Progress on Efficient Integration of Lustre* and Hadoop/YARN

AUBURN UNIVERSITY

Weikuan Yu

intel

Omkar Kulkarni
Bryon Neitzel

Robin Goldstone

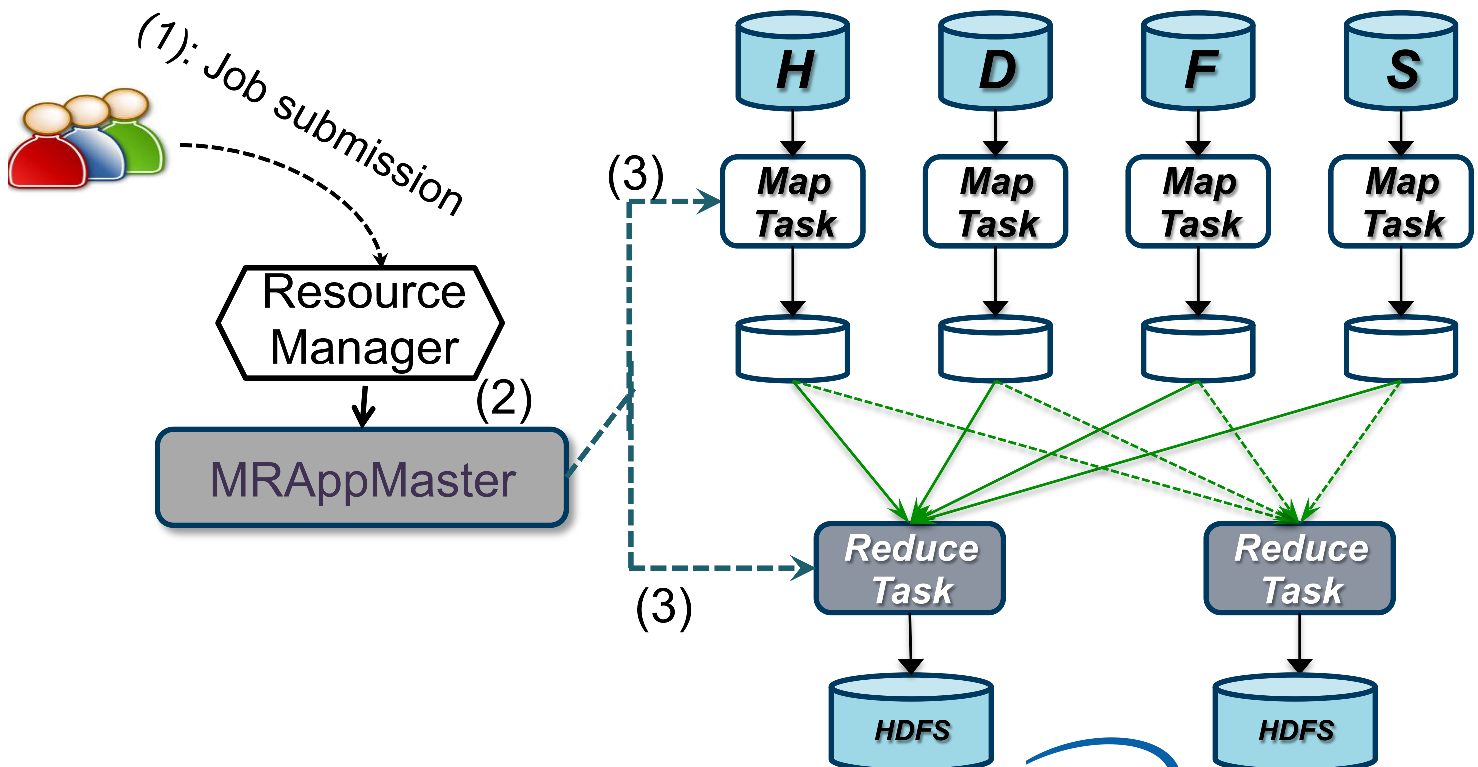
Lawrence Livermore National Laboratory

MapReduce

- A simple data processing model to process big data
- Designed for commodity off-the-shelf hardware components.
- Strong merits for big data analytics
  - **Scalability**: increase throughput by increasing # of nodes
  - **Fault-tolerance** (quick and low cost recovery of the failures of tasks)
- YARN, the next generation of Hadoop MapReduce Implementation

