Experience and Lesson Learned to boost Real-World Spark Applications

Grace Huang (jie.huang@intel.com)
Intel/SSG/Big Data Technology

About Us

- Intel Big Data Technology on open source engineering
- Long history in Spark with AMPLab and community
  - Key contributions to grow itself and its ecosystem
    - E.g., shuffle improvement, yarn-client, metric framework and etc.
  - Among Top contributors
- Intel partnering with several large organizations/websites in China since 2012
  - Building real-world big data analytic applications using Spark stack

Building next-gen big data analytics

- Advanced ML and Graph Analysis
  - Relationship analysis
  - Similarity measure
  - Community detection
  - Deep learning
- Complex, Interactive OLAP/BI
  - Interactive/ad-hoc OLAP Analysis
  - Batch style OLAP Analysis
- Real-time* Stream processing
  - Log analysis

Experience from partnership

- Significant speedup in real-world applications
  - x5-100 performance gains versus to Hadoop MR
- Easy of use on a common deployment
  - Iterative, complex machine learning & graph analysis
  - Complex, Interactive OLAP/BI
  - Real-time analytical processing (RTAP)
- Spark application can perform even better
  - Robustness, Usability, Stability, Performance

Lessons learned

1. Manage Memory
2. Improve IO
3. Optimize Computations

Lessons learned

1. Manage Memory
2. Improve IO
3. Optimize Computations

Free memory space timely (1)

- Commonly use Bagel/GraphX for graph analytics
- The present iteration only depends on its previous step
  - I.e., RDD[n] is only used in RDD[n+1] computation
- Problem statement:
  - Memory space is continuously increased in Bagel app

Iteration	Cache Size/iteration	Total Cached Size (before optimize)
Initial	4.3G	**4.3G**
1	8.2G	**12.5G**
2	98.8G	**111.3G**
3	90.8G	**202.1G**

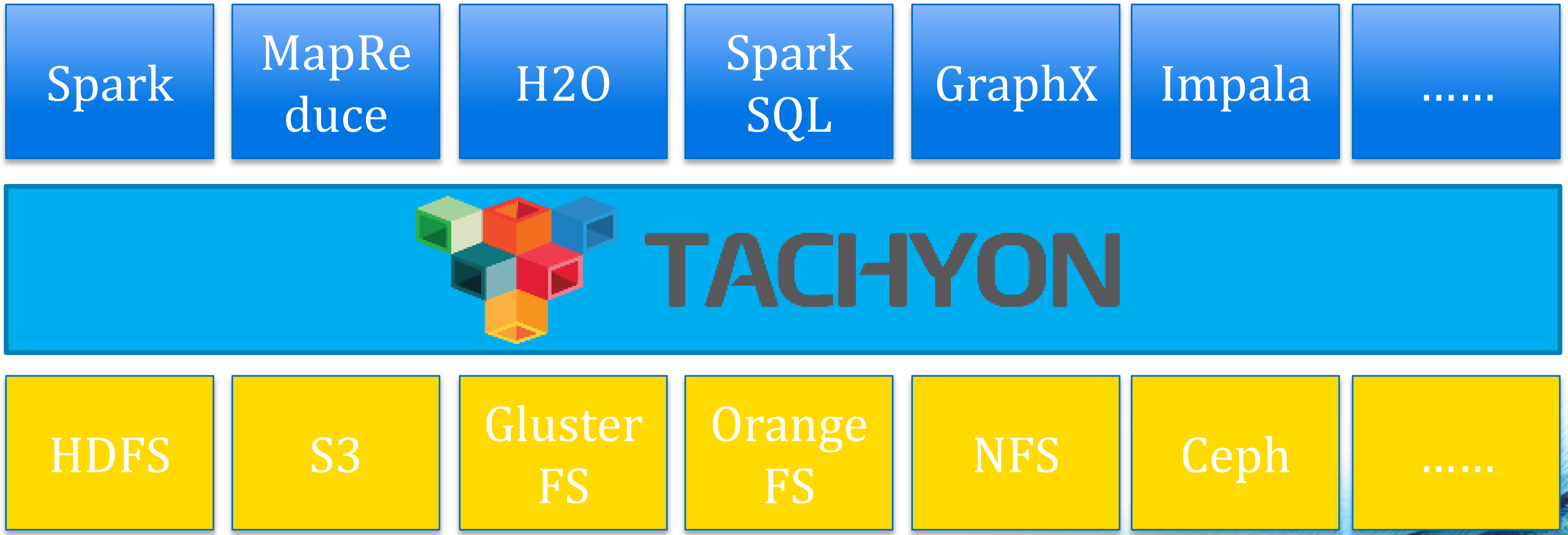
Free memory space timely (1)

- Free those obsolete RDDs not be used anymore
  - I.e., To *un-persist* RDD[n-1] after RDD[n] is done SPARK-2661
- The total memory usage is **> 50% off**

