- Leverages aggressively cached in-memory distributed computing and JVM threads
- Faster than MapReduce for some workloads
- Real-time, fast, efficient access to current transactions, data as well as historical

One of the **main advantages** of Apache Spark lies in its ability to perform federated analytics over a heterogeneous source data landscape.

Ease of use (for programmers)
- Written in Scala, an object-oriented, functional programming language
- Scala, Python and Java APIs
- Scala and Python interactive shells
- Runs on Hadoop, Mesos, standalone or cloud

Co-location with Data & Transactions:
- Performance loading Spark RDDs
- Governance of RDD memory leverages z/OS
- Reduced ETL need

General purpose
- Covers a wide range of workloads
- Provides SQL, streaming and complex analytics

IBM z /OS Systems platform: A unique analytics implementation
- Only Apache Spark z/OS offers integrated, optimized, parallel access to almost all z/OS data environments as well as distributed data sources
- All Spark memory structures that will contain sensitive data are governed with z/OS security capabilities
- Analyze data in place means that you can include real-time operational data and warehoused data
- No need to have all data on z/OS: Spark z/OS can access a wide variety of sources
- Sysplex enabled Spark clusters for world class availability
- Leverages z/OS superior capabilities in memory management, compression, and RDMA communications to provide a high-performance scale up and scale out architecture.
- Uses unique features of z such as: large pages, incorporating DRAM with large amounts of Flash as an attractive means to provide scalable elastic memory.
- Provides a best fit analytic capability for the investments made in SMF in-memory analytics
- Leverages and gets benefit from our zEDC compression technology, particularly when compressing internal data for caching and shuffling.
- SMT2 for added thread performance
- SIMD for better performance on select operations
- zIIP eligible - for affordability
- Intra-SQL and intra-partition parallelism for optimal data access.

Spark is not only about data access, it is about the framework that is offered in terms of analytic programming context.

Sysplex enabled Spark clusters for world class availability.

Resilient Distributed Datasets (RDDs)
- Spark's basic unit of data
- Immutable, fault tolerant collection of elements that can be operated on in parallel across a cluster
- Fault tolerance
  - If data in memory is lost it will be recreated from lineage
- Caching, persistence (memory, spilling, disk) and check-pointing
- Many database or file type can be supported
- An RDD is physically distributed across the cluster, but manipulated as one logical entity
  - Spark will "distribute" any required processing to all partitions where the RDD exists and perform necessary redistributions and aggregations as well.

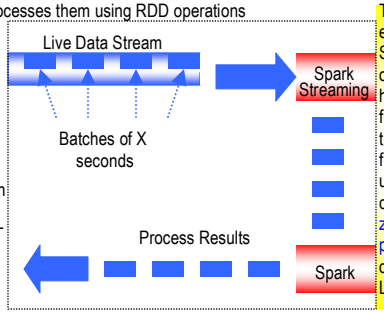
Provides a best fit analytic capability for the investments made in SMF in-memory analytics.

You create a SchemaRDD from existing RDDs, a Parquet file, a JSON dataset, or using HiveQL to query against data.

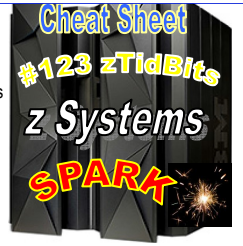
Common, popular methods to access data
- Spark SQL
  - Provide for relational queries expressed in SQL, HiveQL and Scala
  - Seamlessly mix SQL queries with Spark programs
  - Provide a single interface for efficiently working with structured data including Apache Hive, Parquet and JSON files
  - Standard connectivity through JDBC/ODBC
  - Spark z/OS has unique functionality to access data across wide variety of environments, support SQL 92, 99 standards with very high performance and flexibility.

Intra-SQL and intra-partition parallelism for optimal data access.

Spark Streaming
- Run a streaming computation as a series of very small, deterministic batch jobs
  - Chop up live stream into batches of X seconds
  - Spark treats each batch of data as RDDs and processes them using RDD operations
  - The process results of the RDD operations are returned in batches
    - > Combine live data streams with historical data
      - >> Generate historical data models with Spark
      - > > Use data models to process live data
    - > Combine Streaming with MLlib algorithms
      - >> Offline learning, online predictions
      - >> Actionable information



With Apache Spark
- Federate the analytics across the data sources with consistent APIs – leave data in origination point
- "Lake" becomes virtualized across multiple environments
- Resulting in:
  - Current data & reduced time to analytic insight
  - Security and governance matched to where data originated
  - Reduced time spent on just moving data around



Common Patterns

Analytics across OLTP & Warehouse information
- Clients have OLTP on z/OS (DB2, VSAM, IMS, PDSE, ADABAS….)
- Clients have warehouses on distributed platforms
- Analytics across these environments can be challenging and inconsistent

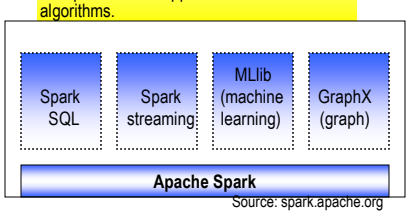
Analytics combining business-owned data and external / social data
- Clients have OLTP on z/OS
- Clients have external – public --- or social data on distributed servers
- External data delivers more value when combined with analytics from business data

