CS 249 - Cloud Computing

Adding support for GPU library within Apache Spark

...

Salil Kanetkar, Suket Sharma
704557096, 504591609

Outline

- Problem definition
- ‘Boost.Compute’ GPU Library
- ‘Pipe’ operator in Spark
- Sorting in Spark with GPU
- Challenges
- Future work
- Conclusion

Problem Definition

- Spark doesn’t have in-built support for native libraries (C/C++).
- Neither does it support any GPU computation.
- A lot of GPU libraries are written in C/C++.
- To determine if sorting on a GPU would be faster than in-built sorting available to Spark and adding support for it.

‘Boost.Compute’ GPU Library

- Boost.Compute is a GPU/parallel-computing library for C++ based on OpenCL.
- On the top of the core library is a generic, STL-like interface providing common algorithms and containers.
- Provides sorting like functionality using the ‘compute::sort()’.
- Sorting is limited to integer and floats. They generally have smaller memories and you cannot store large vectors of objects here.

‘Pipe’ operator in Spark

- Much like the pipe command in Unix, the pipe operator can be used to fork an external process and send it data belonging to the RDD.
- The forked process reads the data via stdin, and can return data through stdout.
- Using pipe to launch external processes is much faster than JNI (Java Native Interface).
- JavaRDD.pipe(“/path/to/script”) , will send each element of the RDD as input to the script.

Methodology

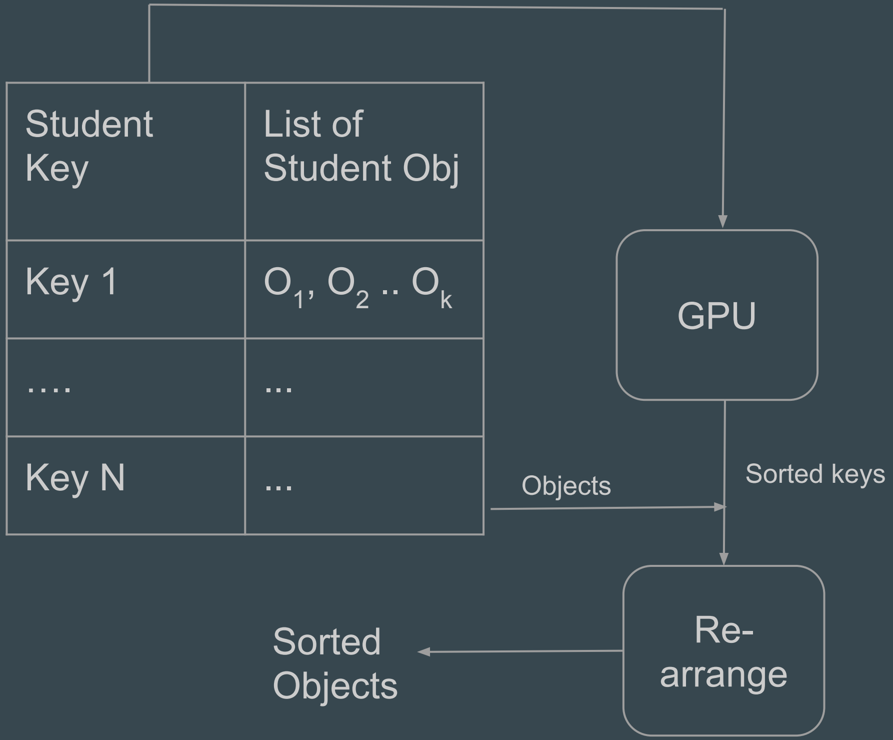
- We need to coalesce the JavaRDD object before sorting. Otherwise each partition within the JavaRDD will call a separate instance of our GPU sorting process.
- Data is sent via stdin and therefore read as strings and typecasted and sent to GPU for sort.
- Data is then printed out to stdout, and returned to Spark as JavaRDD<String>.
- The GPU library will perform Radix Sort on Integers and Merge on Floats.
- Why is GPU better? - Number of cores on a GPU > Number of cores on a CPU.

