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

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley,
Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

Abstract

We present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarse grained transformations rather than fine-grained updates to shared state. However, we show that RDDs are expressive enough to capture a wide class of computations, including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these models do not capture. We have implemented RDDs in a system called Spark, which we evaluate through a variety of user applications and benchmarks.

1 Introduction

1: Cluster computing frameworks like MapReduce [10] and Dryad [19] have been widely adopted for large-scale data analytics. These systems let users write parallel computations using a set of high-level operators, without having to worry about work distribution and fault tolerance.

2: Although current frameworks provide numerous abstractions for accessing a cluster's computational resources, they lack abstractions for leveraging distributed memory. This makes them inefficient for an important class of emerging applications: those that reuse intermediate results across multiple computations. Data reuse is common in many *iterative* machine learning and graph algorithms, including PageRank, K-means clustering, and logistic regression. Another compelling use case is *interactive* data mining, where a user runs multiple ad hoc queries on the same subset of the data. Unfortunately, in most current frameworks, the only way to reuse data between computations (e.g., between two MapReduce jobs) is to write it to an external stable storage system, e.g., a distributed file system. This incurs substantial overheads due to data replication, disk I/O, and serialization, which can dominate application execution times.

3: Recognizing this problem, researchers have developed specialized frameworks for some applications that require data reuse. For example, Pregel [22] is a system for iterative graph computations that keeps intermediate data in memory, while HaLoop [7] offers an iterative MapReduce interface. However, these frameworks only support specific computation patterns (e.g., looping a series of MapReduce steps), and perform data sharing implicitly for these patterns. They do not provide abstractions for more general reuse, e.g., to let a user load several datasets into memory and run ad-hoc queries across them.

4: In this paper, we propose a new abstraction called *resilient distributed datasets* (RDDs) that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators.

5: The main challenge in designing RDDs is defining a programming interface that can provide fault tolerance *efficiently*. Existing abstractions for in-memory storage on clusters, such as distributed shared memory [24], key value stores [25], databases, and Piccolo [27], offer an interface based on fine-grained updates to mutable state (e.g., cells in a table). With this interface, the only ways to provide fault tolerance are to replicate the data across machines or to log updates across machines. Both approaches are expensive for data-intensive workloads, as they require copying large amounts of data over the cluster network, whose bandwidth is far lower than that of RAM,and they incur substantial storage overhead.

6: In contrast to these systems, RDDs provide an interface based on *coarse-grained* transformations (e.g., map, filter and join) that apply the same operation to many data items. This allows them to efficiently provide fault tolerance by logging the transformations used to build a dataset (its lineage) rather than the actual data.1 If a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to recompute just that partition. Thus, lost data can be recovered, often quite quickly, without requiring costly replication.

7: Although an interface based on coarse-grained transformations may at first seem limited, RDDs are a good fit for many parallel applications, because t*hese applications naturally apply the same operation to multiple data items*. Indeed, we show that RDDs can efficiently express many cluster programming models that have so far been proposed as separate systems, including MapReduce, DryadLINQ, SQL, Pregel and HaLoop, as well as new applications that these systems do not capture, like interactive data mining. The ability of RDDs to accommodate computing needs that were previously met only by introducing new frameworks is, we believe, the most credible evidence of the power of the RDD abstraction.

8: We have implemented RDDs in a system called Spark, which is being used for research and production applications at UC Berkeley and several companies. Spark provides a convenient language-integrated programming interface similar to DryadLINQ [31] in the Scala programming language [2]. In addition, Spark can be used interactively to query big datasets from the Scala interpreter. We believe that Spark is the first system that allows a general-purpose programming language to be used at interactive speeds for in-memory data mining on clusters.

9: We evaluate RDDs and Spark through both microbenchmarks and measurements of user applications. We show that Spark is up to 20× faster than Hadoop for iterative applications, speeds up a real-world data analytics report by 40×, and can be used interactively to scan a 1 TB dataset with 5–7s latency. More fundamentally, to illustrate the generality of RDDs, we have implemented the Pregel and HaLoop programming models on top of Spark, including the placement optimizations they employ, as relatively small libraries (200 lines of code each).

10: This paper begins with an overview of RDDs (§2) and Spark (§3). We then discuss the internal representation of RDDs (§4), our implementation (§5), and experimental results (§6). Finally, we discuss how RDDs capture several existing cluster programming models (§7), survey related work (§8), and conclude.

2 Resilient Distributed Datasets (RDDs)
2.1 RDD Abstraction
2.2 Spark Programming Interface
2.2.1 Example: Console Log Mining
2.3 Advantages of the RDD Model
2.4 Applications Not Suitable for RDDs

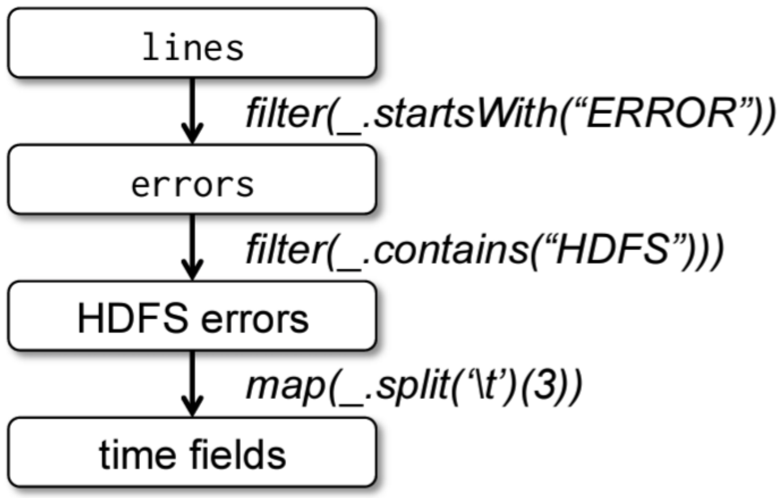

Figure 1: Lineage graph for the third query in our example. Boxes represent RDDs and arrows represent transformations.

3 Spark Programming Interface
3.1 RDD Operations in Spark
3.2 Example Applications
3.2.1 Logistic Regression
3.2.2 PageRank

4 Representing RDDs

5 Implementation
5.1 JobScheduling
5.2 Interpreter Integration
5.3 MemoryManagement
5.4 Support for Checkpointing

6 Evaluation
6.1 Iterative Machine Learning Applications

6.2 PageRank
6.3 Fault Recovery
6.4 Behavior with Insufficient Memory
6.5 User Applications Built with Spark
6.6 Interactive Data Mining

7 Discussion
7.1 Expressing Existing Programming Models
MapReduce, DryadLINQ, SQL, Pregel, IterativeMapReduce, BatchedStreamProcessing, ExplainingtheExpressivityofRDDs
7.2 Leveraging RDDs for Debugging

8 Related Work
Cluster Programming Models, Caching Systems, Lineage, Relational Databases

9 Conclusion
We have presented resilient distributed datasets (RDDs), an efficient, general-purpose and fault-tolerant abstraction for sharing data in cluster applications. RDDs can express a wide range of parallel applications, including many specialized programming models that have been proposed for iterative computation, and new applications that these models do not capture. Unlike existing storage abstractions for clusters, which require data replication for fault tolerance, RDDs offer an API based on coarse-grained transformations that lets them recover data efficiently using lineage. We have implemented RDDs in a system called Spark that outperforms Hadoop by up to 20× in iterative applications and can be used interactively to query hundreds of gigabytes of data. We have open sourced Spark at spark-project.org as a vehicle for scalable data analysis and systems research.