Smart Cache Management for Big Data

Shi, Mingfei mingfei.shi@intel.com
Duan, Jiangang jiangang.duan@intel.com
Huang, Jie jie.huang@intel.com

Agenda

- Introduction and Problem statement
- Introduction of Spark & Tachyon
- Performance testing and key learning
- Summary

Agenda

- Introduction and Problem statement
- Introduction of Spark & Tachyon
- Performance testing and key learning
- summary

Introduction

- Intel Cloud and BigData Engineering Team
  - Working with the community to optimize Apache Spark and Hadoop on Intel platform
  - Improve Spark scalability and reliability
  - Deliver better tools for management, benchmarking, tuning – HiBench, HiMeter
  - Working with China Internet company to build Spark based solution

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 memory and other fast storage media to hold data is an interesting problem
  - There are still some challenges, such as GC overhead and data sharing
  - When data is huge, memory can't fit it in, external storage is still needed
- We share our learning w/ spark and tachyon in a customer workload

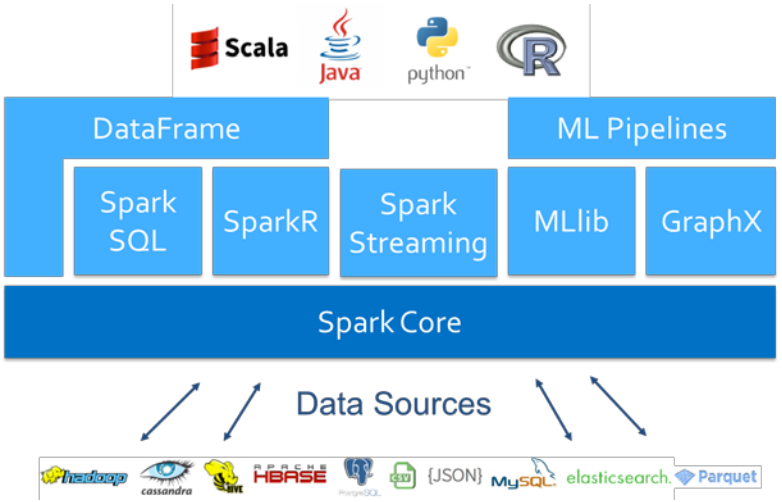
Agenda

- Introduction and Problem statement
- Introduction of Spark & Tachyon
- Performance testing and key learning
- summary

Introduction of Spark

http://spark.apache.org/

- Spark is a fast and general in-memory cluster computing system, interoperable with Apache Hadoop
  - Write programs in terms of coarse-grained transformations on distributed datasets
  - Concept: resilient distributed datasets (RDDs)
- Improves efficiency through:
  - In-memory computing primitives
  - General execution graphs
- Improves usability through:
  - Rich APIs in Java, Scala, Python, R
  - Interactive shell

