Harp Tutorial

Bingjing Zhang

INDIANA UNIVERSITY BLOOMINGTON
SCHOOL OF INFORMATICS AND COMPUTING

Outline

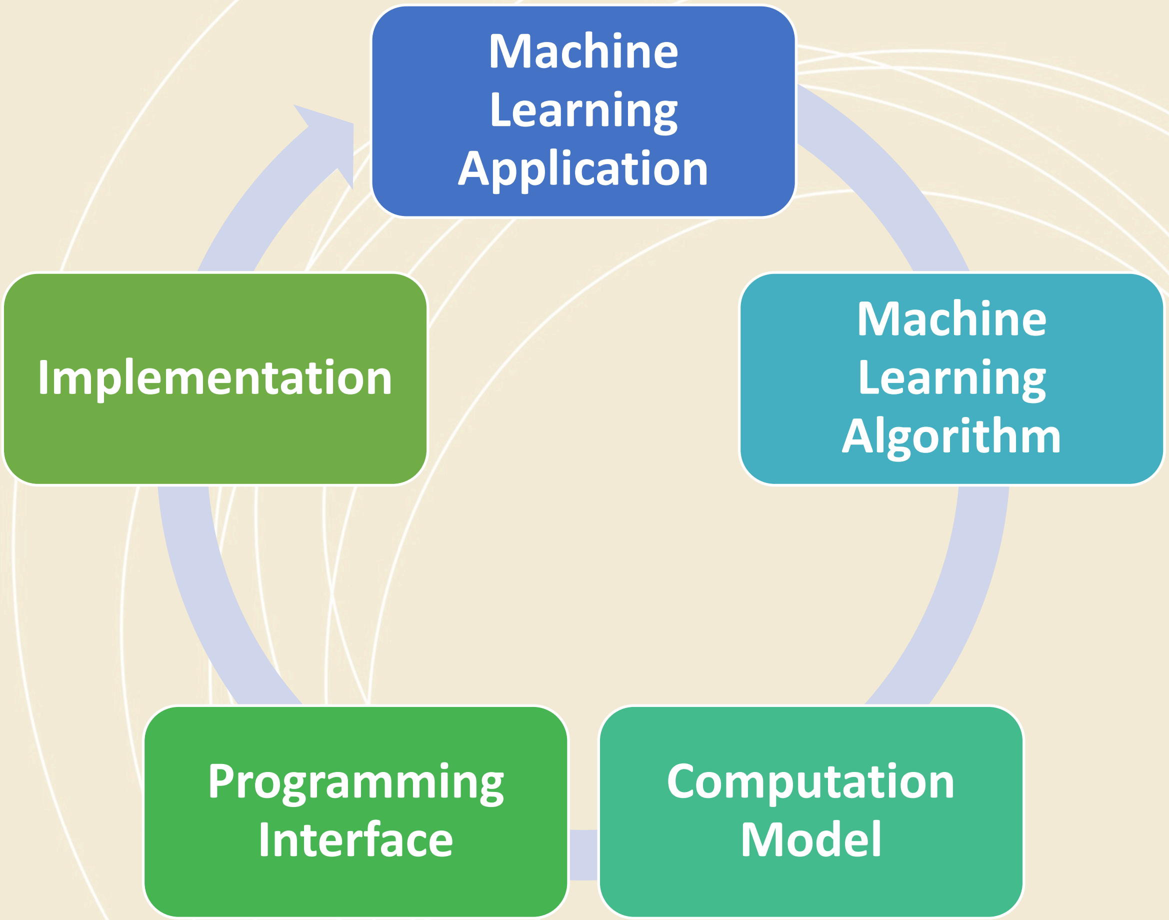
Concepts

Data Types

APIs

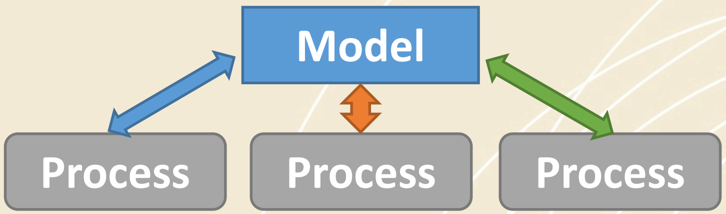
Application Examples

Parallelization of Machine Learning Applications



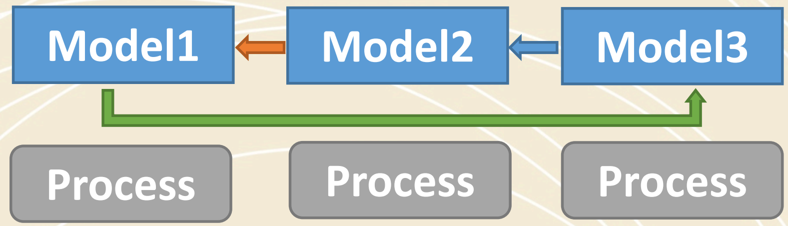
Inter-node Computation Models

(A)



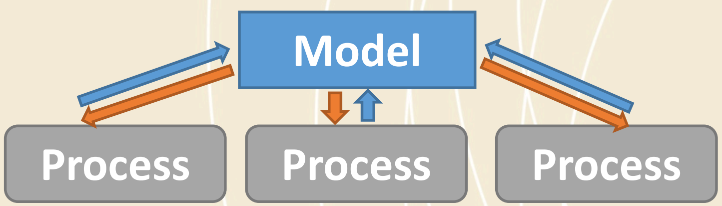
- Synchronized algorithm
- The latest model

(B)



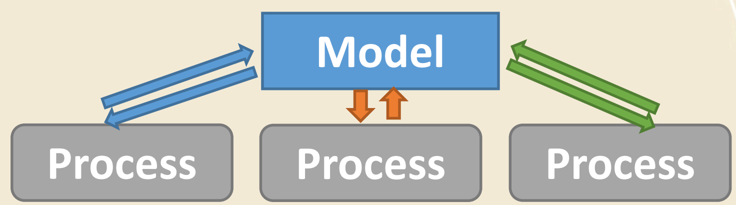
- Synchronized algorithm
- The latest model

(C)



- Synchronized algorithm
- The stale model

(D)



- Asynchronous algorithm
- The stale model

Inter-node Computation Models (Training Data Items Are Partitioned to Each Process)

Computation Model A

- Once a process trains a data item, it locks the related model parameters and prevents other processes from accessing them. When the related model parameters are updated, the process unlocks the parameters. Thus the model parameters used in local computation is always the latest.