Iteration	Cache Size/iteration	Total Cached Size (before optimize)	Total Cached Size (after optimize)
Initial	4.3G	**4.3G**	**4.3G**
1	8.2G	**12.5G**	**8.2G**
2	98.8G	**111.3G**	**98.8G**
3	90.8G	**202.1G**	**90.8G**

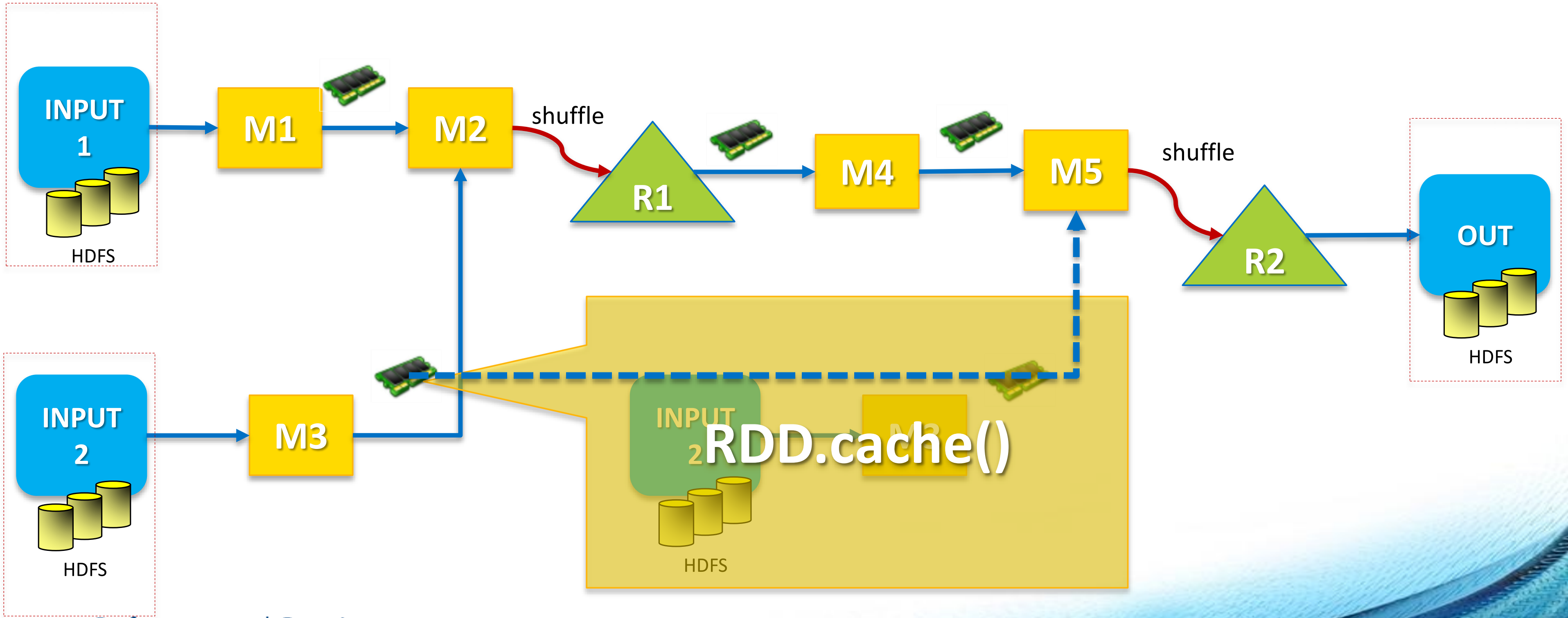
Use off-heap memory by Tachyon

- A Reliable Memory Centric Distributed Storage System