How to sort Objects?

- Objects can only be sent as a stream of bytes to the C++ process. (Java classes are serialized)
- Reconstruction of the object on C++ will require source code manipulation from Spark.
- Say if we know the structure of an object, we could write an equivalent C++ class into a .h file, which could then be included in the C++ process.
- Parsing the byte stream of Java objects into their C++ representations would be equally challenging.
- This is pretty much where JNI would help.

Simple Approach - Sort Key

- Define a sort key associated with a list of objects. For example a list of Student IDs associated with a list of Student objects.
- Send this list to the GPU, sort it, and rearrange the objects based on their sort.
- To handle repeated values in the sort key, we created a hash which slowed down our sorting. (O(n)) memory.



Student Key	List of Student Obj
Key 1	$O_1, O_2 .. O_k$
….	...
Key N	...

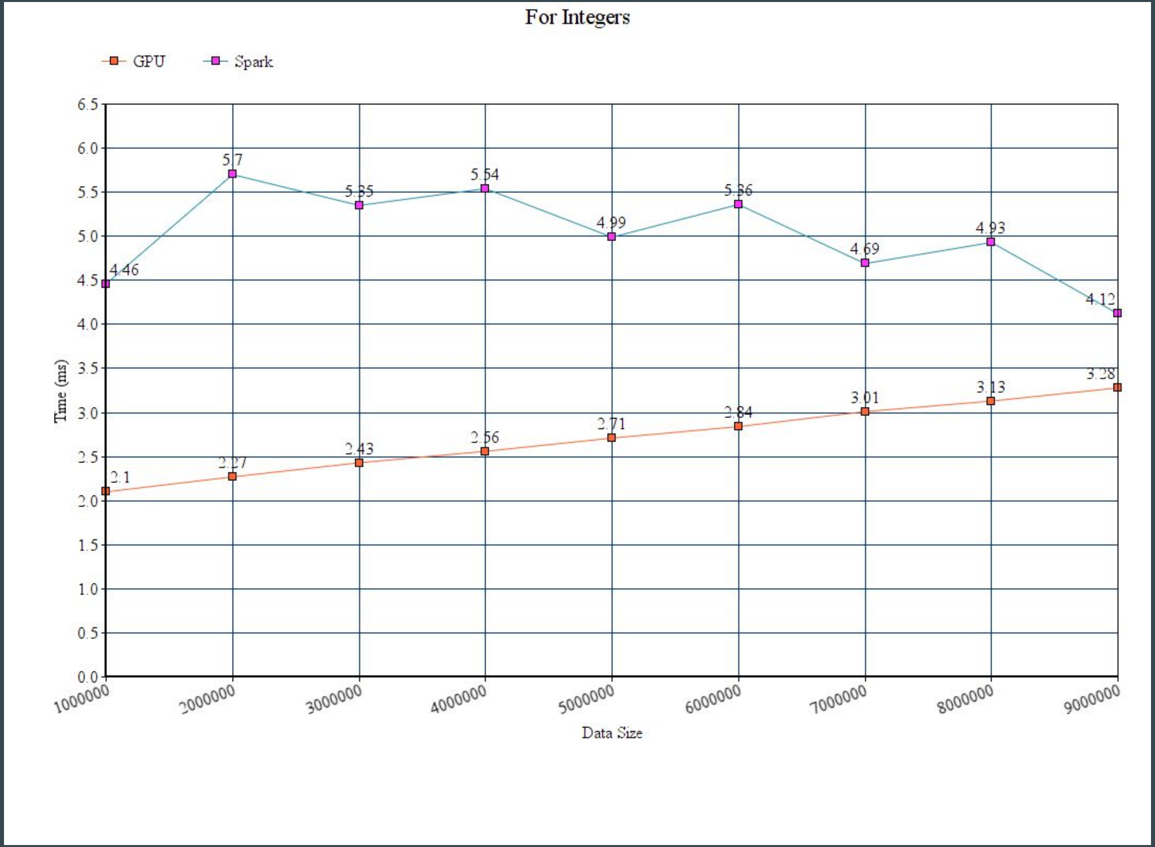
Platforms used

- Intel Core i5 - 2.7Ghz with 2 cores - 8GB
- Intel Iris Graphics 6000 - 1.5GB
- OpenCL 1.1 and Boost.Compute 1.7

Results - Ints

Execution times on Spark were not very stable.
Our guess is that this was because of varying CPU loads when we are testing.

