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

**Summary:**

The main focus of this paper is Resilient Distributed Datasets (RDDs) and nature of problems this structure (RDDs) can solve with different benchmarks it’s differences from existing available architectures.

The already available architecture like Map-reduce lack generality but RDDs solve this issue with fault tolerant, distributed memory abstraction based architecture. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools.

The author put some light on the RDD structure. Deterministic operations on data in stable storage results into immutable partitioned RDDs. RDDs can be reconstructed in case of failures using the **lineage information (the information about the sources from which they are derived). RDDs are created for specific requirement and their life span ends with the end of requirement.**

RDDs are fault tolerant as all functions and transformations are applied to all the data items. This behavior of RDDs is due to its fined grained reads and coarse grained updates but this produces a drawback in shape of being unsuitable for applications like incremental web crawler that needs asynchronous fine-grained updates to a shared state. The author then compare the RDDs with DSM (Distributed Shared Memory). RDDs have a lot of advantage like

But this update nature makes RDDs unsuitable for applications like incremental web crawler that needs asynchronous fine-grained updates to a shared state. In such cases, DSM (Distributed Shared Memory) would be a better choice as it provides fine-grained reads and writes. Although RDDs offer many advantages over DSM. RDDs do not need to incur check pointing overhead. RDDs runs [backup tasks](https://medium.com/@shagun/mapreduce-1c88f8a7c3d2#9f04) like Map-Reduce to mitigate slow nodes by running. RDDs enhance performance by using data locality. Immutable nature of RDDs doesn’t affect its consistency even if RDDs take checkpoints.

The author proposes a graph-based representation of RDDs where an RDD is expressed through a common interface that have functionality of partition, dependencies, iterator, practitioner,preferred Location. In this representation dependencies are either expressed as narrow dependencies or as wide dependencies. Recovery is easier with narrow dependencies while in the case of wide dependencies, failure of a single partition may require a complete re-execution. The author explained the behavior of scheduler, whenever an action is executed the scheduler builds a Directed Acyclic Graph of stages based on lineage graph. Some stages are precomputed while other uses **delay scheduling** to assign tasks to machines based on data locality.

The author evaluated the spark through a series of experiments and it was concluded that the Spark outperforms Hadoop and HadoopBinMem. Spark performs well for interactive data analysis and other user applications. **One limitation of the experiments is that in all the cases comparing the 3 systems, the cluster had sufficient RAM to keep all the data in-memory.**In the later part of the paper the author compare the RDDs and Spark with the existing systems and their contribution towards improvement of existing systems. The author concluded paper with an observation that previous architectures face the problem of lack of data sharing abstractions and this observation tackles a serious question earlier architectures could not provide same level of generality.

RDDs offer a simple and efficient programming model for broad range of applications. Leverage the coarse-grained nature of many parallel algorithms for low-overhead recovery.

**Questions:**

**1 -** What would be the performance of the three systems (Spark, Hadoop and HadoopBinMem) in the case where the cluster does not have sufficient RAM to keep the entire data in main memory?

**2 -** How do the RDDs provide partition control for user?

**3 -** What is efficient fault recovery mechanismof RDDS?

**4 -** RDDs can only be created/“written” through coarse-grained deterministic transformations. Is there any other way around to create or write RDDs?

**5 -** Do RDDs allow to intermix with the existing architectures?