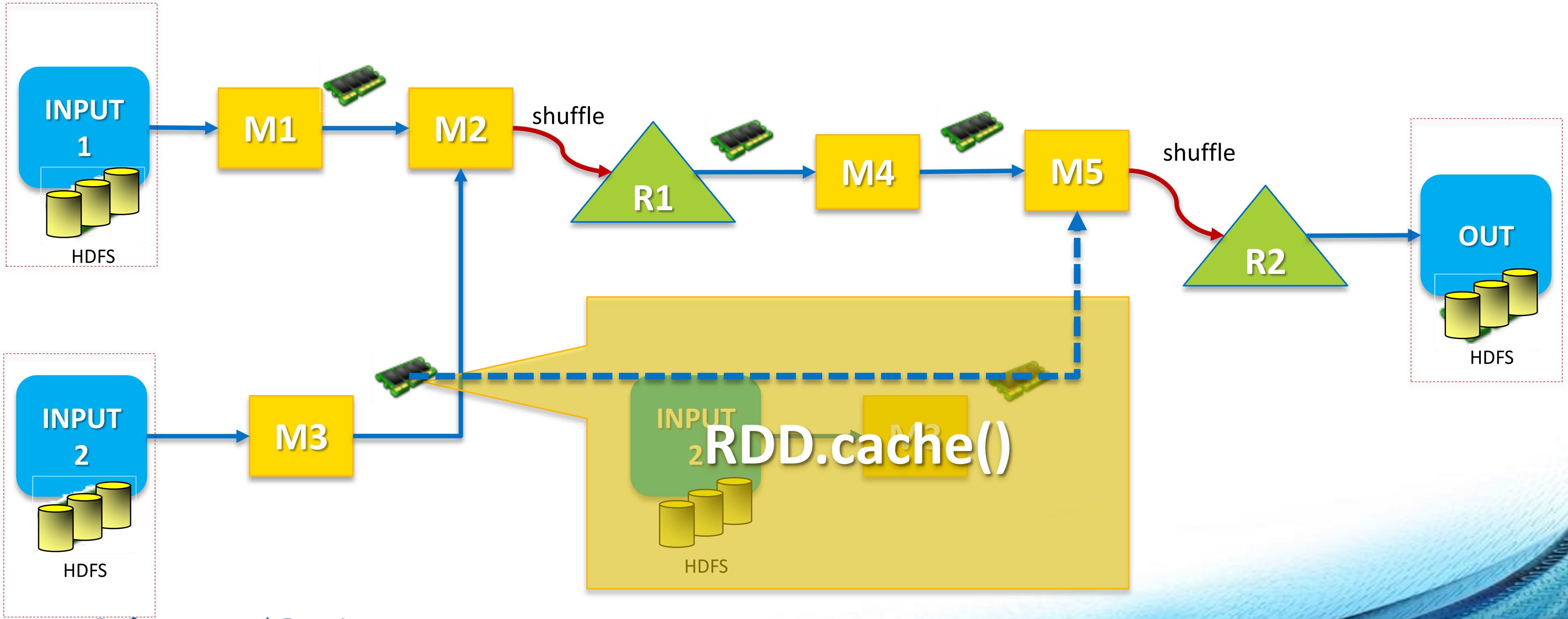


From HY 2014-11-21 AMPCamp

Spark – monopolized memory data

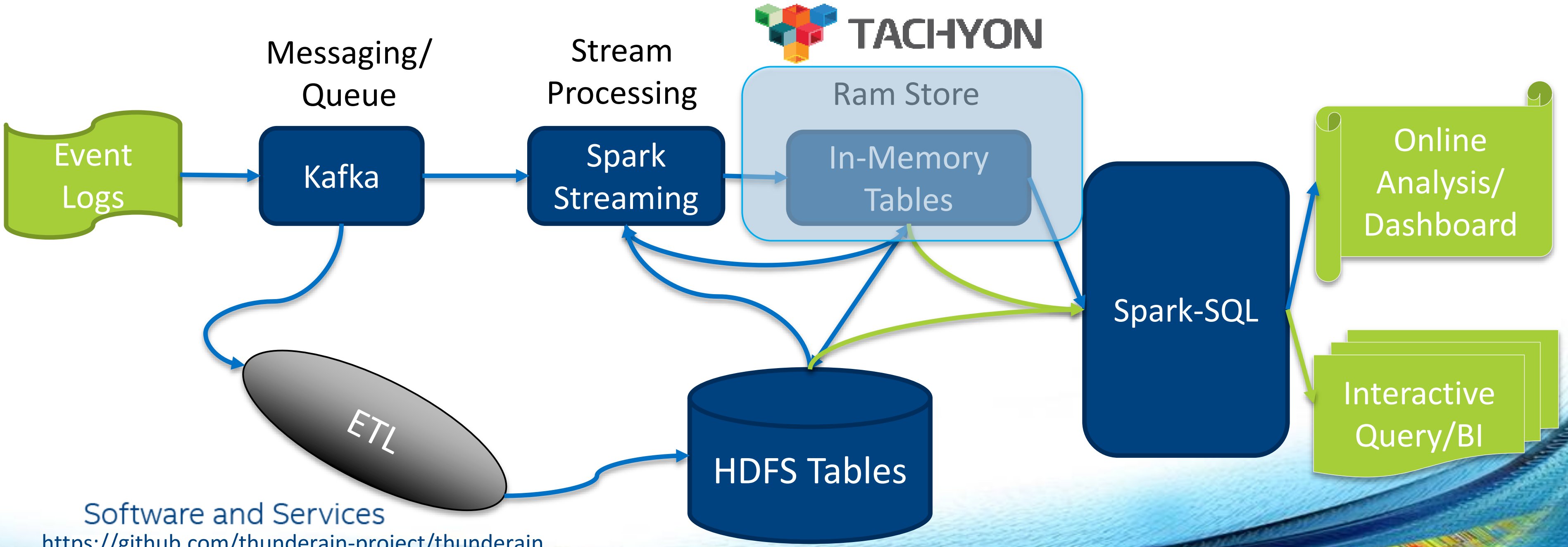


Tachyon – cross-jobs memory share



Use off-heap memory

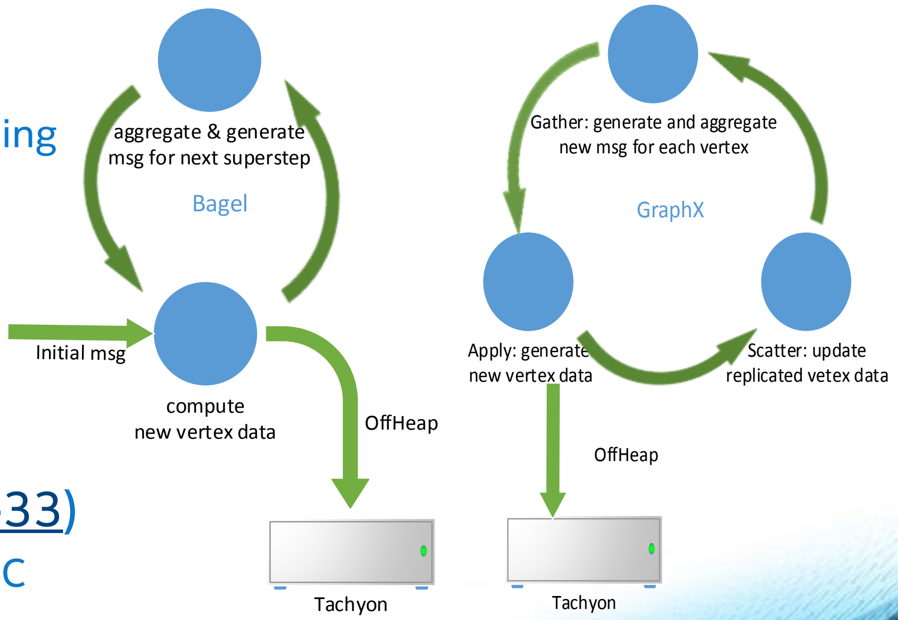
- Sharing in-memory data among different apps/frameworks



https://github.com/thunderain-project/thunderain

Use Tachyon to boost in-mem computation

- Advanced iterative graph analytics
  - Cache iteration data
  - memory is far more than enough to caching entire graph data
- Problem statement:
  - OOM exception by on-heap & GC
- Tachyon Hierarchy store(TACHYON-33)
  - Brings **>30%** performance gain by less GC overhead (vs. Mem_and_ser SL)