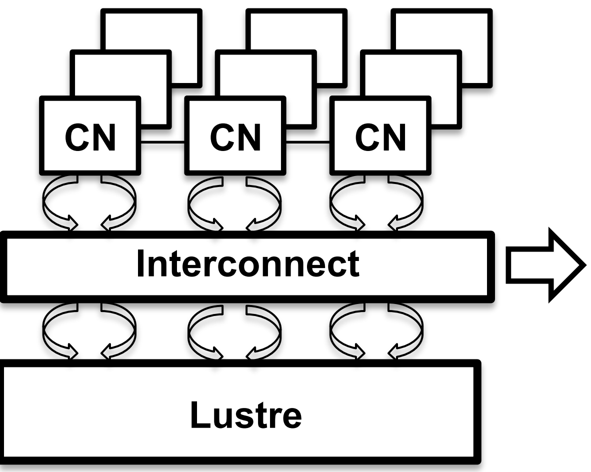
High-Level Overview of YARN

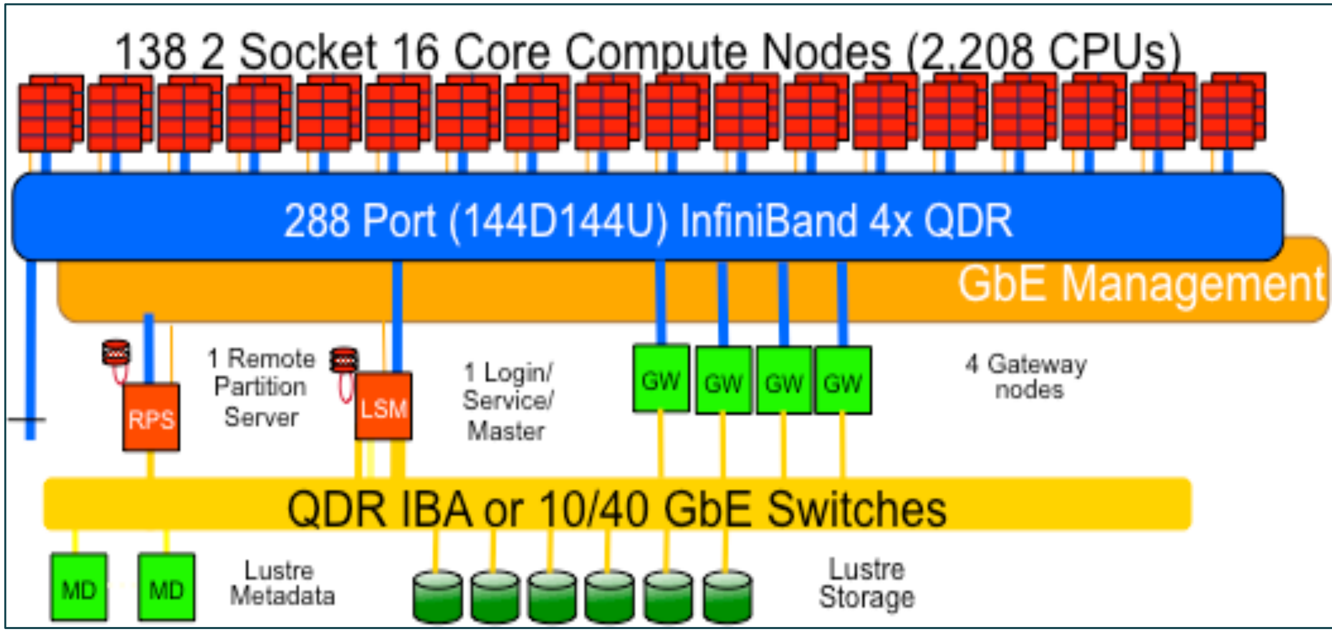
- Consists of HDFS and MapReduce frameworks.
- Exposes ***map*** and ***reduce*** interfaces.
    - ❖ ResourceManager and NodeManagers
    - ❖ MRAppMaster, MapTask, and ReduceTask.



Supercomputers and Lustre*

- Lustre popularly deployed on supercomputers
- A vast number of computer nodes (CN) for computation
- A parallel pool of back-end storage servers, composed a large pool of storage nodes



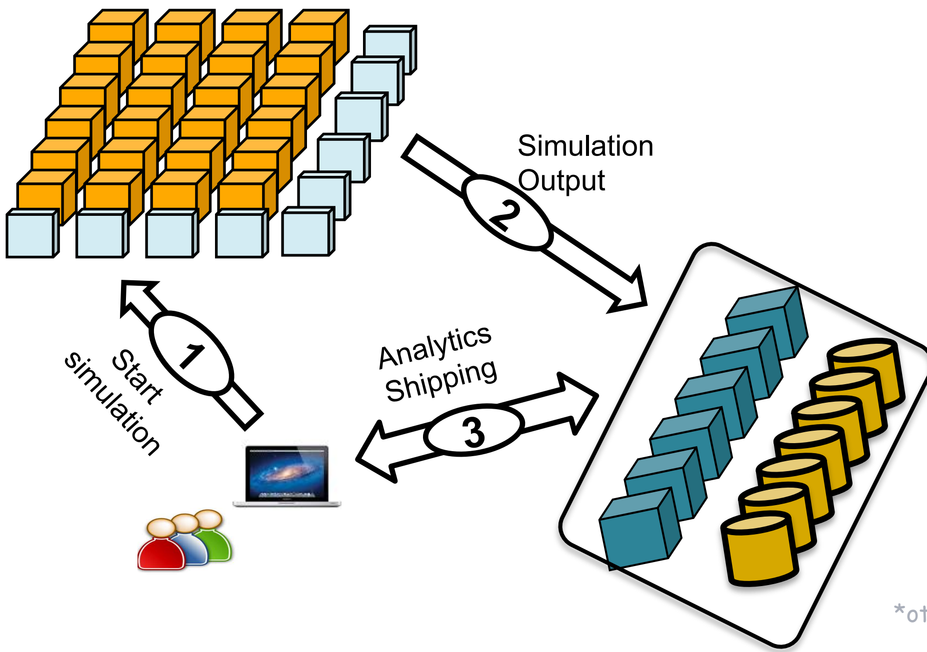


Lustre for MapReduce-based Analytics?

- Desire
  - Integration of Lustre as a storage solution
  - Understand the requirements of MapReduce on data organization, task placement, and data movement, and their implications to Lustre
- Approach:
  - Mitigate the impact of centralized data store at Lustre
  - Reduce repetitive data movement from computer nodes and storage nodes
  - Cater to the preference of task scheduling and data locality

Overarching Goal

- Enable **analytics shipping** on Lustre* storage servers
  - Users ship their analytics jobs to SNs on-demand
- Retain the default I/O model for scientific applications, storing data to Lustre
- Enable *in-situ* analytics at the storage nodes



*other names and brands may be claimed by others

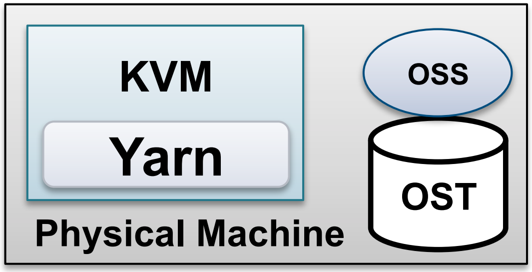
Technical Objectives

- Segregate analytics and storage functionalities within the same storage nodes
  - Mitigate interference between YARN and Lustre*
- Develop a coordinated data placement and task scheduling between Lustre and YARN
  - Enable and exploit data and task locality
- Improve Intermediate Data Organization for Efficient Shuffling on Lustre

*other names and brands may be claimed by others

YARN and Lustre* Integration with Performance Segregation

- Leverage KVM to create VM (virtual machine) instances on SNs
- Create Lustre storage servers on the physical machines (PMs)
- Run YARN programs and Lustre clients on the VMs
- Placement of YARN Intermediate data
  - On Lustre or local disks?