Computation Model B

- Each process first takes a part of the shared model and performs training. Afterwards, the model is shifted between processes. Through model rotation, each model parameters are updated by one process at a time so that the model is consistent.

Computation Model C

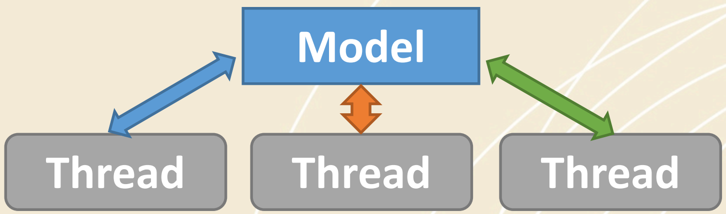
- Each process first fetches all the model parameters required by local computation. When the local computation is completed, modifications of the local model from all processes are gathered to update the model.

Computation Model D

- Each process independently fetches related model parameters, performs local computation, and returns model modifications. Unlike A, workers are allowed to fetch or update the same model parameters in parallel. In contrast to B and C, there is no synchronization barrier.

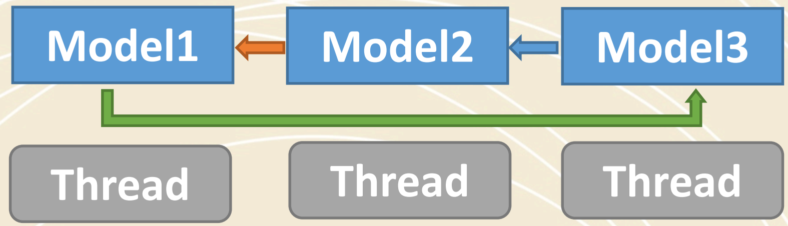
Intra-node Computation Models

(A)



