Harp-DAAL for High Performance Big Data Computing

Large-scale data analytics is revolutionizing many business and scientific domains. Easy-to-use scalable parallel techniques are necessary to process big data and gain meaningful insights. We introduce a novel HPC-Cloud convergence framework named Harp-DAAL and demonstrate that the combination of Big Data (Hadoop) and HPC techniques can simultaneously achieve productivity and performance. Harp is a distributed Hadoop-based framework that orchestrates efficient node synchronization [1]. Harp uses Intel® Data Analytics Accelerator Library (DAAL) [2], for its highly optimized kernels on Intel® Xeon and Xeon Phi architectures. This way the high-level API of Big Data tools can be combined with intra-node fine-grained parallelism that is optimized for HPC platforms. We illustrate this framework in detail with K-means clustering, a computation-bounded algorithm used in image clustering. We also show the broad applicability of Harp-DAAL by discussing the performance of three other big data algorithms: Subgraph Counting by color coding, Matrix Factorization and Latent Dirichlet Allocation. They share issues such as load imbalance, irregular structure, and communication issues that create difficult challenges.

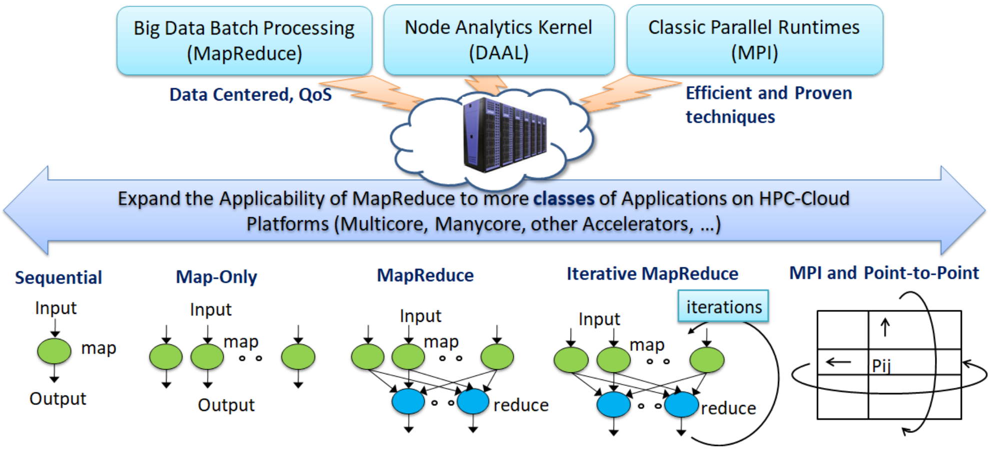

Big Data Batch Processing (MapReduce)
Node Analytics Kernel (DAAL)
Classic Parallel Runtimes (MPI)
Data Centered, QoS
Efficient and Proven techniques
Expand the Applicability of MapReduce to more classes of Applications on HPC-Cloud Platforms (Multicore, Manycore, other Accelerators, …)
Sequential
Input
map
Output
Map-Only
Input
map
Output
MapReduce
Input
map
reduce
Iterative MapReduce
iterations
Input
map
reduce
MPI and Point-to-Point
Pij

Figure 1 Cloud-HPC interoperable software for High Performance Big Data Analytics at Scale

The categories in Figure 1 illustrate a classification of data intensive computation into five computation models that map into five distinct system architectures. It starts with *Sequential*, followed by centralized batch architectures corresponding exactly to the three forms of MapReduce: *Map-Only*, *MapReduce* and *Iterative MapReduce*. Category five is the classic *MPI* model. Harp brings Hadoop users the benefits of supporting all 5 classes of data-intensive computation, from pleasingly parallel to machine learning and simulations. We have expanded the applicability of Hadoop (with Harp plugin) for more classes of Big Data applications, especially complex data analytics such as machine learning and graph. We re-design a modular software stack with native kernels (with DAAL) to effectively utilize scale-up servers for machine learning and data analytics applications. Harp-DAAL shows how simulations and Big Data can use common programming environments with a runtime based on a rich set of collectives and libraries.

How to interface Harp and DAAL?

Intel DAAL already provides API to their native C/C++ kernels by using high-level programming languages such as Java and Python. Since Harp is written in Java and extended from the Hadoop ecosystem, we choose Java to interface Harp and DAAL and optimize it particularly for applications with big intermediate data (e.g., machine learning model) of communication. In Harp-DAAL, data is stored in a hierarchical data structure named *Harp-Table*, which consists of tagged partitions. Each partition contains a partition ID (metadata) and a user-defined serializable Java object such as a primitive array. When doing communication, data is transferred among distributed cluster nodes via Harp collective communication operations. When doing local computation, data moves from *Harp-Table* (JVM heap memory) to DAAL native kernels (off-JVM heap memory). A data copy is unavoidable between Java object and C/C++ allocated memory space, and Figure 2 illustrates two approaches.

