Hadoop* MapReduce over Lustre*
High Performance Data Division
Omkar Kulkarni
April 16, 2013

With additions for CRIS conference
By Dan Ferber, Intel
May 9, 2014

* Other names and brands may be claimed as the property of others.
Legal Disclaimers:

This document contains information on products in the design phase of development.

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel's current plan of record product roadmaps.

FTC Optimization Notice: Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Technical Collateral Disclaimer: 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.
Welcome to Minnesota
Intel Lustre*
Intel® Enterprise Edition for Lustre* Software

- Full open source core (Lustre)
- Simple GUI for install and management with central data collection
- Direct integration with storage HW and applications
- Global 24x7 commercial support
- Storage plug-in; deep vendor integration
- REST API for extensibility
- Hadoop* Adapter for shared simplified storage for Hadoop

Hadoop Adapter
Lustre storage for MapReduce applications

Intel® Manager for Lustre*
Software
Configure, Monitor, Troubleshoot, Manage

CLI

REST API
Extensibility

Management and Monitoring Service

Lustre File System
Full distribution of open source Lustre software

Storage Plug-in
Integration

Intel® value-added Software
Open Source Software

* Some names and brands may be claimed as the property of others
Collaboration with Cloudera for HPC Solutions

- Intel has developed:
  - **Hadoop Adapter for Lustre** (HAL) to enable Lustre as an alternative to HDFS
  - **HPC Adapter for MapReduce** (HAM) to enable Hadoop to use SLURM/MOAB and other schedulers as an alternative to YARN
- Supported several beta customers for HAL & HAM this year
- **Intel® Enterprise Edition of Lustre (IEEL) V2.0** (June GA target)
  - Lustre HAL adapter for Cloudera’s Hadoop
  - And then Lustre HAM adapter in subsequent IEEL version
- Intel and Cloudera will collaborate to enable HPC environments with Cloudera Hadoop and Lustre

* Some names and brands may be claimed as the property of others
Overview of Lustre* and the Lustre HAL Adapter

* Some names and brands may be claimed as the property of others
Agenda

- Hadoop Intro
- Why run Hadoop on Lustre?
- Optimizing Hadoop for Lustre
- Performance
- What’s next?
A Little Intro of Hadoop

- Open source MapReduce framework for data-intensive computing
- Simple programming model – two functions: Map and Reduce
  - Map: Transforms input into a list of key value pairs
    - Map(D) → List[Ki , Vi]
  - Reduce: Given a key and all associated values, produces result in the form of a list of values
    - Reduce(Ki , List[Vi]) → List[Vo]
- Parallelism hidden by framework
  - Highly scalable: can be applied to large datasets (Big Data) and run on commodity clusters
- Comes with its own user-space distributed file system (HDFS) based on the local storage of cluster nodes
A Little Intro of Hadoop (cont.)

- Framework handles most of the execution
- Splits input logically and feeds mappers
- Partitions and sorts map outputs (Collect)
- Transports map outputs to reducers (Shuffle)
- Merges output obtained from each mapper (Merge)
Why Hadoop with Lustre?

- HPC moving towards Exascale. Simulations will only get bigger
- Need tools to run analyses on resulting massive datasets
- Natural allies:
  - Hadoop is the most popular software stack for big data analytics
  - Lustre is the file system of choice for most HPC clusters
- Easier to manage a single storage platform
  - No data transfer overhead for staging inputs and extracting results
  - No need to partition storage into HPC (Lustre) and Analytics (HDFS)
- Also, HDFS expects nodes with locally attached disks, while most HPC clusters have diskless compute nodes with a separate storage cluster
How to make them cooperate?