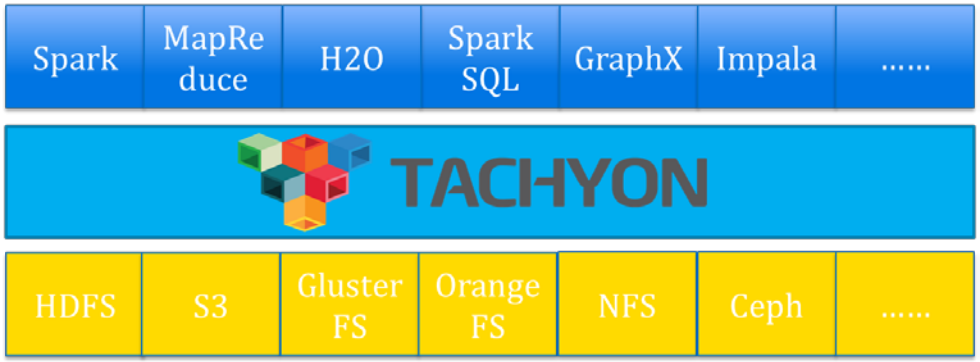


Source: https://dn-idzvideo.qbox.me/Spark_AMPCamp_China.pdf

Introduction of Tachyon

http://www.tachyonproject.org/

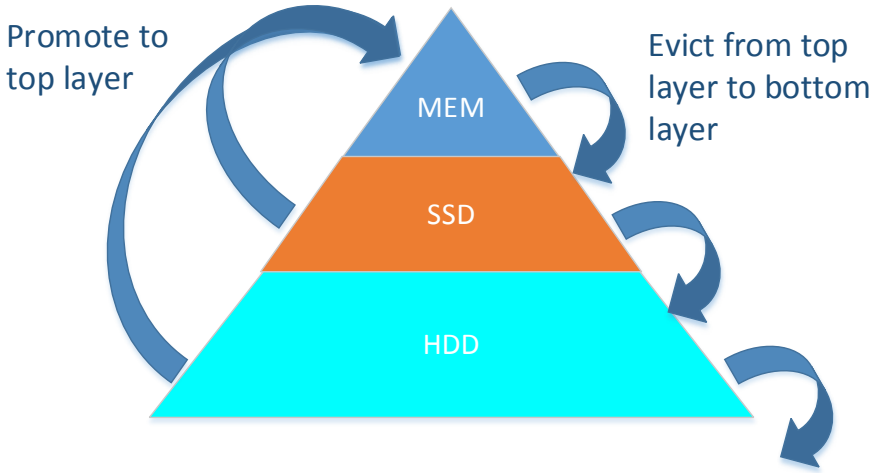
- Tachyon is an memory-centric distributed storage system enabling data sharing at memor-speed acorss cluster frameworks, such as Spark Mapreduce etc.
  - Caches working set files in memory and off-heap
  - Enables different jobs/queries and frameworks to access cached files at memory speed
  - Avoids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



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

Tiered storage in Tachyon

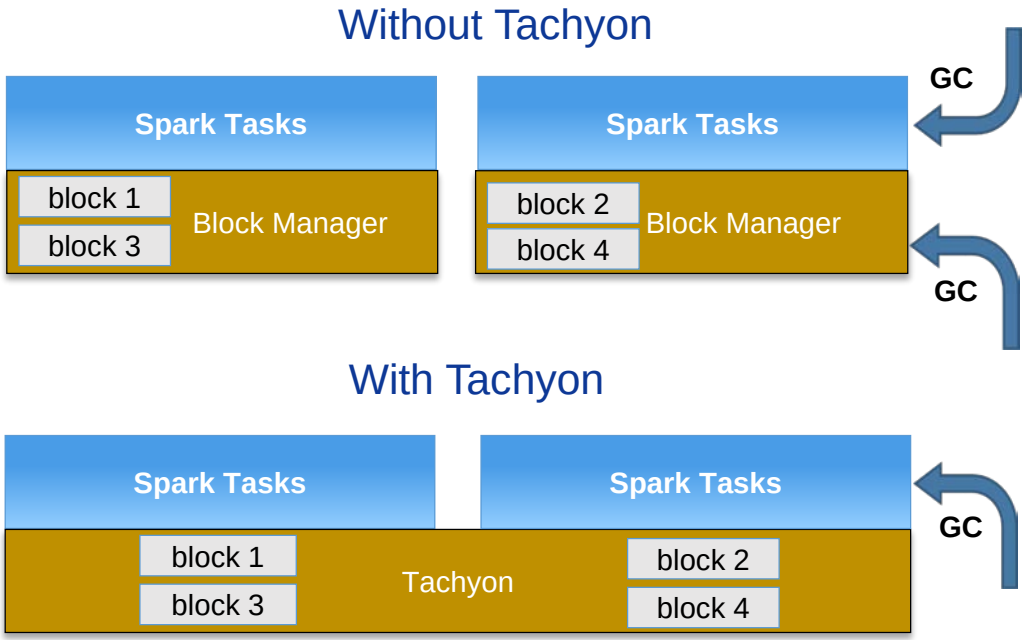
- Tiered block storage is used to extend Tachyon's caching space with external storage, such as SSD HDD etc.
  - Different tiers have different speed and priority
  - "Hot" data and "warm" data are putted on different layers
  - Multiple directories in single tier
- Data migration among tiers
  - "Hot" 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



Spark & Tachyon Integration

- Tachyon is default Off-heap solution for Spark currently. And there are two work model of using Tachyon in Spark jobs.
  - Input/Output data with Tachyon
    - Take Tachyon as input/output file system, just like HDFS
  - Persist Spark RDDs into Tachyon
    - Setting storage level of RDD to OFF_HEAP
- By integrating Tachyon into Spark, it will bring:
  - Fast data access at memory speed
  - Eliminate GC overhead by placing data to OFF_HEAP

