Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Principal Motivation

- MapReduce/Dryad built around acyclic flow of data
- Inefficient at handling iterative computation & data reuse
  - Machine Learning Algorithms
  - Interactive data mining tools
- Propose a solution for a class of applications that require
  - Working sets of data
  - Scalability and fault tolerance
Resilient Distributed Datasets

**Key Idea**

- Leverage distributed memory
- Improve upon specialised frameworks e.g. Haloop, Pregel, etc.

**What are RDDs?**

- Read-only collection objects
- Partitioned across several nodes
- Reconstructible incase of node failure
- Enables in-memory computation
Resilient Distributed Datasets

Representation of RDDs

• set of partitions

• set of dependencies — lineage

• function to compute RDD from parent RDDs

• metadata on partitioning scheme & data placement

Lineage

• Recompute elements of a partition

• Iterate over parent partitions; use the function in RDD
RDDS: Types of Dependencies

Narrow Dependencies
- One-to-one mapping of partitions between parent & child
- Pipelined execution on cluster nodes
- Involve map operation

Wide Dependencies
- Many-to-one mapping between parent & child
- Require data from all parent partitions and shuffle-like operation
- Involve join operation
### Key Differences

<table>
<thead>
<tr>
<th>Aspect</th>
<th>RDDs</th>
<th>Dist. Shared Mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reads</strong></td>
<td>Coarse or fine grained</td>
<td>Fine-grained</td>
</tr>
<tr>
<td><strong>Writes</strong></td>
<td>Coarse-grained; immutable consistency</td>
<td>Fine-grained</td>
</tr>
<tr>
<td><strong>Behaviour if not enough RAM</strong></td>
<td>Similar to existing data flow systems</td>
<td>Poor performance</td>
</tr>
<tr>
<td><strong>Fault Recovery</strong></td>
<td>Fine grained &amp; low-overhead using lineage</td>
<td>Requires checkpoints &amp; rollbacks</td>
</tr>
</tbody>
</table>

1. Resilient Distributed Datasets
Resilient Distributed Datasets

Computational Factors

• Cost of storage
• Disk I/O overhead
• Probability of node failure
• Cost of recomputing a partition

Limitations

• Inefficient for asynchronous fine-grained updates
• E.g. incremental web crawler, storage system for a webApp, etc.
Spark: Cluster Computing Framework

Introduction

- Implemented in Scala
- Built on top of Mesos (cluster operating system)
  - Enables resource sharing with Hadoop MPI
- RDD implementation
  - HDFS file objects
  - partition-to-block size mapping
Spark: RDD representation

Types of RDD constructs

• File in a shared file system e.g. HDFS

• Scala collection object e.g. an array

• Transforming an existing RDD using flatMap()

• Change persistence of an existing RDD
  
  • Cache action: dataset is kept in memory

  • Save action: dataset is written to the file system
Spark: Dataflow

- Driver program implements control flow
- Parallel programming abstractions
  - RDDs
  - parallel operations
- Types of parallel operations
  - reduce
  - collect
  - foreach
Spark: Dataflow

Job Scheduling

- RDD lineage graph examined
- DAG of stages is built
- Characteristics of a stage
  - as many narrow dependencies
- Wide dependencies require shuffle operation
- Tasks assigned on data locality
Spark: Limitations

- Scheduler failures not tolerated
  - re-run the task till stage’s parents available
  - else, replicate RDD lineage graph to compute partition
- Checkpointing API application/user dependent
  - Replicate Flag to persist
Spark: Assessment

Datasets

• User written applications
• ML algorithms: K-means & logistical regression
• 1 TB dataset for interactive queries

Benchmarks

• Hadoop: 0.20.2 stable release
• HadoopBinMem
  • converts input data to binary format
  • reduces over-head
Spark: Assessment

ML Algorithms

• Spark outperforms hadoop by 20x
• Avoided repeated I/O and deserialisation cost

Interactive query dataset

• Spark performed with the response time of 5.5-7s
• Dependent on the page rank implementation

User Applications

• Analytics report execution improved by 40x
• Other apps scale and perform well
RDDs: Conclusion

- Showed better performance
- Express cluster programming models
- Capture optimisations
  - keeping specific data in-memory
  - partitioning to minimize communication
  - recover from failures efficiently
- Promising paradigm in cluster computing