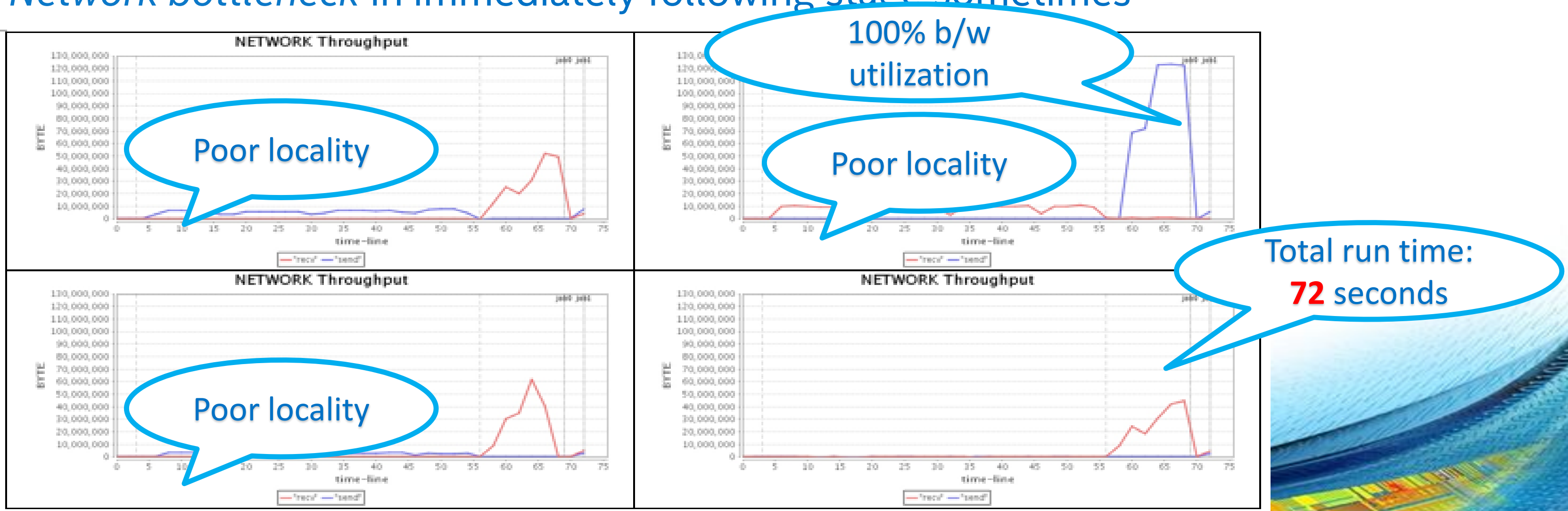


Lessons learned

1. Manage Memory
2. Improve IO
3. Optimize Computations

Improving task locality in scheduler

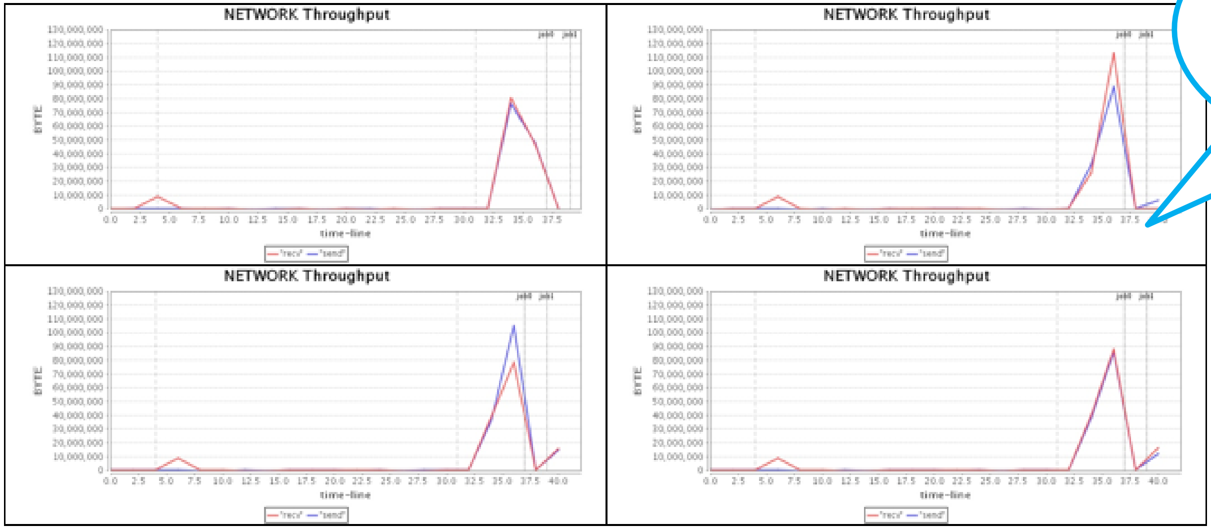
- Poor locality for not enough executors registered
- Problem Statement:
  - *Extra network transportation*
  - *Network bottleneck* in immediately following stage sometimes



Improving task locality in scheduler

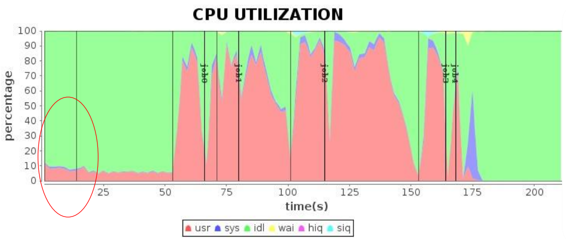
- Wait till enough executors ready SPARK-1946, SPARK-2635
  - Tunable knobs: *spark.scheduler.minRegisteredResourcesRatio*, *spark.scheduler.maxRegisteredResourcesWaitingTime*
- Prioritized locality list SPARK-2193

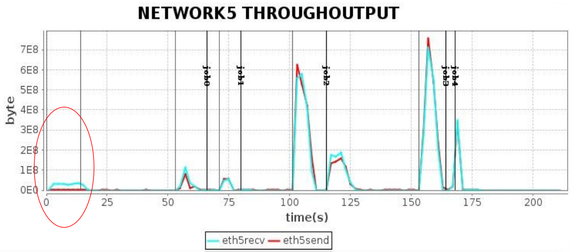
Total run time:
40 (**72**) seconds,
x1.75 speedup

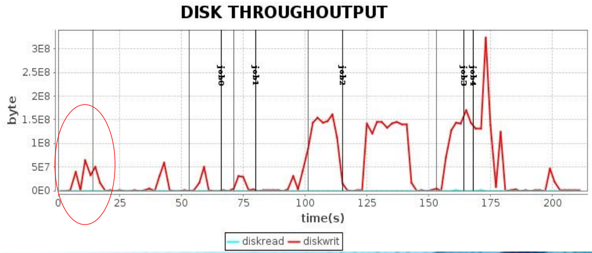


Redundant Jar file copies

- Mostly to run Spark on Yarn in data center
- Each executor copies one job jar in Yarn
- Problem statement:
  - Co-located executors(containers) on the same NM have redundant copies
  - Leads to network/disk IO bandwidth consumption with big files
  - Causes long time dispatching period in bootstrap