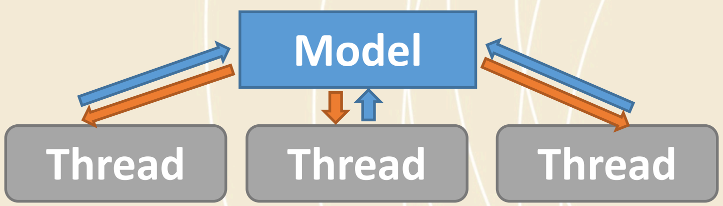
- Synchronized algorithm
- The latest model

(B)



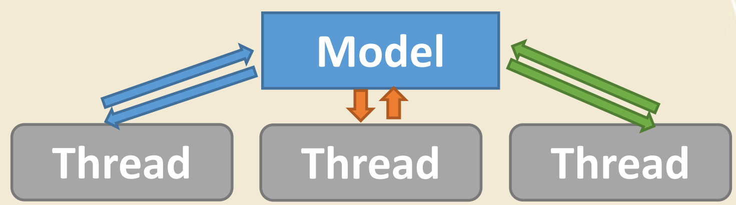
- Synchronized algorithm
- The latest model

(C)



- Synchronized algorithm
- The stale model

(D)



- Asynchronous algorithm
- The stale model

Intra-node Computation Models
(Training Data Items Are Scheduled to Each Thread)

Computation Model A

- Once a thread trains a data item, it locks the related model parameter and prevents other threads from accessing it. When the model parameter is updated, the thread unlocks the parameter. Thus the model parameters used per thread is always the latest.

Computation Model B

- Each thread first takes a part of the shared model and performs training. Afterwards, the model part is sent or scheduled to another thread. Model parameters in each model part are updated by one thread at a time so that the model is consistent.