GC overhead in Spark



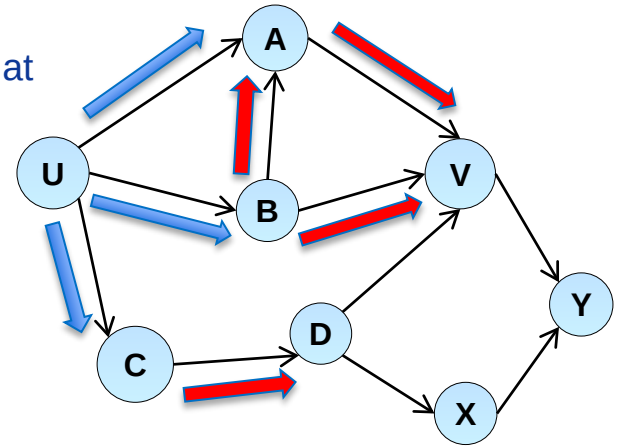
- GC overhead can be divided into two parts:
  - Over temporary data generated during task execution
  - Over caching data in Spark's block manager
- Tachyon can eliminate the second part of GC overhead

Agenda

- Introduction and Problem statement
- Introduction of Spark & Tachyon
- Performance testing and key learning
- summary

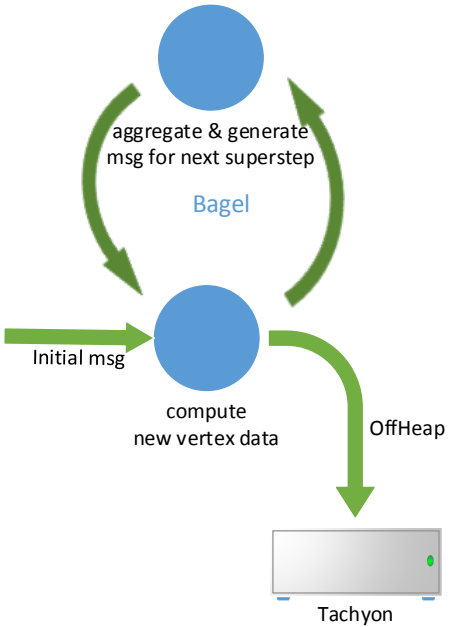
Test case description

- Compute associations between two vertices that are n-hop away
  - Way of getting Vertices that have N degree association
  - Weightn(u, v) = $\sum_{k=1}^{M} \boldsymbol{Wp}\ (\boldsymbol{U},\boldsymbol{V})\boldsymbol{k}$ (M is the number of paths that have exactly n edges)
  - $\boldsymbol{Wp}(\boldsymbol{U},\boldsymbol{V})\boldsymbol{k} = \prod_{e=1}^{n} We$ (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)



Bagel over Tachyon

- Bagel is a Spark implementation of Google's Pregel graph processing framework.
  - In Pregel programming model, jobs run as a sequence of iterations called supersteps.
  - In each superstep, each vertex in the graph runs a user-specified function that can 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
  - Eliminate GC overhead because of huge data size and long running time



Test Bed

- One master node with four worker nodes

Worker node Configuration	
CPU	SNB 2660 @ 2.2G 32 logical cores
Memory	192GB DDR3 @ 1066 MHZ
Disk	4 * 400G SSD（S3700）
OS Distribution	Redhat 6.2 (kernel 2.6.32-220.el6.x86_64)
JDK version	JDK1.7.0_79 (64 bit server)
Spark version	1.4.1 release
Tachyon version	0.7.0-SNAPSHOT
Hadoop version	2.3.0