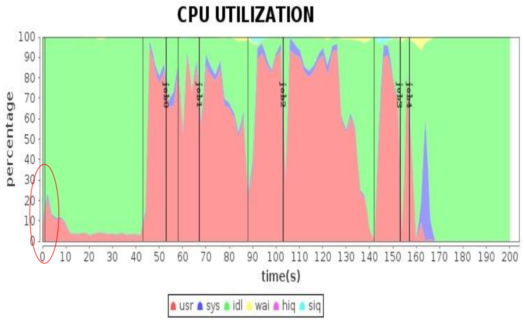


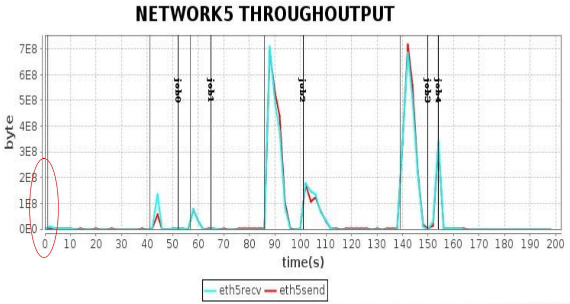


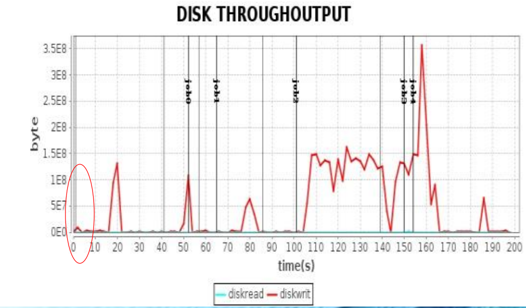


Redundant Jar file copies

- Only send jar file once for those co-located executors in Yarn SPARK-2713
- **>10x** speedup in bootstrap

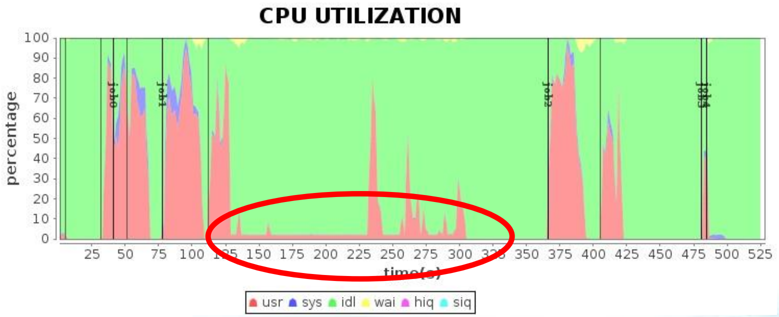


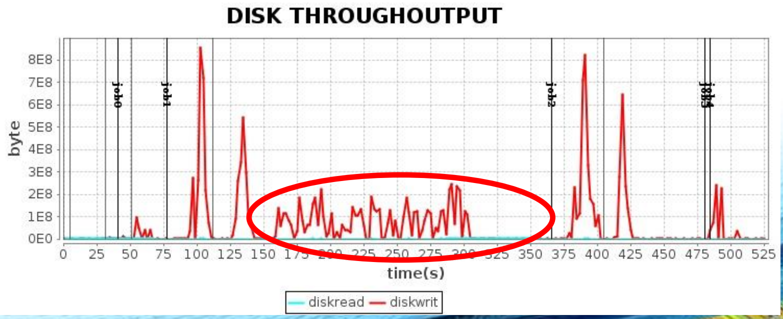




Spread out IO loads in RDD Cache

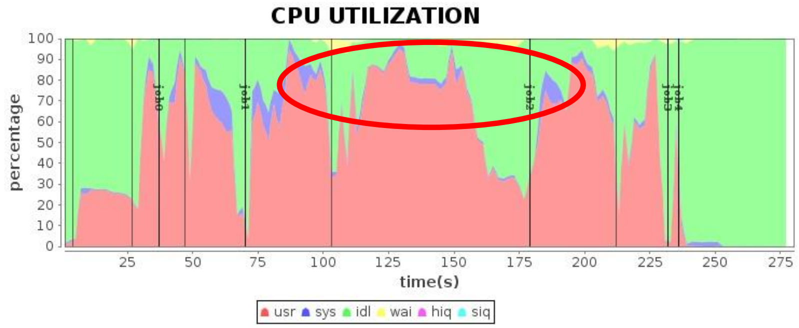
- Usually large overflow data onto disk in RDD cache
  - To choose MEMORY_AND_DISK as preferred storage level
- Problem statement:
  - Exists synchronization issue while flushing to disks (i.e., only single HDD BW is used)
  - Comma separated storage list in Spark doesn't help

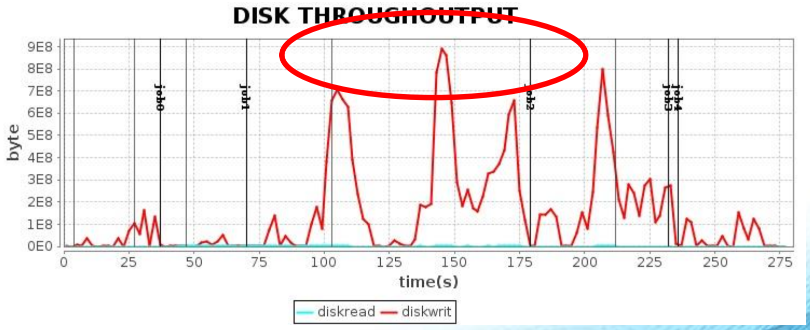




Spread out IO loads in RDD Cache

- Resolve the synchronization issue in data spill (SPARK_3000)
- Increased concurrent IO throughputs, leading to higher CPU utilization
- Speedup **x3+** in RDD cache