- Hadoop uses pluggable extensions to work with different file system types
- Lustre is POSIX compliant:
  - Use Hadoop’s built-in LocalFileSystem class
  - Uses native file system support in Java
- Extend and override default behavior: LustreFileSystem
  - Defines new URL scheme for Lustre – lustre:///  
  - Controls Lustre striping info
  - Resolves absolute paths to user-defined directory
  - Leaves room for future enhancements
- Allow Hadoop to find it in config files
Sort, Shuffle & Merge

- M → Number of Maps, R → Number of Reduces
- Map output records (Key-Value pairs) organized into R partitions
- Partitions exist in memory. Records within a partition are sorted
- A background thread monitors the buffer, spills to disk if full
- Each spill generates a spill file and a corresponding index file
- Eventually, all spill files are merged (partition-wise) into a single file
- Final index is file created containing R index records
- Index Record = [Offset, Compressed Length, Original Length]
- A Servlet extracts partitions and streams to reducers over HTTP
- Reducer merges all M streams on disk or in memory before reducing
Sort, Shuffle & Merge (Cont.)

Mapper X
- Sort
- Map

TaskTracker
- Servlet

Reducer Y
- Merge
- Copy
- Reduce

Input Split X
- Partition 1
- Partition 2
- Partition R
- Output

Index
- Idx 1
- Idx 2
- Idx R

Merged Streams
- Map 1: Partition Y
- Map 2: Partition Y
- Map M: Partition Y

Output Part Y

HDFS
Optimized Shuffle for Lustre

- Why? Biggest (but inevitable) bottleneck – bad performance on Lustre!

- How? Shared File System → HTTP transport is redundant

- How would reducers access map outputs?
  - First Method: Let reducers read partitions from map outputs directly
    - But, index information still needed
  - Either, let reducers read index files, as well
    - Results in \((M*R)\) small (24 bytes/record) IO operations
  - Or, let Servlet convey index information to reducer
    - Advantage: Read entire index file at once, and cache it
    - Disadvantage: Seeking partition offsets + HTTP latency
  - Second Method: Let mappers put each partition in a separate file
    - Three birds with one stone: No index files, no disk seeks, no HTTP
Optimized Shuffle for Lustre (Cont.)

Mapper X
- Map
- Sort

Reducer Y
- Reduce
- Merge
- Merged Streams
- Output Part Y

Input Split X
- Map X:Partition 1
- Map 2:Partition Y
- Map X:Partition Y
- Map M:Partition Y

Merged Streams
- Map 1:Partition Y
- Map X:Partition Y
- Map X:Partition R

Lustre

Mapper X: Partition Y
Mapper M: Partition Y
Reducer Y: Partition R
Merge Streams
Performance Tests

- Standard Hadoop benchmarks were run on the Rosso cluster

- Configuration – Hadoop (Intel Distro v1.0.3):
  - 8 nodes, 2 SATA disks per node (used only for HDFS)
  - One with dual configuration, i.e. master and slave

- Configuration – Lustre (v2.3.0):
  - 4 OSS nodes, 4 SATA disks per node (OSTs)
  - 1 MDS, 4GB SSD MDT
  - All storage handled by Lustre, local disks not used
TestDFSIO Benchmark

- Tests the raw performance of a file system
- Write and read very large files (35G each) in parallel
- One mapper per file. Single reducer to collect stats
- Embarrassingly parallel, does not test shuffle & sort

Throughput

\[
\frac{\sum \text{filesize}}{\sum \text{time}} \text{ MB/s}
\]

More is better!

<table>
<thead>
<tr>
<th></th>
<th>Write</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lustre</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

More is better!
Terasort Benchmark

- Distributed sort: The primary Map-Reduce primitive
- Sort a 1 Billion records, i.e. approximately 100G
  - Record: Randomly generated 10 byte key + 90 bytes garbage data
- Terasort only supplies a custom partitioner for keys, the rest is just default map-reduce behavior.
- Block Size: 128M, Maps: 752 @ 4/node, Reduces: 16 @ 2/node

![Runtime Chart]

Lustre 10-15% Faster
Work in progress

- Planned so far
  - More exhaustive testing needed
  - Test at scale: Verify that large scale jobs don’t throttle MDS
  - Port to IDH 3.x (Hadoop 2.x): New architecture, More decoupled
  - Scenarios with other tools in the Hadoop Stack: Hive, HBase, etc.

- Further Work
  - Experiment with caching
  - Scheduling Enhancements
  - Exploiting Locality