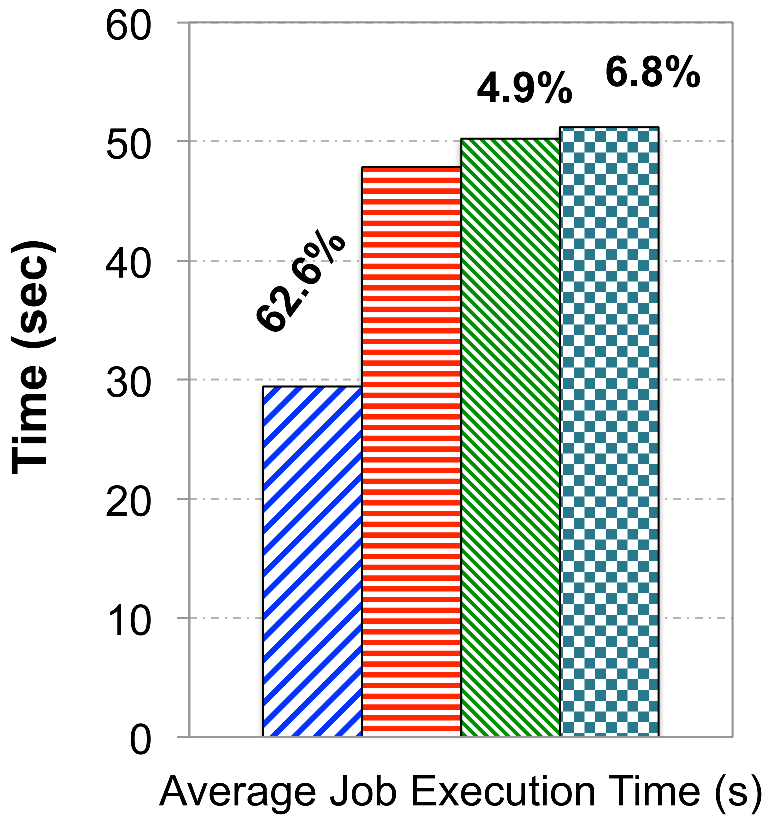


A Total of 8 OSTs

*other names and brands may be claimed by others

Running Hadoop/YARN on KVM

- 50 TeraSort jobs, 1GB input each. One job submitted every 3 seconds,
- There is a huge overhead caused by running YARN on KVM.
- Running IOR on 6 other machines. The impact is not very significant.



KVM configuration on each node:

- 4 cores, 6 GB memory.
- Using a different hard drive for Lustre*.
- Using a different network port