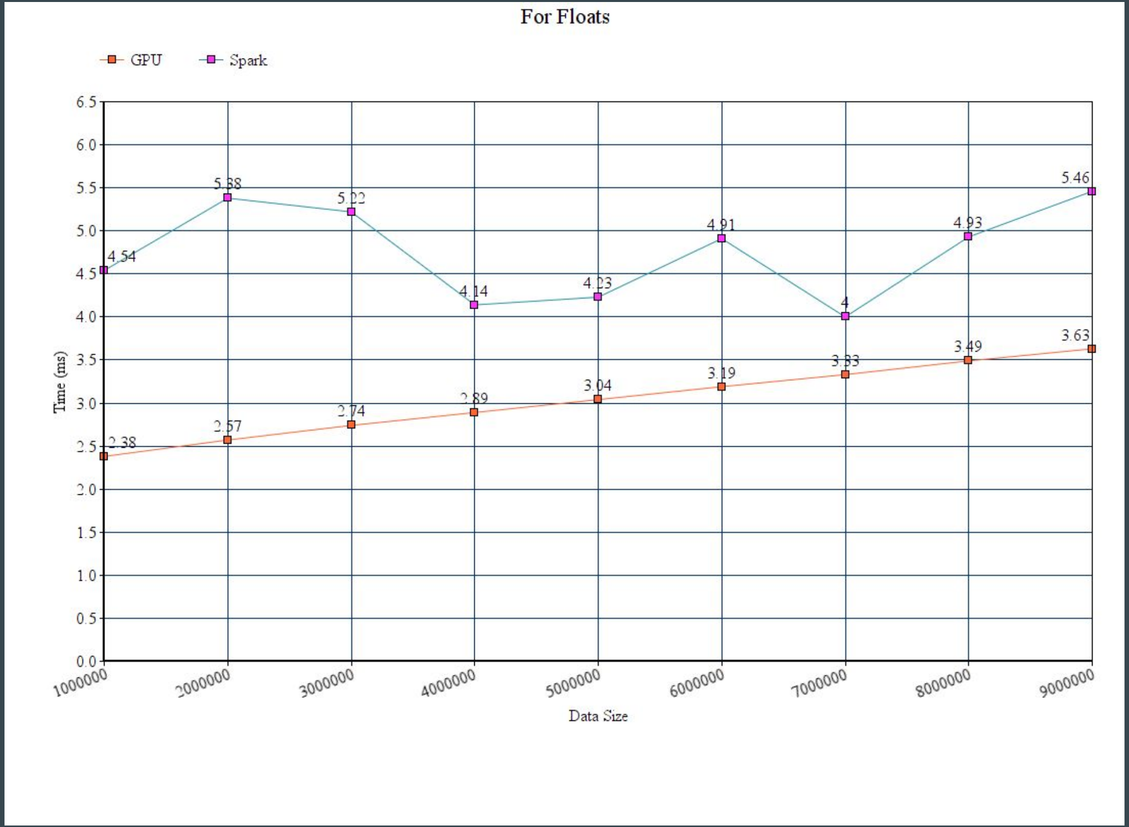
However, what GPU sorting was clearly always faster than using sorting primitives available in Spark.



Results - Floats

Very similar to results obtained for integers.