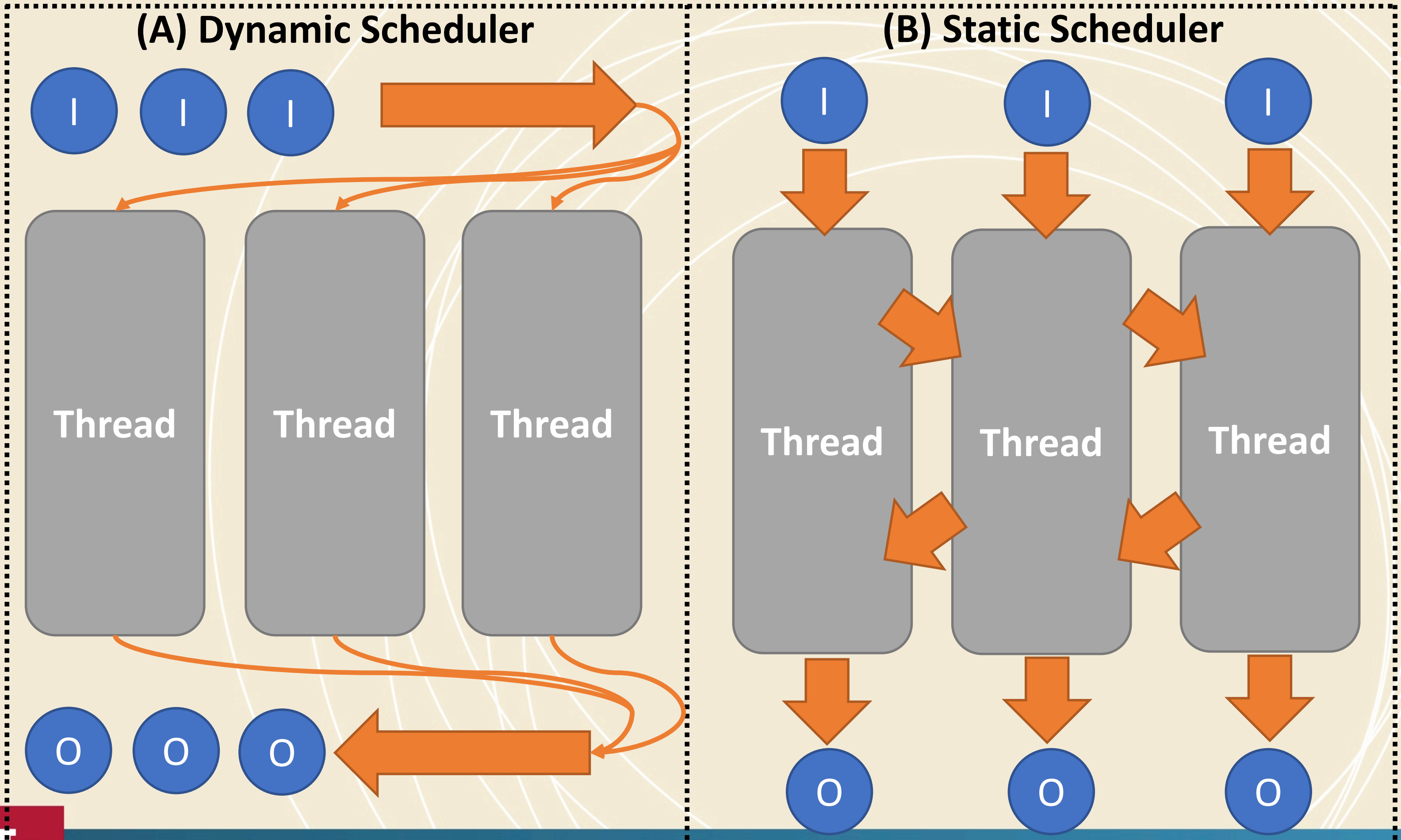
Computation Model C

- Each thread has a copy of the related model parameters. When the local computation is completed, modifications of the local model from all threads are combined to update the model.

Computation Model D

- Each thread independently reads the related model parameters, performs computation, and update the parameters. Unlike A, threads are allowed to read or write the same model parameter in parallel.

Schedule Training Data Partitions to Threads (only Data Partitions in Computation Model A, C, D; Data and/or Model Partitions in B)