- *Direct Bulk Copy*: if a DAAL kernel allocates a continuous memory space for dense problems, Harp-DAAL will launch a bulk copy operation between a *Harp-Table* and a native memory address.
- *Multithreading Irregular Copy*: if a DAAL kernel involves an irregular and sparse data structure, which means that the data shall be stored in segments of non-consecutive memory space, Harp-DAAL provides a second copy operation by using Java/OpenMP threads, where each thread transfers a data segment concurrently.

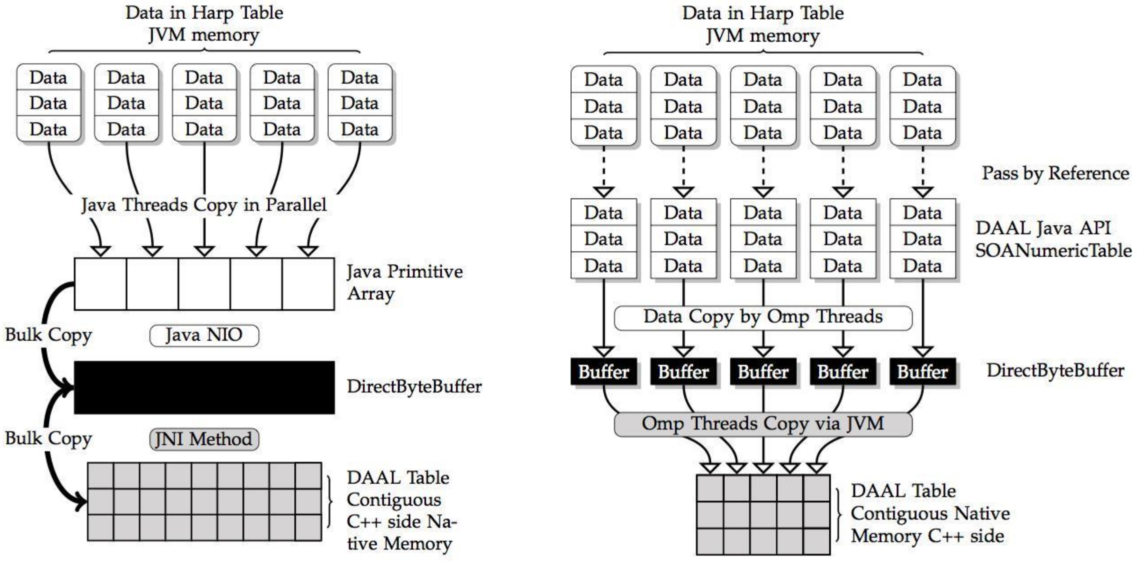


Figure 2. Direct Bulk Copy (left) vs. Multi-threading irregular Copy (right)

Applying Harp-DAAL to Data Analytics

K-means is a widely used and relatively simple clustering algorithm, which allows a clear example of how to use Harp-DAAL. K-means uses cluster centers to model the data and converges quickly via an iterative refinement approach. K-means runs on a large image dataset from Yahoo! Flickr, which includes 100 million images, each with 4096 dimensional deep features extracted by using a deep convolutional network model trained on ImageNet. Data preprocessing includes data format transformation and dimension reduction from 4096 to 128 by applying Principal Component Analysis (PCA). A detailed tutorial is available online at the website [3].

Case Study: Harp-DAAL Implementation of K-means

Harp-DAAL provides modular Java functions for developers to customize the K-means algorithm as well as tuning parameters for end users. The programming model consists of map functions linked by collectives. The K-means example takes seven steps as follows:

Step 1: Load training data (feature vectors) and model data (cluster centers)

Use the following function to load training data from HDFS.

```
// create a pointArray
List<double[]> pointArrays = LoadTrainingData();
```

Similarly, create a Harp table object *cenTable* and load centers from HDFS. Since centers are requested by all the mappers, the master mapper will load them and broadcast to all other mappers. Different center initialization methods can be supported in this fashion.

```
// create a table to hold cluster centers
Table<DoubleArray> cenTable = new Table<>(0, new DoubleArrPlus());

if (this.isMaster()) {
  createCenTable(cenTable);
  loadCentroids(cenTable);
}
// Bcast centers to other mappers
bcastCentroids(cenTable, this.getMasterID());
```

