Concepts and Models of Parallel and Data-centric Programming

Apache Spark – Concluding Remarks

Lecture, Summer 2020

Simon Schwitanski
Dr. Christian Terboven

i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Outline

0. Organization
1. Foundations
2. Shared Memory
3. GPU Programming
4. Bulk-Synchronous Parallelism
5. Message Passing
6. Distributed Shared Memory
7. Parallel Algorithms
8. Parallel I/O
9. MapReduce
10. **Apache Spark**

a. Spark Programming Model
b. Resilient Distributed Datasets (RDDs)
c. Job Scheduling and Fault Tolerance
d. Streaming and Applications
e. **Concluding Remarks**

MapReduce in Spark

- MapReduce can be expressed in Spark
- Reminder: $Map(k_1, v_1) \rightarrow list(k_2, v_2)$ / $Reduce\big(k_2, list(v_2)\big) \rightarrow list(k_3, v_3)$

```
1  // Input + Splitting
2  JavaPairRDD<K1,V1> input = // read input
3  // Mapping (k1,v1) -> list(k2,v2)
4  JavaPairRDD<K2,V2> mapOutput = input.flatMap(mapFunc)
5  // Shuffling and Sorting
6  JavaPairRDD<K2,V2> shuffled = mapOutput.groupByKey().sortByKey()
7  // Reducing (k2, list(v2)) -> list(k3,v3)
8  JavaPairRDD<K3,V3> output = shuffled.flatMap(reduceFunc)
9  // Output
10 output.save(...)
```

- Note 1: *sortByKey()* induces total sort instead of per-partition sort in MapReduce
- Note 2: *groupByKey()* shuffles *all* values, replacing it by *reduceByKey()* is like using a combiner in MapReduce (more efficient)

RDDs vs. Distributed Shared Memory (DSM)

- RDDs and *distributed shared memory (DSM)* both provide abstraction of distributed memory

Aspect	RDDs	DSM / PGAS
Reads	Coarse- or fine-grained	Fine-grained
Writes	Coarse-grained	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using backup tasks (see MapReduce)	Difficult
Work placement	Automatically based on data locality	Up to application
Behavior if not enough RAM	Similar to existing data flow systems (spill to disk)	Not intended, maybe swapping

Table Source: Matei Zaharia et al. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.” NSDI2012: 15-28

What You Have Learnt

What You Have Learnt

- Apache Spark: Generalization of MapReduce programming model
  - Supports caching of data in memory (beneficial for iterative algorithms)
  - Supports interactive analyses of data
  - Runs standalone or on top of cluster framework (YARN)
- Resilient Distributed Datasets (RDDs): Abstraction of distributed memory
  - Consists of datasets partitioned across cluster nodes
  - Modify RDDs with transformations (lazy operations) and materialize with actions
  - Narrow and wide dependencies between RDDs
  - Fault tolerance via lineage graph
  - More abstract than distributed shared memory (DSM / PGAS)