*other names and brands may be claimed by others

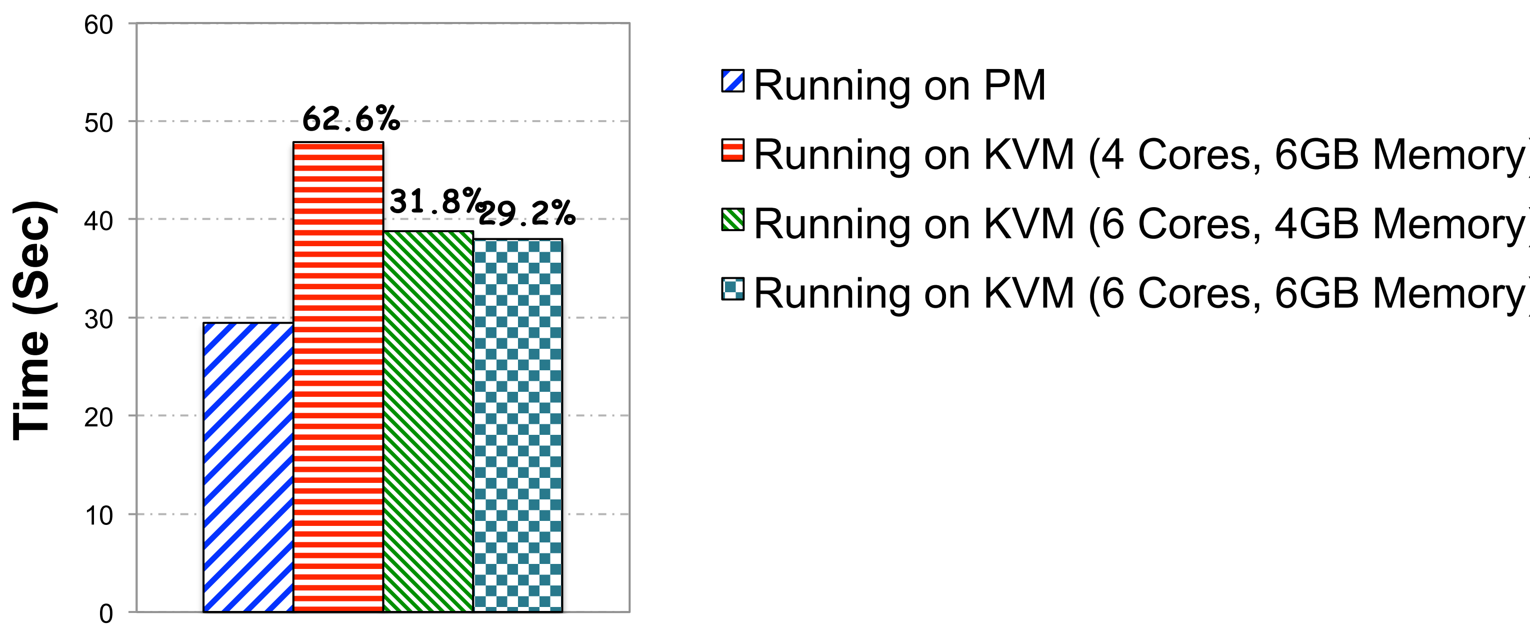
AUBURN UNIVERSITY

Lawrence Livermore National Laboratory

intel

KVM Overhead

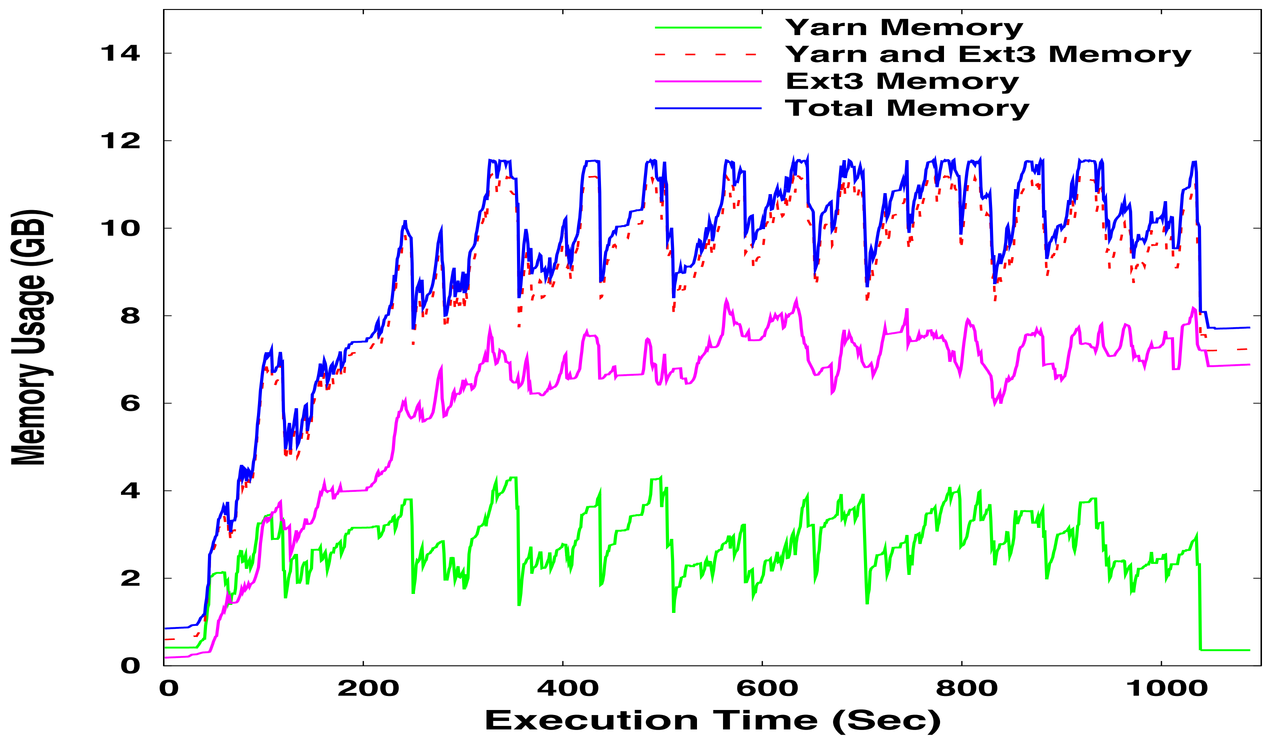
- 4 cores are not enough for YARN jobs
- 6 cores help improve the performance of YARN
- Increasing memory size from 4GB to 6GB has little effects when number of cores is the bottleneck



Average Job Excution Time (s)

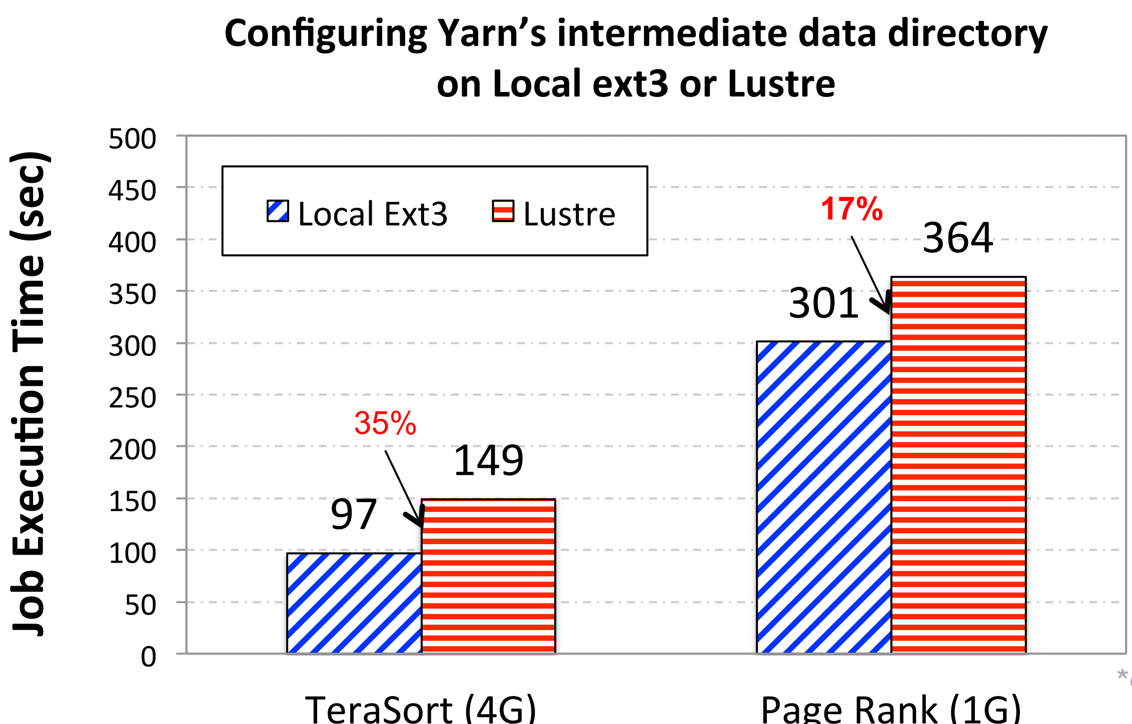
YARN Memory Utilization

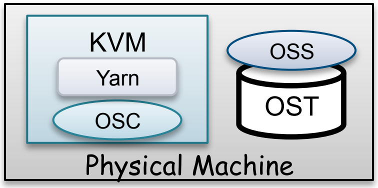
- Running Yarn on the physical machines alone.
- NodeManager is given 8GB memory, 1GB per container, 1GB heap per task.
- HDFS with local ext3 disks. Intensive writes to HDFS (via local ext3)