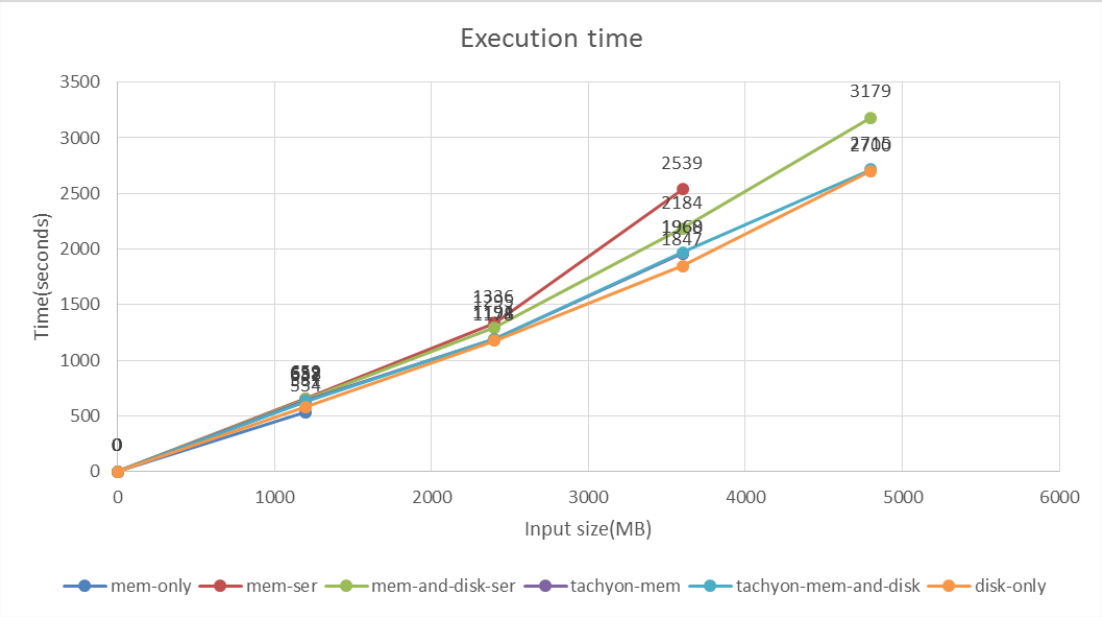
Common configuration

- Total memory size used on each machine: 140G
  - Size of SPARK_MEM + TACHYON_MEM
- Spark configuration:
  - Spark local directory: 4 SDD
  - GC strategy: Parallel GC
- Tachyon tiered storage configuration
  - Level1: MEM
  - Level2: 4 SSD

Workload configuration

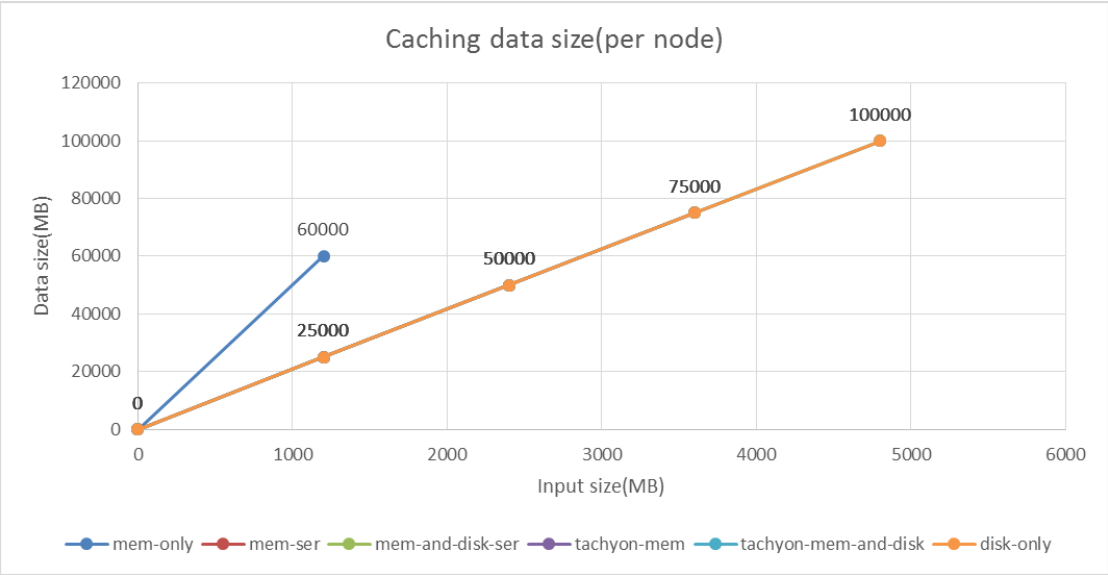
Data size	Configuration					
	MEM_ONLY	MEM_SER	MEM_AND_DISK_SER	DISK_ONLY	TACHYON_MEM	TACHYON_MEM_AND_DISK
1200MB	SPARK_MEM=140G MEM_FRACTION=0.6	SPARK_MEM=140G MEM_FRACTION = 0.6	SPARK_MEM=140G MEM_FRACTION = 0.1	SPARK_MEM=140G MEM_FRACTION = 0.1	SPARK_MEM=115G TACHYON_MEM=25G MEM_FRACTION = 0.1	SPARK_MEM=125G TACHYON_MEM=15G MEM_FRACTION = 0.1
2400MB	N/A	SPARK_MEM=140G MEM_FRACTION=0.6	SPARK_MEM=140G MEM_FRACTION = 0.2	SPARK_MEM=140G MEM_FRACTION = 0.1	SPARK_MEM=90G TACHYON_MEM=50G MEM_FRACTION = 0.1	SPARK_MEM=115G TACHYON_MEM=25G MEM_FRACTION = 0.1
3600MB	N/A	SPARK_MEM=140G MEM_FRACTION = 0.8	SPARK_MEM=140G MEM_FRACTION = 0.3	SPARK_MEM=140G MEM_FRACTION = 0.1	SPARK_MEM=65G TACHYON_MEM=75G MEM_FRACTION = 0.1	SPARK_MEM=105G TACHYON_MEM=35G MEM_FRACTION = 0.1
4800MB	N/A	N/A	SPARK_MEM=140G MEM_FRACTION = 0.35	SPARK_MEM=140G MEM_FRACTION = 0.1	N/A	SPARK_MEM=100G TACHYON_MEM=40G MEM_FRACTION = 0.1