Step 2: Convert training data from Harp to DAAL

The training data loaded from HDFS is stored in the Java heap memory. To invoke the DAAL kernel, this step converts the data to a DAAL *NumericTable*, which includes allocating the native memory for the *NumericTable* and copying data from *pointArrays* to *trainingdata_daal*.

```
// convert training data from Harp to DAAL
NumericTable trainingdata_daal = convertTrainData(pointArrays);
```

Step 3: Create and set up DAAL K-means kernel

DAAL provides Java APIs to invoke native kernels for K-means on each node. It is called with a specification of the input training data object, the number of centers, and the number of threads to be used by the thread scheduler TBB and MKL libraries.

```
// create a DAAL K-means kernel object
DistributedStep1Local kmeansLocal = new DistributedStep1Local(daal_Context, Double.class,
Method.defaultDense, this.numCentroids);
// set up input training data
```

```
kmeansLocal.input.set(InputId.data, trainingdata_daal);
// specify the threads used in DAAL kernel
Environment.setNumberOfThreads(numThreads);
// create cenTable at daal side
NumericTable cenTable_daal = createCenTableDAAL();
```

Step 4: Convert center format from Harp to DAAL

The centers are stored in the Harp table *cenTable* for inter-process (mapper) communication. The centers are converted to DAAL format at each iteration.

```
//Convert center format from Harp to DAAL
convertCenTableHarpToDAAL(cenTable, cenTable_daal);
```

Step 5: Local computation by DAAL kernel

Call DAAL K-means kernels of local computation at each iteration.

```
// specify cluster centers to DAAL kernel
kmeansLocal.input.set(InputId.inputCentroids, cenTable_daal);
// first step of local computation by using DAAL kernels to get a partial result
PartialResult pres = kmeansLocal.compute();
```

Step 6: Inter-mapper communication

Harp-DAAL K-means uses an *AllReduce* computation model, where each mapper keeps a local copy of the whole model data (cluster centers). However, Harp provides different communication operations to synchronize model data among mappers.

- regroup & allgather (default)
- allreduce
- broadcast & reduce
- push & pull

In a *regroup & allgather* operation, it first combines the same center from different mappers and re-distributes them to mappers by a specified order. After averaging the centers, an allgather operation makes every mapper get a complete copy of the averaged centers.

```
comm_regroup_allgather(cenTable, pres);
```

In an *allreduce* operation, the centers are reduced and copied to every mapper. Then on each mapper an average operation is applied to the centers.

```
comm_allreduce(cenTable, pres);
```

At the end of each iteration, call *printTable* to check the clustering result.

```
//for iteration i, check the first ten centers and print out their first ten dimensions
```

```
printTable(cenTable, 10, 10, i);
```

Step 7: Release memory and store cluster centers

After all of the iterations, release the memory allocated for DAAL and for Harp table object. The center values are stored on HDFS as the output.

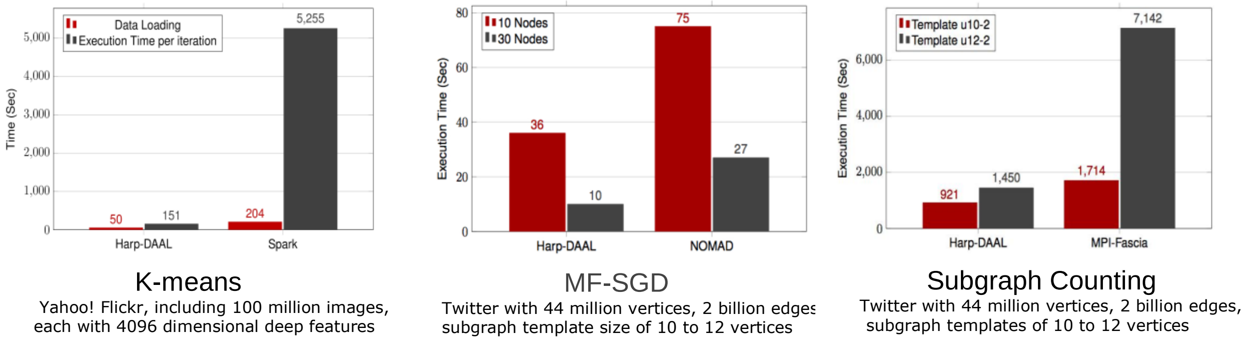
```
// free memory and record time
cenTable_daal.freeDataMemory();
trainingdata_daal.freeDataMemory();
// Write out the cluster centers
if (this.isMaster()) {
        KMUtil.storeCentroids(this.conf, this.cenDir,
        cenTable, this.cenVecSize, "output");
}
cenTable.release();
```