Intermediate Data on Local Disk or Lustre*

- Place intermediate data on local ext3 file system or Lustre, which is mapped to KVM (yarn.nodemanager.local-dirs).
- Yarn and Lustre Clients are placed on the KVM, OSS/OST on the Physical Machine
- Terasort (4G) and PageRank (1G) benchmarks have been measured





8 OSS/OST + 8 OSC

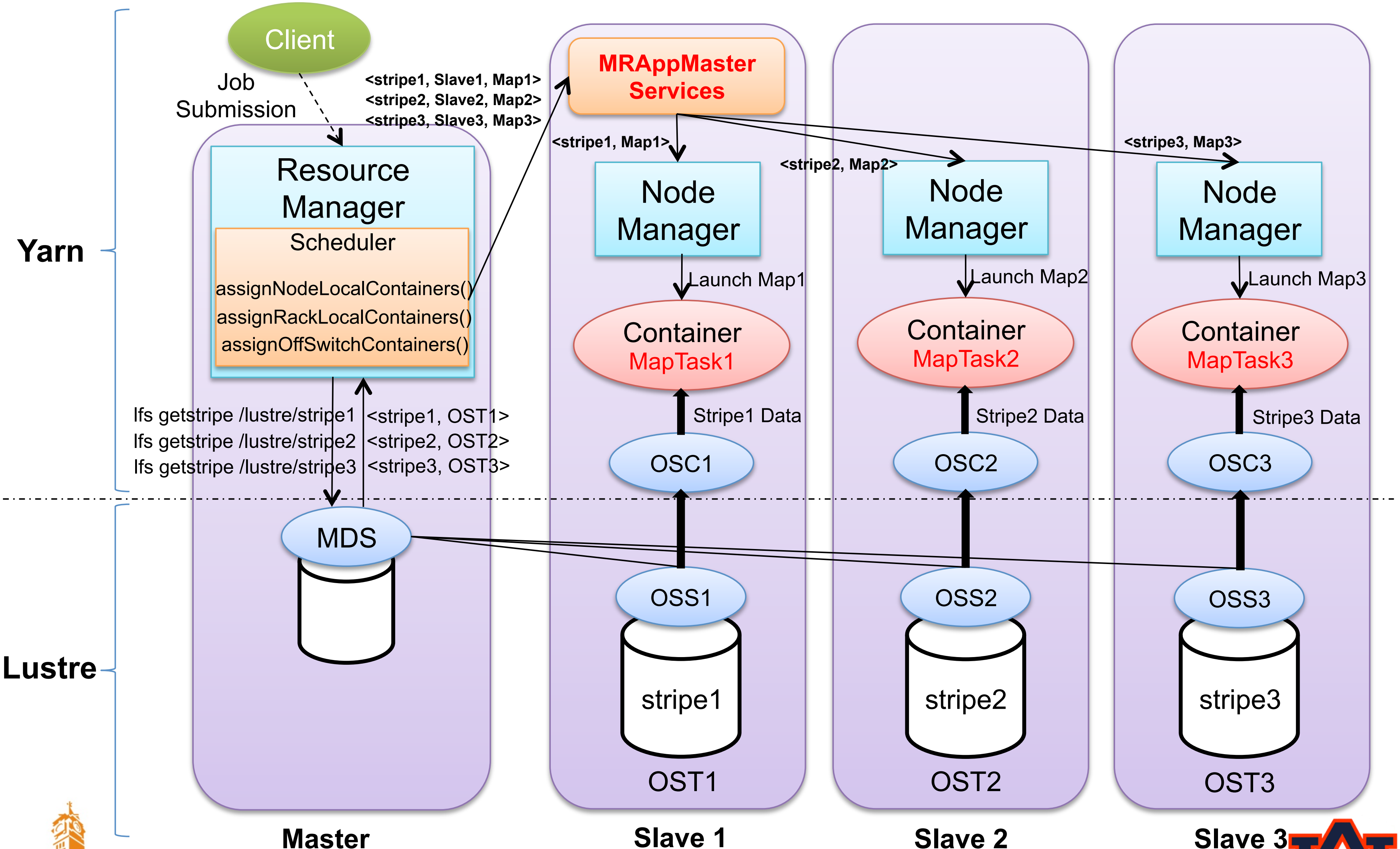
*other names and brands may be claimed by others