Intra-node Schedulers

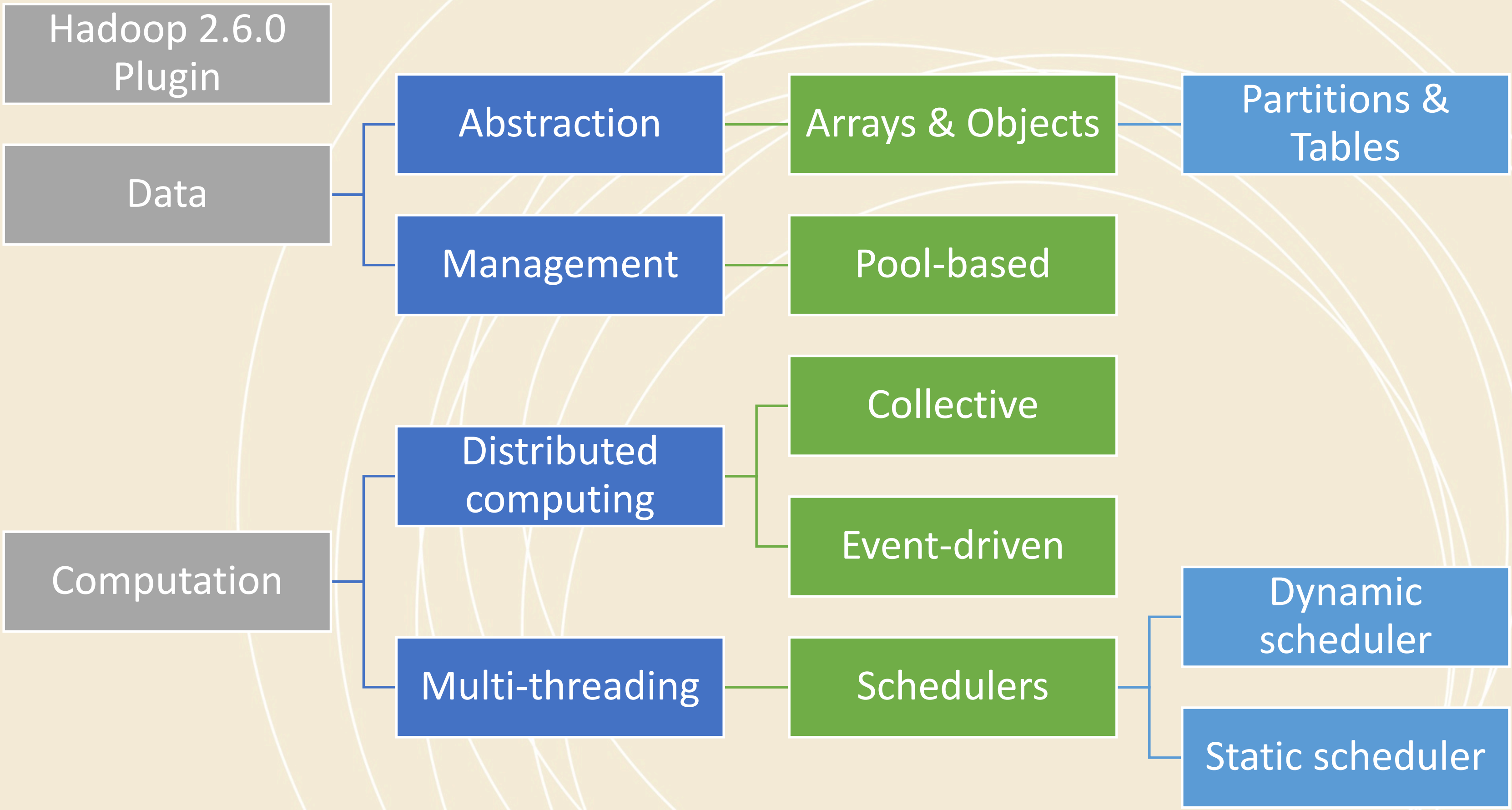
(A) Dynamic Scheduler

- All computation models can use this scheduler.
- All the inputs are submitted to one queue.
- Threads dynamically fetch inputs from the queue.
- The main thread can retrieve the outputs from the output queue.

(B) Static Scheduler

- All computation models can use this scheduler.
- Each thread has its own input queue and output queue.
- Each thread can submit inputs to another thread .
- The main thread can retrieve outputs from each task's output queue.