Lessons learned

1. Manage Memory
2. Improve IO
3. Optimize Computations

Sort-based Shuffle Read

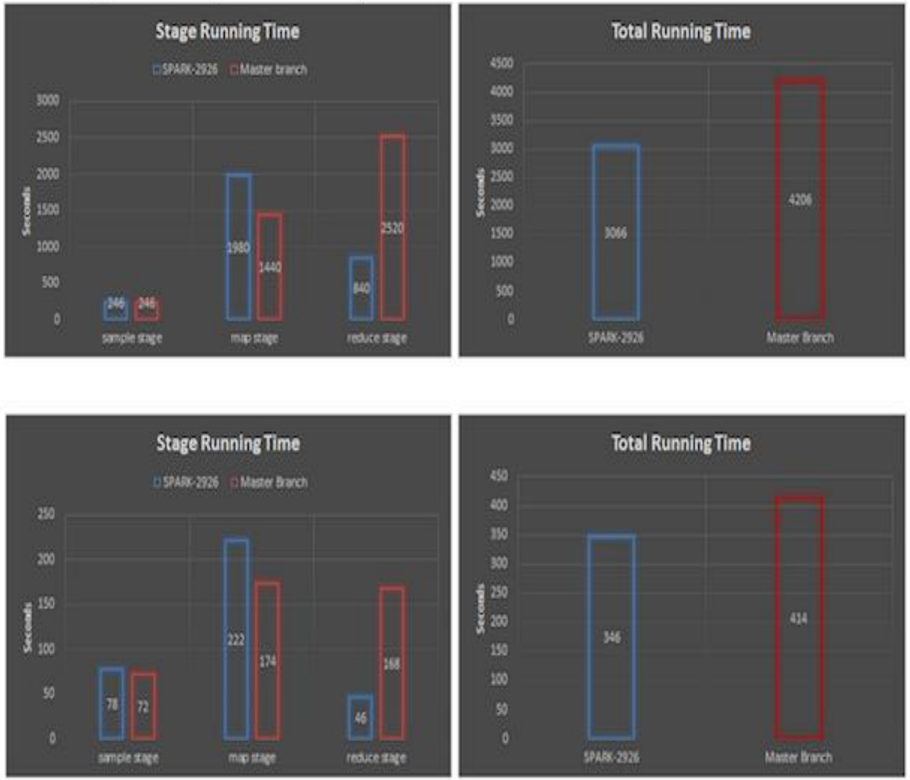
- Sort is popular in OLAP query (e.g., ORDERBY, DISTINCT, JOIN, …)
- Sort based shuffle write introduced for scalability
    - I.e, Map outputs have been sorted in shuffle-out

- Problem statement:
    - Shuffle read is still hash based (not fully resolved)
    - Sort-liked Ops need extra comparison again after shuffle
    - possible to leverage sorted map outputs in reduce(i.e., shuffle-in)?

Sort-based Shuffle Read

- Save extra sort effort by leveraging sorted shuffle-out (SPARK_2926)
  - Sort records inside each partition in shuffle write (i.e., map)
  - Merge sorted data in memory space as more as possible
  - Return back sorted list in reduce

- **x2** performance gains in reduce
- Provide MR-style shuffle results for easy migration

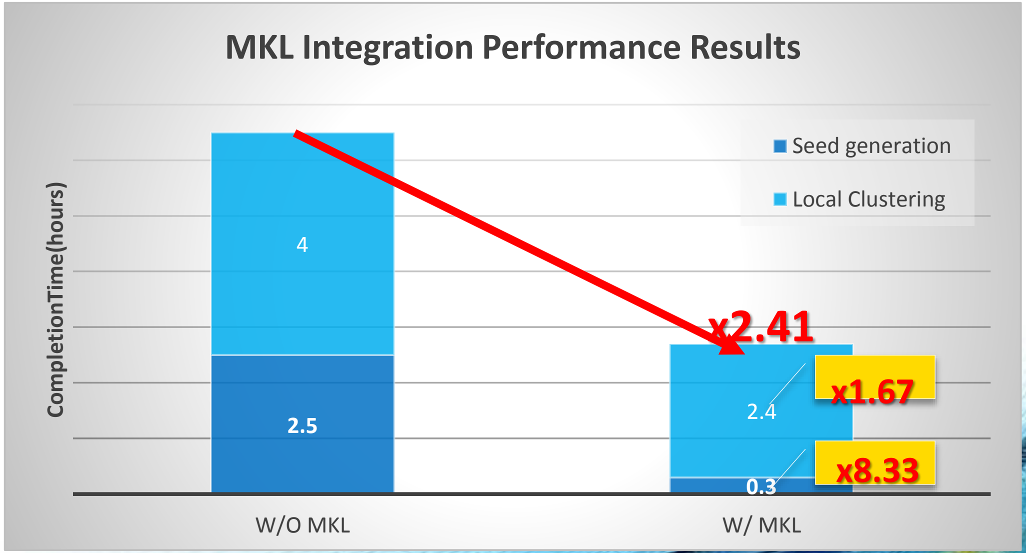


Speedup calculation by MKL

- Iterative, complex machine learning & graph analysis
  - Complex matrix (e.g., multiplication) dominates CPU time
  - Requires sparse matrix libs