AUBURN UNIVERSITY

Lawrence Livermore National Laboratory

intel

Data Locality for YARN on Lustre*



AUBURN UNIVERSITY

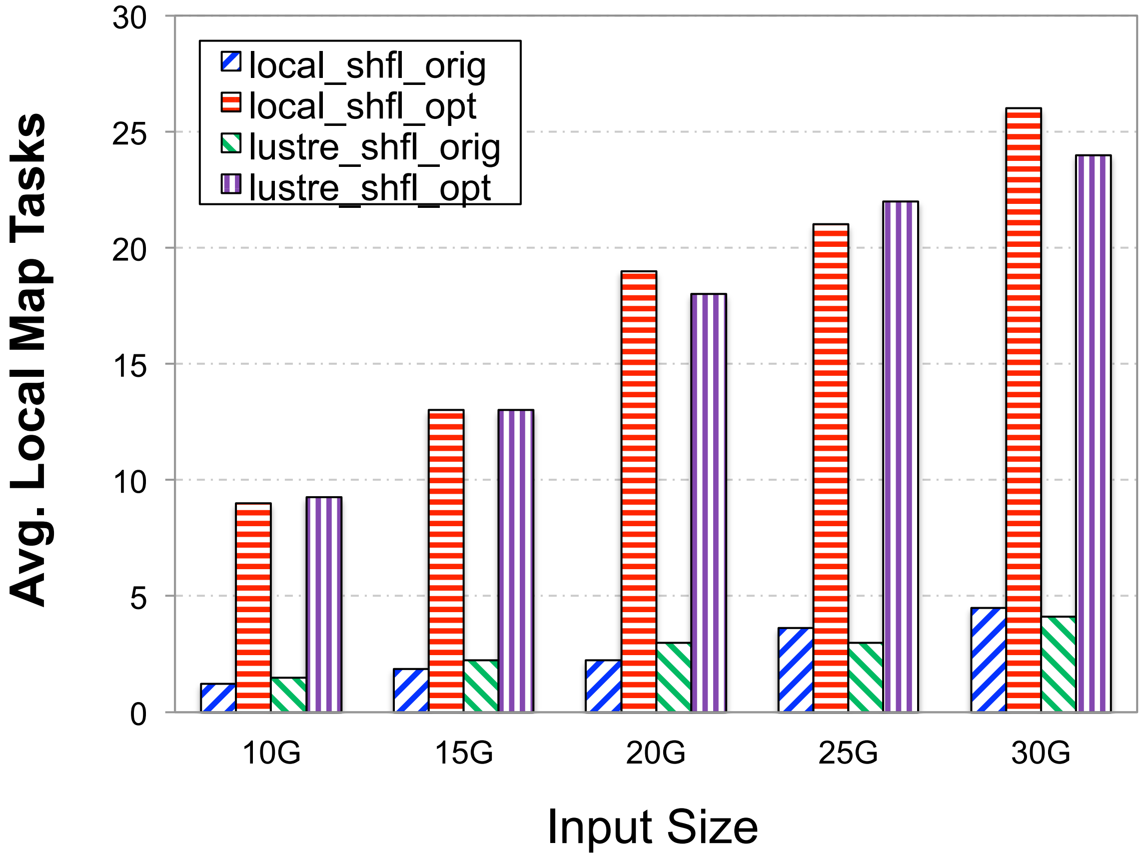
*other names and brands may be claimed by others

Background of TeraSort Test

- Four cases being compared
  - Intermediate Data on Lustre* or Local disks
  - Scheduling Map tasks with or without data locality
  - lustre_shfl_opt: (on lustre, with locality)
  - lustre_shfl_orig: (on lustre, without locality)
  - local_shfl_opt: (on local disks, with locality)
  - local_shfl_orig: (on local disks, without locality)
- Test environments

  -- Lustre 2.5 with dataset from 10GB to 30GB and 128MB stripe size and block size

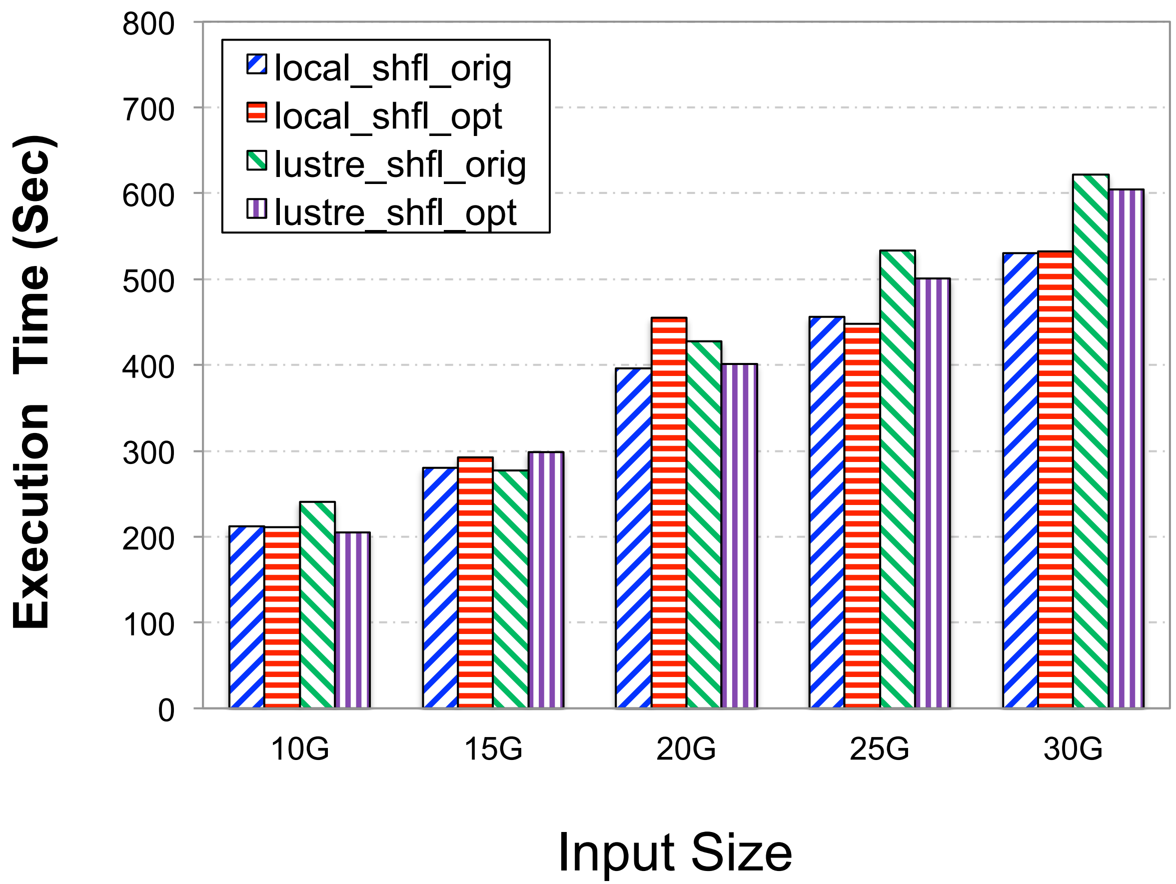
Average Number of Local Map Tasks



- local_shfl_opt and lustre_shfl_opt achieve high locality
- The other two have low locality.