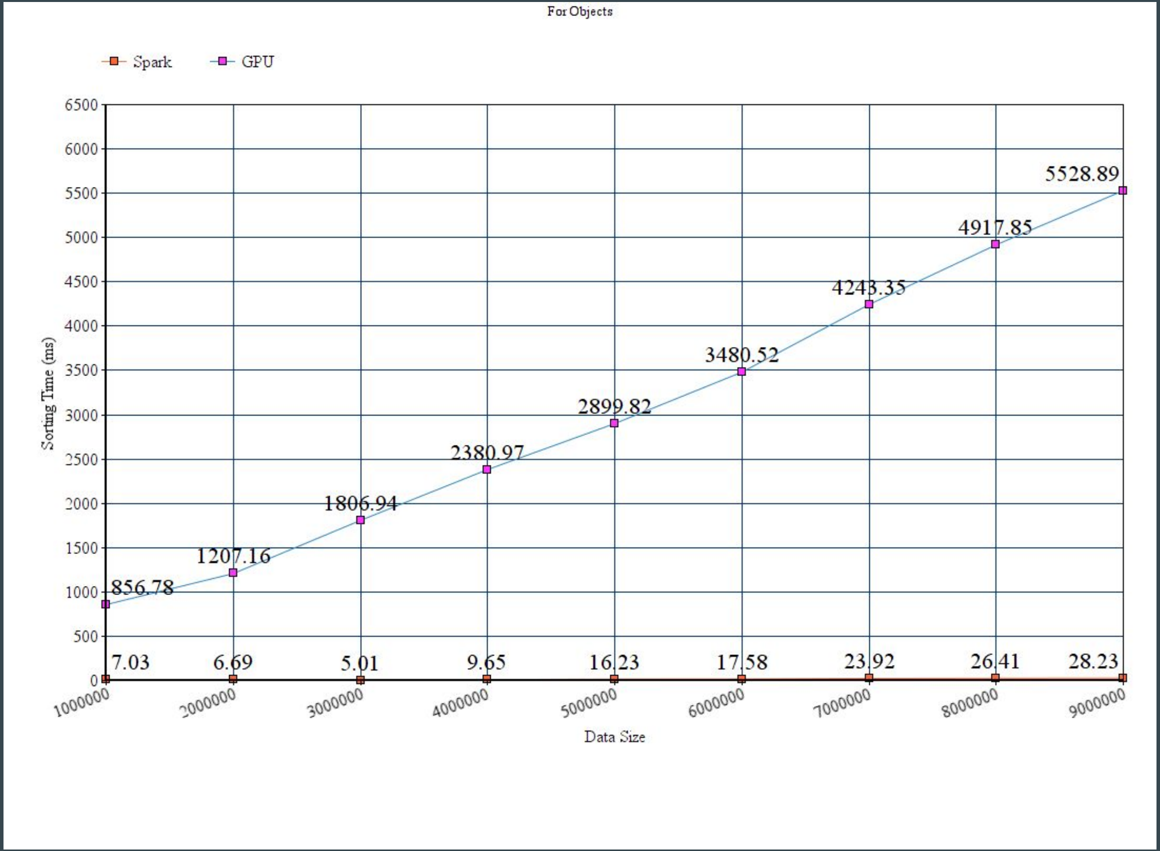
GPUs can help sort elements much faster. This includes time taken for sending and retrieving data from the GPU.



Results - Objects

There are two major overheads in sorting objects - Creating a hash table and converting a list to JavaRDD.

Lesson Learnt - Speed-Up in sorting ints and floats is not sufficient to offset overhead in hash creation and re-arranging.



Future Work

- Parsing Java Object Streams and reconstructing CPP objects could be faster.
- It would be easier to just let the Boost.Compute library take over sorting of all <Integer> and <Float> objects.
- Kyle Lutz has mentioned here that he intends to add support for sorting buffers soon.
- Once that happens, shifting sorting of strings etc to the GPU would make more sense.

Conclusion

- Sorting on GPU seems to be faster as compared to Spark for primitive data types like ‘int’ and ‘float’.
- Sorting of objects on GPU seems to be much more costly instead.