Harp Features



Data Types

Arrays & Objects

Primitive Arrays

- ByteArray, ShortArray, IntArray, FloatArray, LongArray, DoubleArray

Serializable Objects

- Writable

Partitions & Tables

Partition

- An array/object with partition ID

Table

- The container to organize partitions

Key-value Table

- Automatic partitioning based on keys

APIs

Scheduler

- DynamicScheduler
- StaticScheduler

Collective

- broadcast
- reduce
- allgather
- allreduce
- regroup
- pull
- push
- rotate

Event Driven

- getEvent
- waitEvent
- sendEvent

Harp K-means

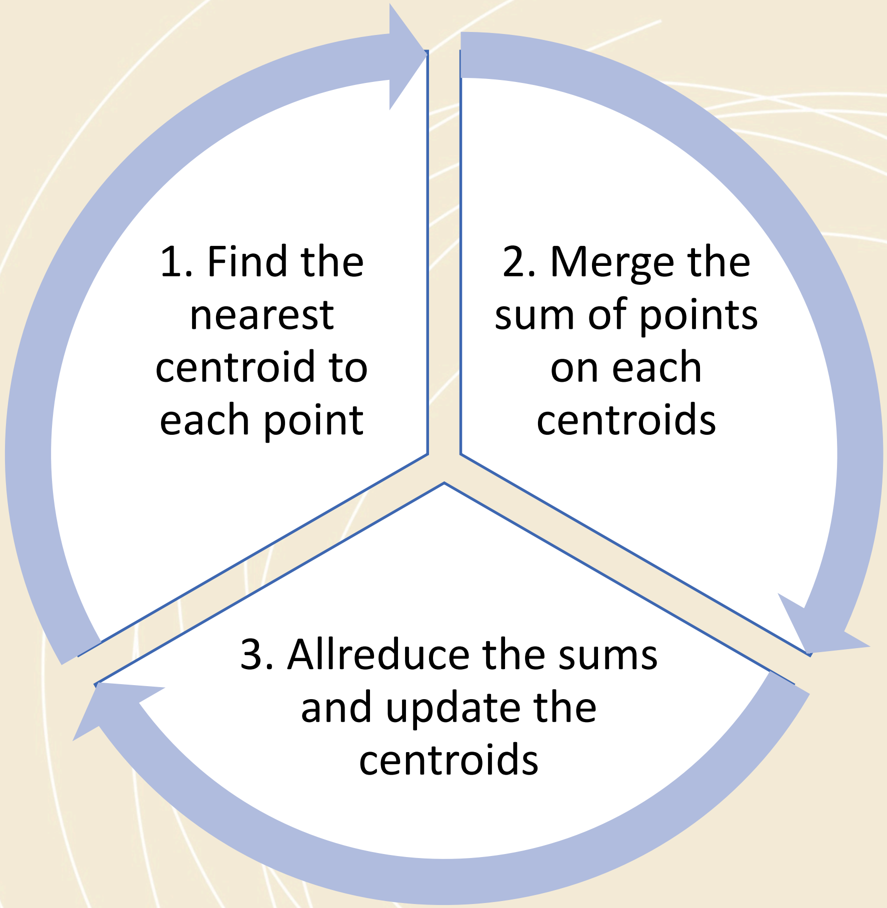
Model Allreduce

- Uses allreduce to sum all the local versions of intermediate centroids

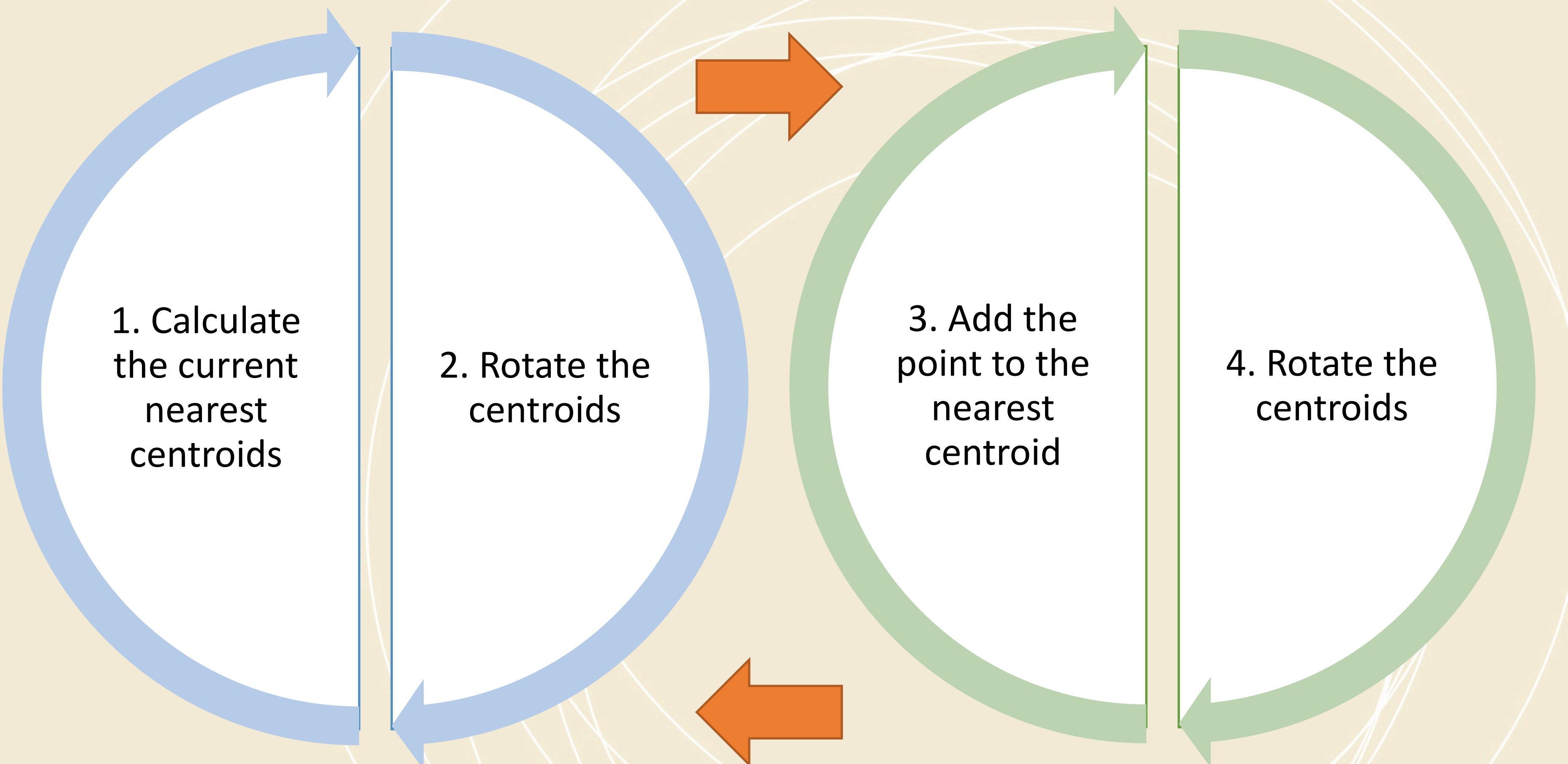
Model Rotation

- Use rotation to do partial centroid comparison/update each time

Model Allreduce



Model Rotation



mapCollective interface

protected void setup(Context context)

- The interface invoked before running the task, used for fetching job configurations to the Map task.

protected void mapCollective(KeyValReader reader, Context context)

- The main interface to process key-value pairs
- KeyValReader is used to read all the key-value pairs to the task

Partition & Table

```
Table<DoubleArray> table = new Table<>(0, new DoubleArrPlus());
for (int i = 0; i < numPartitions; i++) {
    DoubleArray array = DoubleArray.create(size, false);
    table.addPartition(new Partition<>(i, array));
}
```

- To create a table, an ID and a combiner is required.
- Table ID is user defined. Default ID is allowed.
- Combiner can combine partitions with the same ID in the table.
- A partition contains a partition ID and an object <? extends Simple>.

Array & Writable

Array

- **ByteArray**, **ShortArray**, **IntArray**, **FloatArray**, **LongArray** and **DoubleArray**
- **get –** get the array body
- **size –** get the size of array

Writable

- **getNumWriteBytes –** calculate the number of bytes to be serialized
- **write** & **read -** methods required to implement to direct the serialization / deserialization
- **clear** – how the object is cleaned before returning to the cache pool

Array & Writable

public static DoubleArray create(int size, boolean approximate)