AUBURN UNIVERSITY

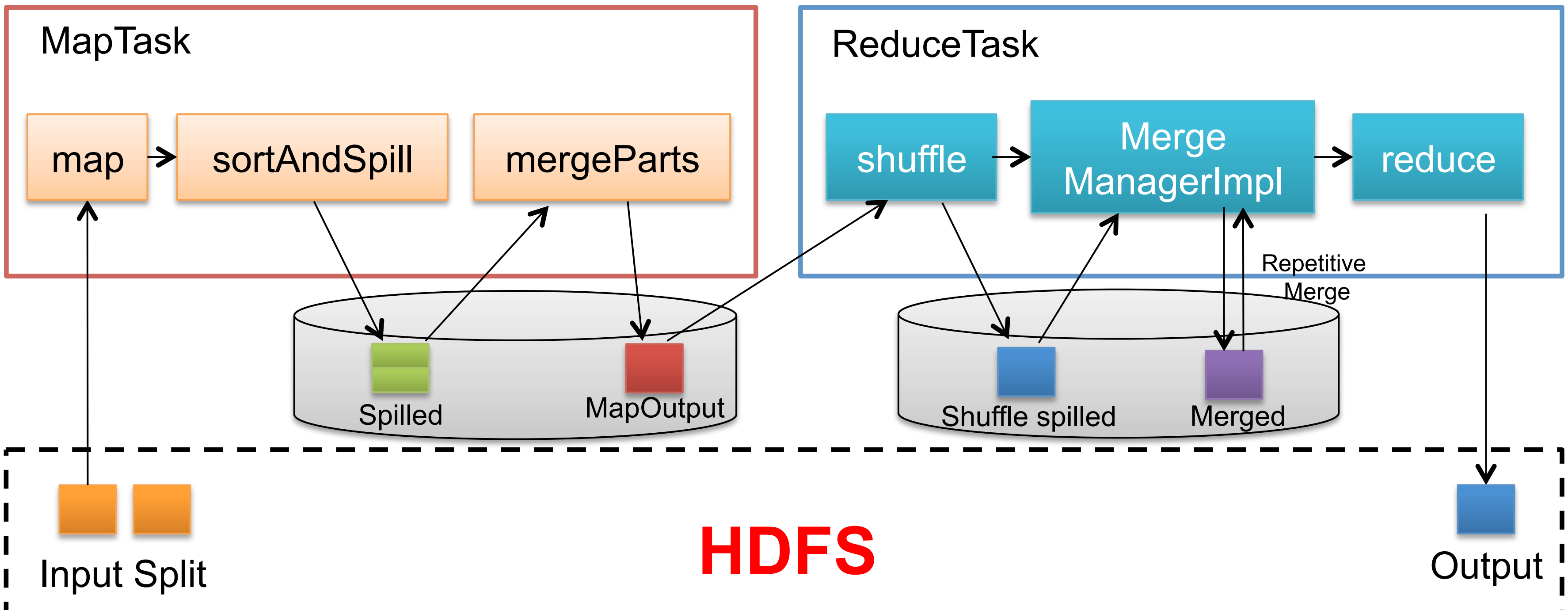
Terasort under Lustre* 2.5



- On average, local_shfl_orig has best performance
- lustre_shfl_opt is in the middle of best case and worst case;

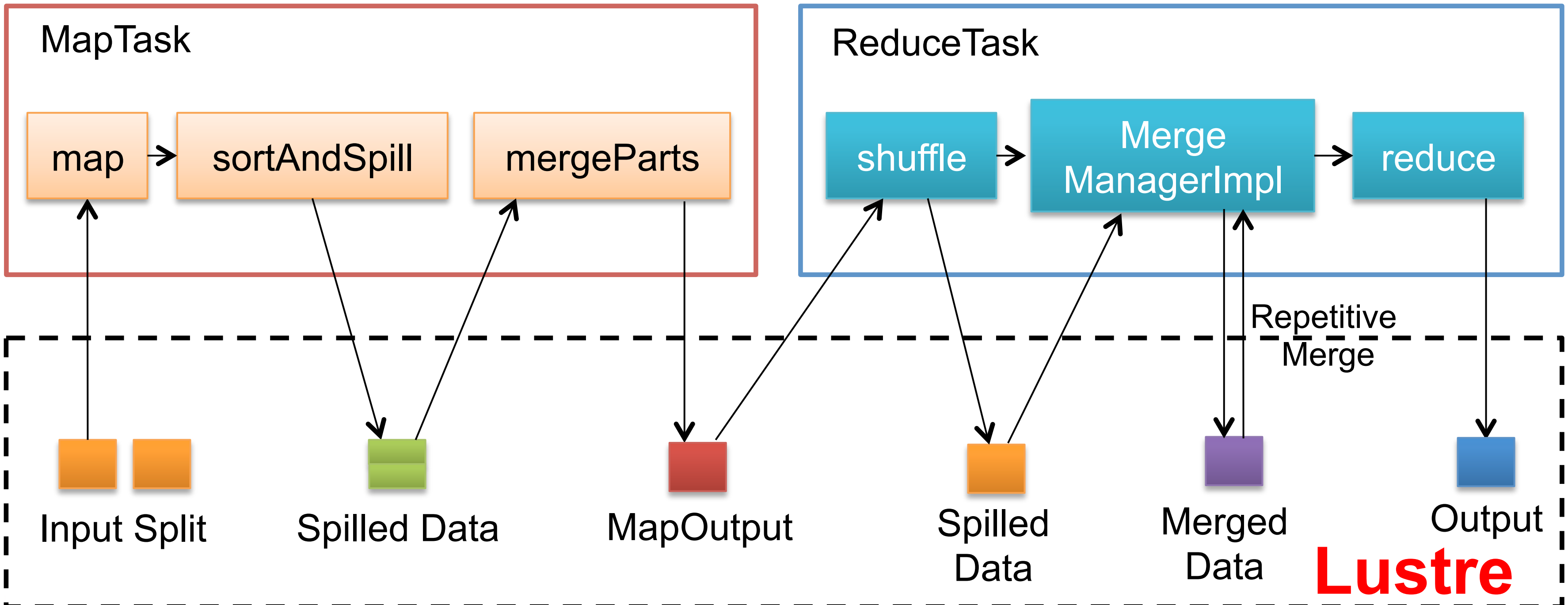
Data Flow in Original YARN over HDFS

- This figure shows all of the Disk I/O in original Hadoop
- Map Task: **Input Split**, **Spilled Data**, **MapOutput**
- Reduce Task: **Shuffled Data**, **Merged Data, Output Data**



