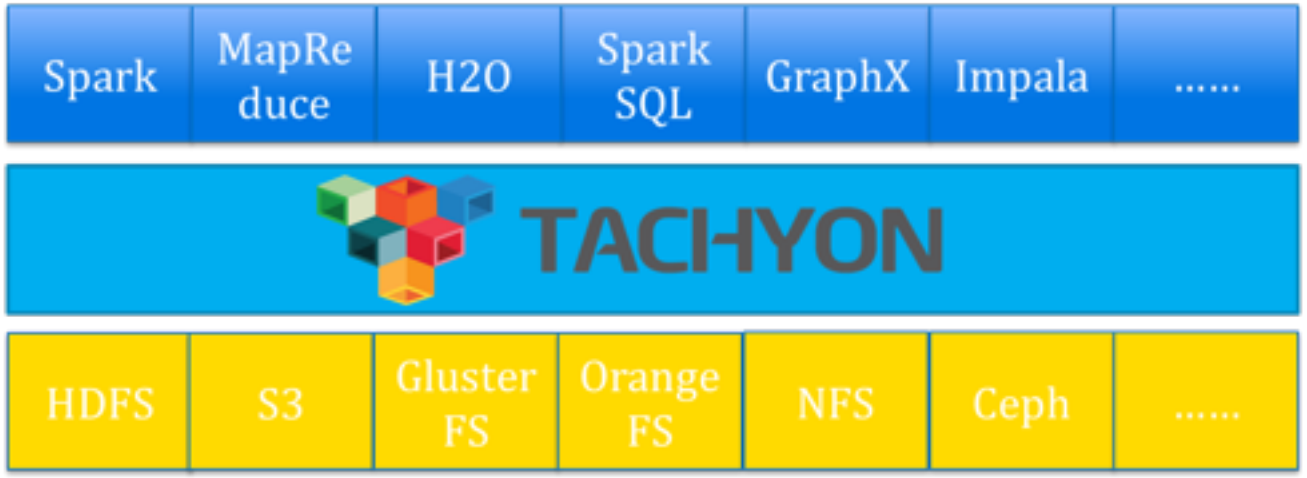
Problem statement

- In memory becomes more important and popular
  - Cost / capacity of Memory is lower and lower, which makes it possible to handle huge size of data in memory
  - Many computation frameworks leverage memory
  - How to manage huge caching data is an interesting problem
- There are still some challenges
  - Data sharing among applications
  - GC overhead introduced by in memory caching
  - When data is huge, external storage is still needed

Introduction of Tachyon

http://www.tachyonproject.org/

- Tachyon is an memory-centric distributed storage system enabling data sharing at memor-speed acorss cluster frameworks, such as Spark Mapreduce etc.
  - Caches working set files in memory and off-heap
  - Enables different jobs/queries and frameworks to access cached files at memory speed
- Features
  - Java-like file API
  - Hadoop file system compatible
  - Pluggable under layer file system
- Tiered block storage



Source: http://www.cs.berkeley.edu/~haoyuan/talks/Tachyon_2014-10-16-Strata.pdf

Test cases for remote data cache

- Input data is used by different applications
  - Input: event logs
  - Data format:
    - Timestamp\Category\ObjectId\EventID\ …
- Input data location:
  - Without Tachyon, all data is putted on remote HDFS cluster
  - With Tachyon, all data is cached in local Tachyon
- There are two cases, which share the same input data：
  - Case1 TopN
  - Case2 Nreach

Intro for TopN

- Compute the top N object in each category
  - Used fields from input data:
    - <Category\ObjectId\...>
  - select Category, ObjectId, count(*) as events from Input_data group by Category ,ObjectId order by events desc limit N where category='a';(for all categories)
  - Output:
    - Category\ObjectId\Visits
- It can be used to calculate:
  - The best selling products in each category
  - Most popular videos in each category

Intro for Nreach

- Computing unique event occurrence across the time range
  - Used fields from input data:
    - <TimeStamp, ObjectId, EventId, ...>
  - Output:
    - <ObjectId, TimeRange, Unique Event#, Unique Event(≥2)#, …, Unique Event(≥n)#>
    - Accumulated unique event# for each object and certain time range
- It is used to calculate:
  - Visits of certain advertisement by unique users in one day/week

Tiered storage in Tachyon

- Tiered block storage is used to extend Tachyon's caching space with external storage, such as SSD HDD etc.
  - Different tiers have different speed and priority
  - "Hot" data and "warm" data are putted on different layers
  - Multiple directories in single tier
- Data migration among tiers
  - "Hot" data can be evicted to lower layer, when it is not "hot" any longer by eviction strategies.
  - "Warm" data can be promote to top layer, when it becomes "hot" again.
- It is available since Tachyon 0.6 Release
  - The JIRA for tiered storage: TACHYON-33