- **create -** utilizes the pool based management to fetch an cached allocation. After using, the array/object can be **release**d back to the pool or **free**d to GC.
- **approximate** - indicates if the real array allocation size can be padded to a size in order to increase the chance for reuse.

broadcast

public <P extends Simple> boolean broadcast(String contextName, String operationName, Table<P> table, int bcastWorkerID, boolean useMSTBcast)

- **Simple** – the interface for **ByteArray**, **ShortArray**, **IntArray**, **FloatArray**, **LongArray** and **DoubleArray** and **Writable**
- **contextName** – user defined name to separate operations in different groups
- **operationName** – user defined name to separate operations
- **Table<P> table** – the data structure to broadcast/receive data
- **bcastWorkerID** – the worker to broadcast data
- **useMSTBcast** – default broadcast algorithm is pipelining, set this option to true to enable minimum spanning tree algorithm

reduce

public <P extends Simple> boolean reduce(String contextName, String operationName, Table<P> table, int reduceWorkerID)

- **contextName** - user defined name to separate operations in different groups
- **operationName** - user defined name to separate operations
- **table** - the data structure for reducing data
- **reduceWorkerID** - the worker ID to receive the reduced data

regroup

public <P extends Simple, PT extends Partitioner> boolean regroup(String contextName, String operationName, Table<P> table, PT partitioner)

- **contextName** – user defined name to separate operations in different groups
- **operationName** - user defined name to separate operations
- **table** - the data structure for regrouping data
- **partitioner** – tells which partition to go to which worker for regrouping, e.g. **new Partitioner(numWorkers)**

allgather

public <P extends Simple> boolean allgather(String contextName, String operationName, Table<P> table)

- **contextName** – user defined name to separate operations in different groups
- **operationName** - user defined name to separate operations
- **table** - the data structure for allgather data

allreduce

Normally…

- public <P extends Simple> boolean **allreduce**(String contextName, String operationName, Table<P> table)

An Alternative Way (if the dataset is large…)