- Our approach brings x2-4 speedup
  - Native math lib
  - CPU instruction

level optimization



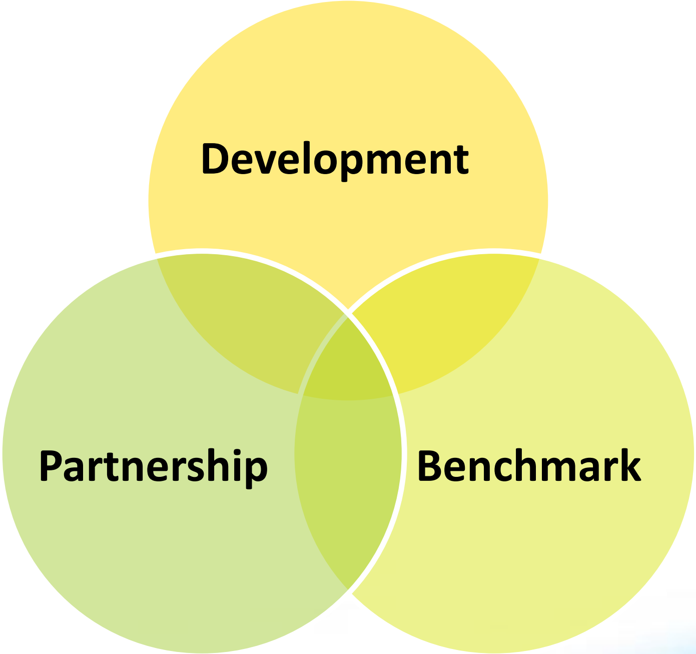
To measure your Spark deployment

- HiBench – big data benchmark (https://github.com/intel-hadoop/HiBench)
  - Consists of 10 representative kernel workloads
  - Hadoop MR, Spark, Streaming*
  - MR1/Standalone, Yarn

- Dew – big data profiler (public release upcoming soon)
  - A lightweight big data profiler (non-intrusively)
  - Support Spark's workflow based breakdown reports
  - Web portal for both brief and application specified info

Current & Future focuses in Intel

- Key contributions to grow Spark and its ecosystem by
  - Robustness,
  - Usability,
  - Scalability,
  - Performance



Summary

- Spark plays an important role in big data
- Lessons learned to speed up Spark even more – A more balanced system performs better
- Continuously mature Spark for next-gen big data on IA altogether

Notices and Disclaimers

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
  A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.
- Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.
- The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
- Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm
- Intel, the Intel logo, Intel Xeon, and Xeon logos are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.
Copyright © 2015 Intel Corporation. All rights reserved.

Disclaimers - Continued

- Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families: Go to: Learn About Intel® Processor Numbers http://www.intel.com/products/processor_number
- All the performance data are collected from our internal testing. Some results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

BACKUP

Configurations

Hardware	Testcase	CPU	Nodes	CPU Cores	Threads per Core	Sockets	Memory	Disks	Network
	#A	Intel Xeon WSM L5640 @ 2.27G	1+4	6	2	2	48GB	1TB * 4	1G
	#B	Intel Xeon L5640 @ 2.27 G	1+4	6	2	2	96GB	1TB * 11	1G
	#C	Intel Xeon E5-2660 @ 2.2G	1+4	8	2	2	192GB	1TB * 4	1G
	#D	Intel Xeon WSM L5640 @ 2.27G	1+4	6	2	2	48GB	1TB * 12	1G

Software		System	Kernel	File System	Spark stack version	Hadoop version	JDK version	Scala version
	#A	Ubuntu11.04	2.6.38-8-server.X86_64	Ext4	0.8.0-SNAPSHOT (Spark); internal dev branch (GraphX)	1.0.4	1.7.0_55-b13	2.9.3
	#B	RHEL6.0(Santiago)	2.6.32-71.el6.x86_64	Ext4	0.9.1(Spark); 0.9.1 Snapshort(Shark); 0.5.0-SNAPSHOT(Tachyon)	CDH 4.3.0	1.7.0_04-b20	2.10.3
	#C	RHEL6.2	2.6.32-220.el6.x86_64	Ext4	1.0.0 (Spark)	1.0.4	1.7.0_55-b13	2.10.4
	#D	Ubuntu12.04	2.6.38-8-server.X86_64	Ext4	1.1.0-SNAPSHOT	1.0.4	1.7.0_04	2.10.4

intel®