Test result – execution time



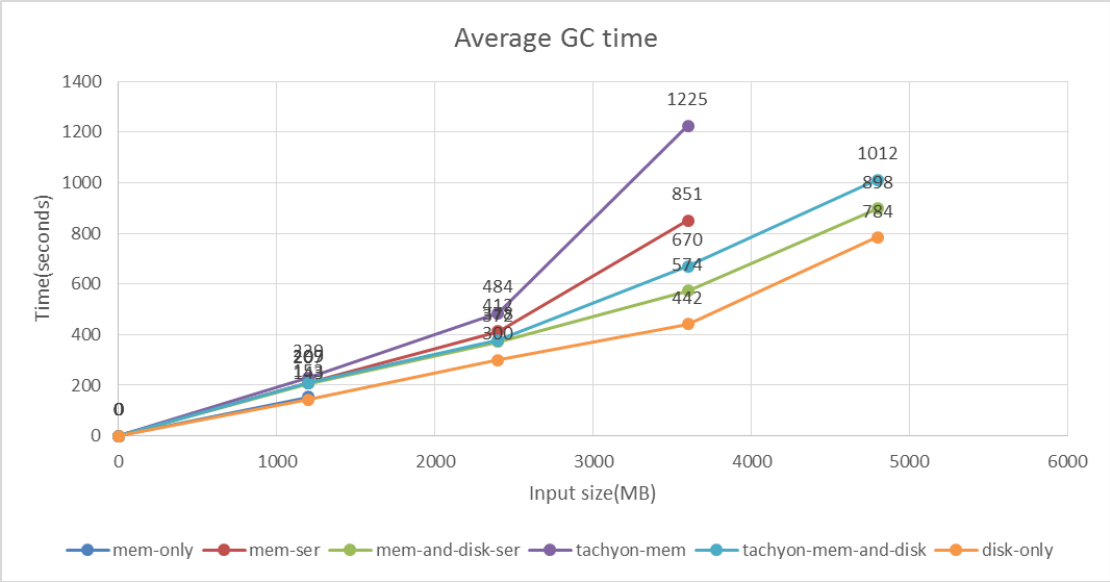
- With small data size(1200MB), MEM_ONLY gets the best performance, others are similar
- With medium data size(2400MB), MEM_ONLY failed to run because of huge caching data size.
- With large data size(3600MB), TACHYON_MEM and TACHYON_MEM_AND_DISK out performs MEM_SER and MEM_AND_DISK_SER(10+%), DISK_ONLY gets the best performance
- With huge data size(4800MB) , TACHYON_MEM and MEM_SER failed to run because of huge caching data size, TACHYON_MEM_AND_DISK outperforms MEM_AND_DISK_SER(10+%), DISK_ONLY and TACHYON_MEM_AND_DISK are very similar

