Progress on Efficient Integration of Lustre* and Hadoop/YARN

Robin Goldstone

Weikuan Yu

Omkar Kulkarni
Bryon Neitzel

* Some name and brands may be claimed as the property of others.
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 of 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

*other names and brands may be claimed by others
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.

![Average Job Execution Time (s)]

- Running Alone on PM: 6.8%
- Running Alone on KVM: 4.9%
- Running with IOR - POSIX: 62.6%
- Running with IOR - MPIIO: 4.9%

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
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

![Chart showing comparison of average job execution time between PM and KVM configurations. The chart indicates that running on KVM with 6 cores and 6GB memory has the lowest average job execution time at 29.2%, followed by running on KVM with 4 cores and 6GB memory at 31.8%, and running on KVM with 4 cores and 6GB memory has the highest average job execution time at 62.6%.]
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.

![Chart showing job execution time for TeraSort and PageRank benchmarks]

*other names and brands may be claimed by others
Data Locality for YARN on Lustre*

resource manager

schedule

assignNodeLocalContainers()
assignRackLocalContainers()
assignOffSwitchContainers()

ResourceManager

Scheduler

Client

Job Submission

Container

MapTask1

MRAppMaster

Services

Node

Manager

Container

MapTask2

Node

Manager

Container

MapTask3

Node

Manager

Master

Slave 1

Slave 2

Slave 3

<stripe1, Slave1, Map1>

<stripe2, Slave2, Map2>

<stripe3, Slave3, Map3>

lfs

gestripe

/lustre/stripe1

<stripe1, OST1>

<stripe2, OST2>

<stripe3, OST3>

<stripe1, Map1>

<stripe2, Map2>

<stripe3, Map3>

Launch Map1

Launch Map2

Launch Map3

Stripe1 Data

Stripe2 Data

Stripe3 Data

<stripe2, Slave2, Map2>

<stripe3, Slave3, Map3>

lfs

gestripe

/lustre/stripe2

lfs

gestripe

/lustre/stripe3

*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

*other names and brands may be claimed by others
Average Number of Local Map Tasks

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

*other names and brands may be claimed by others
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**

MapTask:
- **map**
- **sortAndSpill**
- **mergeParts**

ReduceTask:
- **shuffle**
- **MergeManagerImpl**
- **reduce**

HDFS:
- **Input Split**
- **Spilled**
- **MapOutput**
- **Shuffle spilled**
- **Merged**
- **Output**
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

Slide --- 18
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

*other names and brands may be claimed by others
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).