Data Flow in YARN over Lustre*

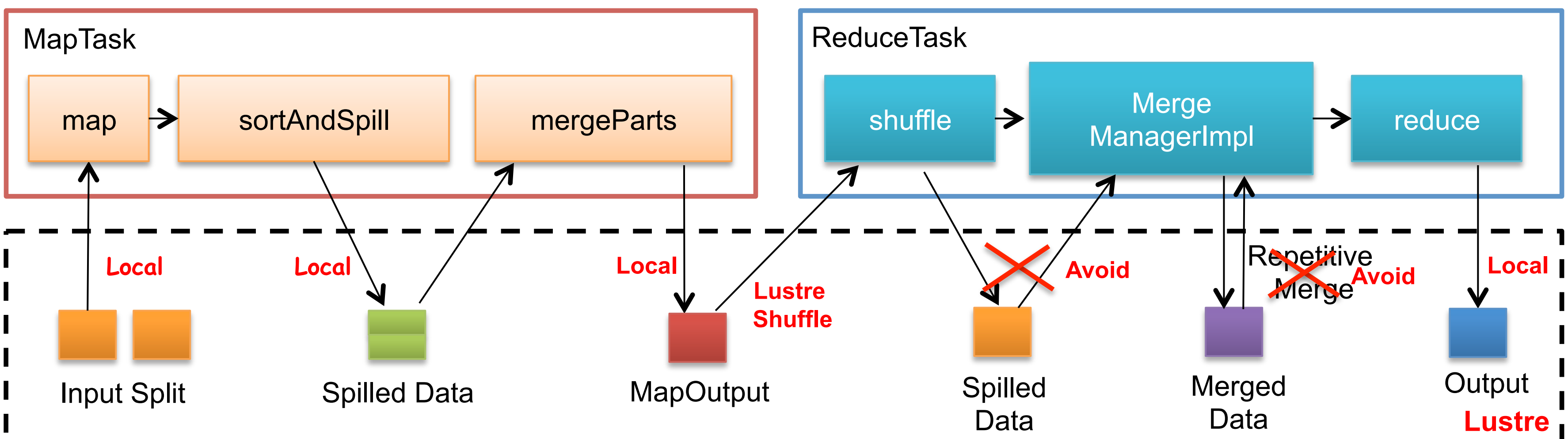
- This figure shows all of the Disk I/O of YARN over Lustre
- Avoid as much Disk I/O as possible
- Speed up Reading Input data and Writing Output data



*other names and brands may be claimed by others

New Implementation Design Review

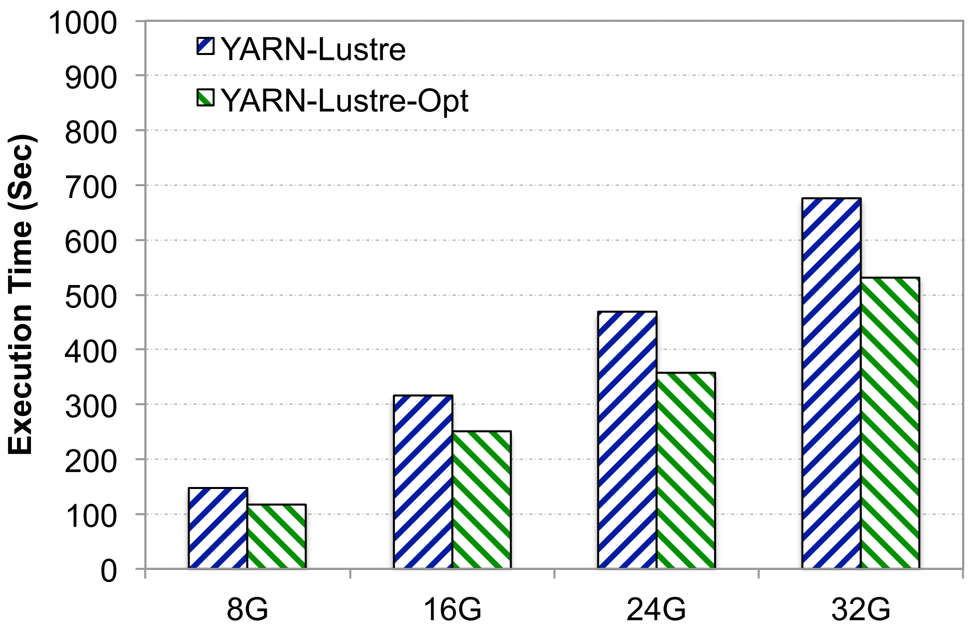
- Improve I/O performance
  - Read/Write from/to local OST
  - Avoid unnecessary shuffle spill and repetitive merge
  - After all MapOutput has been written, launch reduce task to read data
    - Avoid Lustre* write/read lock issues?
    - Reduce Lustre write/read contention?
- Reduce network contention
  - Most of data is written/read from local OST through virtio bridged network
  - Reserve more network bandwidth for Lustre Shuffle



*other names and brands may be claimed by others

Evaluation Results

- SATA Disk for OST, 10G Networking, Lustre* 2.5
- Running Terasort Benchmark, 1 master node, 8 slave nodes
- Optimized YARN performs on 21% better than the original YARN on average



*other names and brands may be claimed by others

Summary

- Explore the design of an Analytics Shipping Framework by integrating Lustre* and YARN
- Provided End-to-End optimizations on data organization, movement and task scheduling for efficient integration of Lustre and YARN
- Demonstrated its performance benefits to analytics applications

Acknowledgment

- Contributions from Cong Xu, Yandong Wang, and Huansong Fu.
- Awards to Auburn University from Intel, Alabama Department of Commerce and the US National Science Foundation.
- This work was also performed under the auspices of the US Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-647813).