Sparkle: Optimizing Spark for Large Memory Machines and Analytics

Mijung Kim[1], Jun Li[2*], Haris Volos[1], Manish Marwah[1], Alexander Ulanov[3*], Kimberly Keeton[1], Joseph Tucek[4*], Lucy Cherkasova[5*], Le Xu[6*], Pradeep Fernando[7*]

[1]*Hewlett Packard Labs* [2]*Ebay* [3]*Facebook* [4]*Amazon Web Services* [5]*HyTrust* [6]*University of Illinois* [7]*Georgia Institute of Technology*

Introduction: Given the growing availability of affordable scale-up servers, our goal is to bring the performance benefits of in-memory processing on scale-up servers to an increasingly common class of data analytics applications that process small to medium size datasets (up to a few 100GBs) that can easily fit in the memory of a typical scale-up server [3]. To achieve this, we choose to leverage Spark, an existing memory-centric data analytics framework with wide-spread adoption among data scientists. Bringing Spark's data analytic capabilities to a scale-up system requires rethinking the original design assumptions, which although effective for a scale-out system, are a poor match to a scale-up system resulting in unnecessary communication and memory inefficiencies.

To address the inefficiencies and scalability issues, we have designed and implemented *Sparkle*, an enhanced Spark that leverages the large shared memory in scale-up systems to optimize Spark's performance for communication and memory intensive workloads. We have released Sparkle, our shuffle engine and off-heap memory store code to the public under Apache 2.0 License [2]. We have also released the generalized version of belief propagation algorithm [1] as an example of an application that benefits from our optimized Spark engine.

Sparkle Architecture: Figure 1 shows how Sparkle exploits global shared memory to transform Spark from a cluster-based scale-out architecture to a scale-up architecture.

At the bottom, a retail memory broker (RMB) layer provides a native memory management scheme that allows higher layers allocate and free blocks of global shared memory in a scalable manner.

For data shuffle, we have developed a shared-memory shuffle engine and integrated it into Spark under its pluggable shuffle interface. For data caching, we have developed an off-heap memory store that allows us to construct various large scale data structures in shared-memory regions managed by the RMB. The data structures developed include a sorted array and a hash table, to store intermediate data processing models, and to allow these models to be updated in place during iterations.

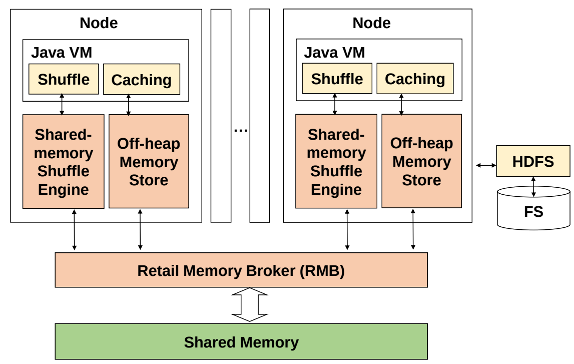

Figure 1: Sparkle using the global shared memory-based architecture to transform Spark from a cluster-based scale-out architecture to a scale-up architecture

Experimental Results: We conducted a series of experiments to estimate the effectiveness of the shared-memory shuffle engine and off-heap memory store. Our baselines are Vanilla Spark on a scale-out cluster (Vanilla-Scaleout) and Vanilla Spark on a scale-up hardware (Vanilla-Scaleup).

Our experiments include micro-benchmarks (*GroupBy*, *Join*, *PartitionBy*, *ReduceBy*, and *SortBy* Spark operators) and macro-benchmarks (TeraSort, PageRank and Belief Propagation (BP) applications). They represent typical Spark benchmarks and workloads. **FIXME: [show table or graph?]**

REFERENCES

[1] Project sandpiper: Implementation of the loopy belief propagation algorithm for apache spark. https://github.com/HewlettPackard/sandpiper.

[2] Sparkle: Optimizing spark for large memory machines. https://github.com/HewlettPackard/sparkle.

[3] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. Scale-up vs Scale-out for Hadoop: Time to rethink? In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 20. ACM, 2013.

*Work done while at Hewlett Packard Labs