Analytics of real-time transactions via streaming, combine with OLTP & social
- Combining real-time transaction data streaming into Spark via CICS with high performance with OLTP data and social media data --- for example, claims analytics

Analytics to improve system performance and operations in real-time using streaming as well as archived data
- SMF real-time data can add much more insight for IT across multiple systems, add in analytics over SMF data that has been archived and integrate log information

The IBM z/OS Platform for Apache Spark is changing analytics forever …

- Apache Spark is an open source, in-memory analytics computing framework offered by the Apache Foundation; not a product.
- An in-memory compute engine that works with data; not a data store.
- Enables highly iterative analysis on large volumes of data at scale
- Unified environment for data scientists, developers and data engineers
- Radically simplifies the process of developing intelligent apps fueled by data
- Spark offers a unified programming environment and is extremely lightweight. What is most important is that Spark is function-rich in that it provides libraries for commonly used analytic methodologies for data access, manipulation and application of various algorithms.

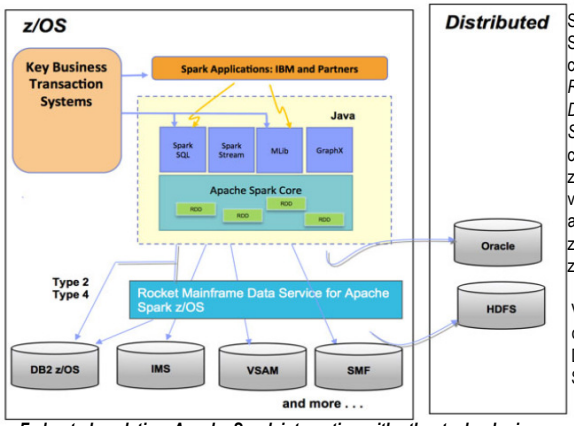


Source: spark.apache.org

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 which these models do not capture.

The key values for enterprises is why IBM Systems has enabled Spark natively for both z/OS and Linux on z Systems. Apache Spark is enabled on both of the operating system environments supported on z Systems hardware; clients can choose the configuration that fits best with their needs. The suggestion is to consider the originating sources of data and transactions that will feed the Spark analytics. If most of the data that will be used for Spark analytics, or the most sensitive or quickly changing data is originating on z/OS, then a Spark z/OS based environment will be the optimal choice for performance, security, and governance. If most of the data that will be used for Spark analytics originates on Linux on z, then a Linux on z Spark is a viable approach.

Below displays a more closely the structure of an Apache Spark environment on z/OS.



Federated analytics: Apache Spark integration with other technologies

Specific to z/OS Apache Spark, Rocket Software created a function named *Rocket Software Mainframe Data Service for Apache Spark z/OS*. This capability can enable Apache Spark z/OS to have optimized, virtualized, and parallelized access to a wide variety of z Systems and non-z Systems data sources.

VSAM leverages the optimized Rocket Mainframe Data Service for Apache Spark z/OS.

- Spark can also be clustered across more than one JVM, and these Spark environments can be dispersed across an IBM Parallel Sysplex®.
- Spark is based on Java, the potential exists for z Systems transactional environments, customer-provided applications, and IBM and other vendor applications to leverage the consistent Spark interfaces with almost all zIIP-eligible MIPS.
  - In this way, analytics processing on z/OS becomes extremely affordable.
  - With the IBM z13™ system, IBM supports up to 10 TB of memory that can enable the in-memory RDD Spark structures for optimal performance.
  - Through the Spark SQL interfaces, access to DB2 z/OS and IMS can be facilitated through standard types 2 and 4 connections.
    NOTE: Depending on the level of data integration that you want, access to VSAM, physical sequential, SMF, SYSLOG, and other environments is also possible.

All Spark memory structures that will contain sensitive data are governed with z/OS security capabilities.

More on RDD
- Formally, an RDD (Spark's basic unit of data) is a read-only, partitioned collection of records.
  - RDDs can only be created through deterministic operations on either:
    a. Data in stable storage
    b. Other RDDs.
  - We call these operations transformations to differentiate them from other operations on RDDs. Examples of transformations include map, filter, and join.

Data in place on z Systems means that clients *can* choose to run other SW products that use this data on z Systems.

- RDDs do not need to be materialized at all times.
  - An RDD has enough information about how it was derived from other datasets (its lineage) to compute its partitions from data in stable storage.
    NOTE: This is a powerful property: in essence, a program cannot reference an RDD that it cannot reconstruct after a failure.
  - Finally, users can control two other aspects of RDDs: persistence and partitioning.
    - > Users can indicate which RDDs they will reuse and choose a storage strategy for them (e.g., in-memory storage).
    - > They can also ask that an RDD's elements be partitioned across machines based on a key in each record.
      NOTE: This is useful for placement optimizations, such as ensuring that two datasets that will be joined together are hash-partitioned in the same way.