Experimental Performance Results

The performance for Harp-DAAL is illustrated by the results for four applications [4][5] with different algorithmic features:

- K-means: A dense clustering algorithm with regular memory access
- MF-SGD (Matrix Factorization for Stochastic Gradient Descent): A dense recommendation algorithm with irregular memory access and large model data
- Subgraph Counting: A sparse graph algorithm with irregular memory access
- LDA (Latent Dirichlet Allocation): A sparse topic modeling algorithm with large model data and irregular memory access

The testbed has two clusters, one with Xeon E5 2670 (Haswell) nodes and infiniband interconnect; the other with Xeon Phi 7250 (KNL) processors and Intel Omni-Path interconnect.



K-means

Yahoo! Flickr, including 100 million images, each with 4096 dimensional deep features

MF-SGD

Twitter with 44 million vertices, 2 billion edges subgraph template size of 10 to 12 vertices

Subgraph Counting

Twitter with 44 million vertices, 2 billion edges, subgraph templates of 10 to 12 vertices

Figure 3. Performance Comparison on three different important machine learning algorithms: K-means, MF-SGD and Subgraph counting.

In Figure 3, Harp-DAAL achieves around a 30x speedup over Spark for K-means on 30 KNL nodes using its highly vectorized kernels from the MKL library of DAAL. MF-SGD was run on up to 30 KNL nodes and achieved a 3x speedup over NOMAD, a state-of-the-art MPI C/C++

solution. The benefits come from Harp's rotation collective operation that accelerates the communication of the big model data in the recommender system.

Harp-DAAL subgraph counting on 16 Haswell nodes, has a 1.5x to 4x speedup over MPI-Fascia for large subtemplates with billion-edged Twitter graph data The performance improvement comes from node-level pipeline overlapping of computation and communication, and single node thread concurrency improved by neighbor list partitioning of graph vertex.

Figure 4 shows that Harp LDA achieves better convergence speed and speedup over other state-of-the-art MPI implementations such as LightLDA and NomadLDA [5]. This advantage comes from two optimizations for parallel efficiency: 1) Harp adopts the rotation computation model for inter-node communication of the latest model, and at the same time utilizes timer control to reduce the overhead of synchronization. 2) Further, at the intra-node level, a dynamic scheduling mechanism is developed to handle load imbalance.

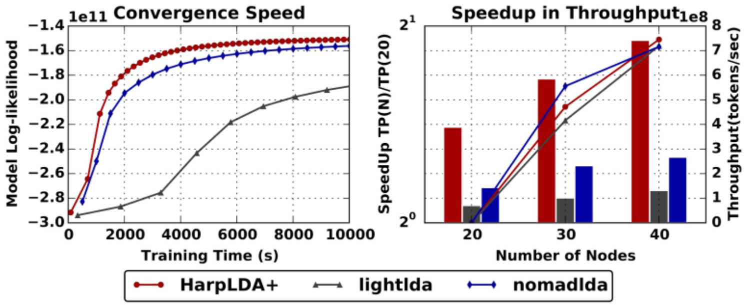


Figure 4. Performance of various LDA implementations on the Clueweb Dataset
(30 Billion Tokens, 5000 Topics)

The current Harp-DAAL system provides 13 distributed data analytics and machine learning algorithms leveraging the local computation kernels like K-means from Intel DAAL 2018 release. In addition, Harp-DAAL is developing its own data intensive kernels. This includes the large-scale subgraph counting algorithm given above, that can process social network Twitter graph with billions of edges and subtemplates of 10 vertices in 15 minutes. The Harp-DAAL framework and machine learning algorithms are publicly accessible [5] and users are encouraged to download the software, explore the tutorials, and apply Harp-DAAL to other data intensive applications.

References

[1] Harp code repository, https://github.com/DSC-SPIDAL/harp

[2] Intel® DAAL, https://github.com/intel/daal

[3] Harp-DAAL tutorial, https://github.com/DSC-SPIDAL/harp/tree/tutorial

[4] Langshi Chen et. al, *Benchmarking Harp-DAAL: High Performance Hadoop on KNL Clusters*, in the Proceedings of the 10th IEEE International Conference on Cloud Computing (IEEE Cloud 2017), June 25-30, 2017.

[5] Bo Peng et. al, *HarpLDA+: Optimizing Latent Dirichlet Allocation for Parallel Efficiency*, the IEEE Big Data 2017 conference, December 11-14, 2017.