Test result – caching data size



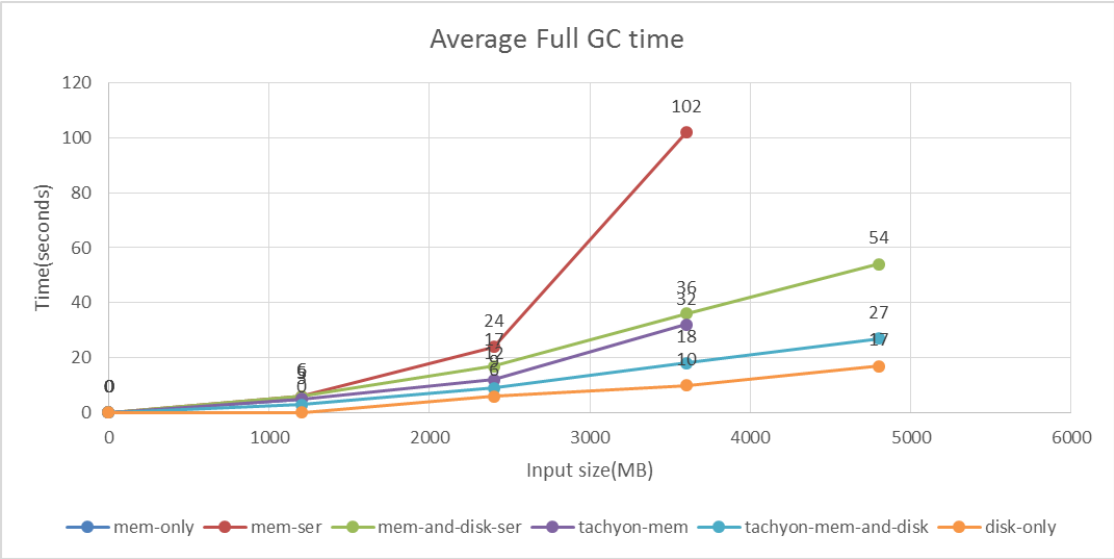
- The data size without serialization is about three times of the size with serialization.
- Without serialization, the application can only successfully run on small data size, though it gets the best performance

Test result – total GC time



- GC time is very related to the HEAP size that allocated to the Spark executor engine
- DISK_ONLY has the least GC overhead, because it uses disk space to act as memory
- TACHYON_MEM has the biggest GC overhead, because it allocates smallest HEAP size, much memory is used as Tachyon's cache space

Test result – full GC time



- Full GC time is very related to both the memory allocated to Spark executor and long live caching data size on heap
- MEM_SER has the biggest full GC overhead, because all caching data is on HEAP
- DISK_ONLY has the least Full GC overhead

Test summary

- Performance
  - With SSD, Spark with DISK_ONLY gets the best performance
  - When memory is not enough, use external storage(large but slower) as additional storage will speed up execution, compared with memory only
  - TACHYON_MEM and TACHYON_MEM_AND_DISK outperforms MEM_SER and MEM_AND_DISK_SER in Spark, The performance gain comes from:
    - Enhancement of disk IO throughput
    - Eliminate GC overhead
- GC overhead
  - DISK_ONLY gets the least GC overhead, including full GC overhead
  - Tachyon can not eliminate GC overhead all the time
    - Take the memory space that can be allocated to application
    - Have good effect for full GC overhead

Key learning

- Not use memory only mode (including ser etc) – because the run fail ratio may be high
- When memory size is too big – the GC overhead may become higher and higher – in this case, we suggest to use off heap mode - TACHYON_MEM_AND_DISK out performs MEM_SER and MEM_AND_DISK_SER(10+%)
- Current DISK_ONLY and TACHYON_MEM_AND_DISK are very similar – however, we believe TACHYON_MEM_AND_DISK can provide better flexibility – e.g.
  - Mem+SSD+HDD w/ higher disk size and better performance.
  - Hot data will be cached in memory – thus reduce the read latency
- We are working on asynchronous mode – which is expected to improve TACHYON_MEM_AND_DISK performance
  - The JIRA for asynchronous eviction: TACHYON-334

Agenda

- Introduction and Problem statement
- Introduction of Spark & Tachyon
- Performance testing and key learning
- summary

Summary

- Memory is the new disk
- Memory management (GC) is the key problem for big data application
- Tiered storage (mem+SSD+HDD) w/ offheap can provide better performance

Q & A

Legal Notices and Disclaimers

Legal Information: Benchmark and Performance Claims Disclaimers

Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase.

Test and System Configurations: See Back up for details.

For more complete information about performance and benchmark results, visit http://www.intel.com/performance.