- **regroup**("main", "regroup", table, new Partitioner(getNumWorkers()));
- **allgather**("main", "allgather", table);

push & pull

public <P extends Simple, PT extends Partitioner> boolean push(String contextName, String operationName, Table<P> localTable, Table<P> globalTable, PT partitioner)

- Send the partitions from localTable to globalTable based on the partition ID matching
- **localTable** - contains temporary local partitions
- **globalTable** - is viewed as a distributed dataset where each partition ID is unique across processes
- **partitioner** – if some local partitions is not shown in the globalTable, a partitioner can be used to decide where partitions with this partition ID go

public <P extends Simple> boolean pull(String contextName, String operationName, Table<P> localTable, Table<P> globalTable, boolean useBcast)

- Retrieve the partitions from globalTable to localTable based on partition ID matching
- **useBcast** – if broadcasting is used when a partition is required to send to all the processes.

rotate

rotate(String contextName, String operationName, Table<P> globalTable, Int2IntMap rotateMap)

- **contextName** - user defined name to separate operations in different groups
- **operationName** - user defined name to separate operations
- **globalTable** - the data structure for reducing data
- **rotateMap** – the mapping between source worker and target worker

DynamicScheduler

public class **DynamicScheduler<I, O, T extends Task<I, O>>** - class declaration

public **DynamicScheduler**(List<T> tasks) - constructor

public synchronized void **submit**(I input) – submit an input

public synchronized void **start**() – start the threads

public synchronized void **pause**() – pause the threads after the current inputs are processed

public synchronized void **pauseNow**() - pause the threads immediately

public synchronized void **cleanInputQueue**() – clean the input queue when the execution is paused or stopped

public synchronized void **stop**() – join the threads

public synchronized boolean **hasOutput**() – check if there is an output in the output queue

public synchronized O **waitForOutput**() – fetch an output from the output queue

StaticScheduler

public class **StaticScheduler<I, O, T extends Task<I, O>>** - class declaration

public **StaticScheduler**(List<T> tasks) - constructor

public synchronized void **submit**(int taskID, I input) – submit an input to a task

public synchronized void **start**() – start threads

public synchronized void **pause**() – pause the threads

public synchronized void **cleanInputQueue**() – clean the input queue

public synchronized void **stop**() – stop the threads

public O **waitForOutput**(int taskID) – wait for the output from a task

public boolean **hasOutput**(int taskID) – check if a task has an output

Adjust Memory Settings

yarn-site.xml

```
<property>
   <name>yarn.scheduler.minimum-allocation-mb</name>
   <value>512</value>
</property>

<property>
   <name>yarn.scheduler.maximum-allocation-mb</name>
   <value>2048</value>
</property>

<property>
   <name>yarn.nodemanager.resource.memory-mb</name>
   <value>4096</value>
</property>

<property>
   <name>yarn.nodemanager.vmem-check-enabled</name>
   <value>false</value>
</property>
```

Adjust Memory Settings cont’d

mapred-site.xml

```
<!– set the Map process memory size -->
<property>
   <name>mapreduce.map.collective.memory.mb</name>
   <value>512</value>
</property>

<property>
   <name>mapreduce.map.collective.java.opts</name>
   <value>-d64 -server -Xmx256m -Xms256m</value>
</property>

<!-- minimize the application master memory size -->
<property>
   <name>yarn.app.mapreduce.am.resource.mb</name>
   <value>512</value>
</property>

<property>
   <name>yarn.app.mapreduce.am.command-opts</name>
   <value>-Xmx256m -Xms256m</value>
</property>
```

Harp MapCollective Job Settings at the Client Side

```
Job job = Job.getInstance(configuration, "kmeans_job");
FileInputFormat.setInputPaths(job, inputDir);
FileOutputFormat.setOutputPath(job, outputDir);
job.setInputFormatClass(MultiFileInputFormat.class);
job.setJarByClass(KMeansLauncher.class);
job.setMapperClass(KMeansCollectiveMapper.class);
org.apache.hadoop.mapred.JobConf jobConf =
   (JobConf) job.getConfiguration();
jobConf.set("mapreduce.framework.name", "map-collective");
jobConf.setNumMapTasks(numMapTasks);
jobConf.setInt("mapreduce.job.max.split.locations", 10000);
job.setNumReduceTasks(0);
```