Memory Speed Big Data Analytics

Alluxio vs Apache Ignite

Irfan Elahi

Deloitte.

About Me

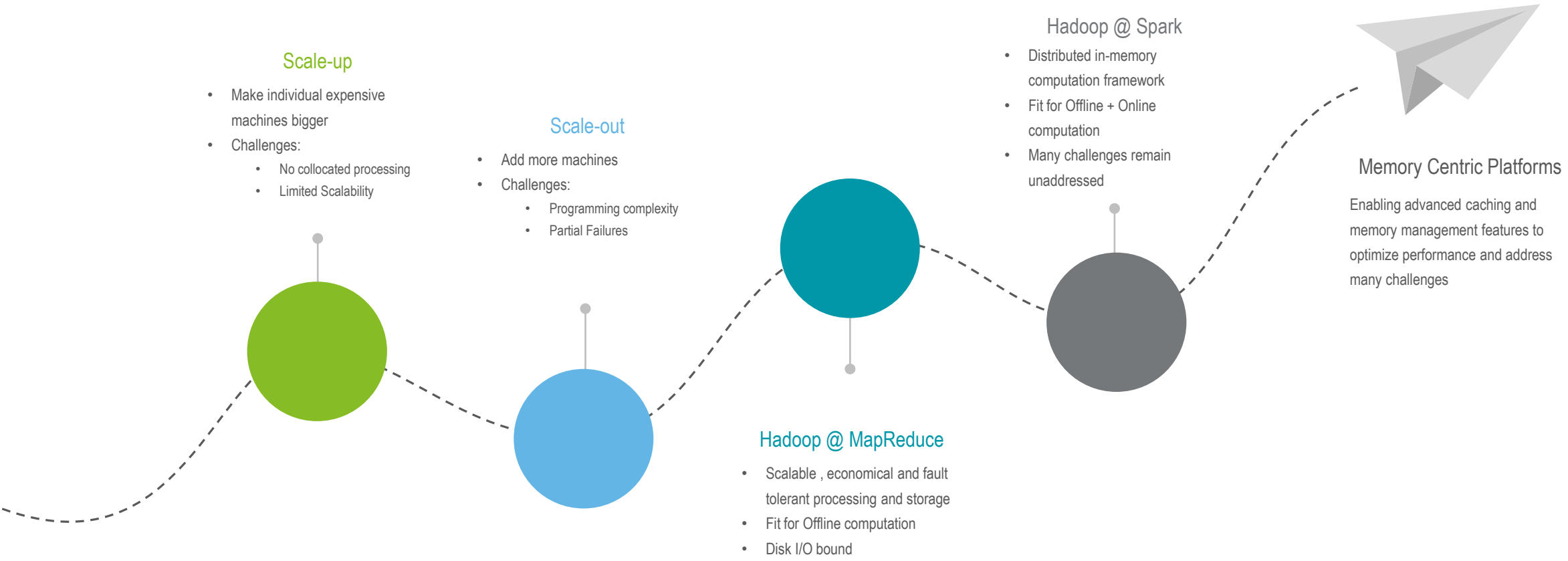
- Working as a Consultant in Deloitte (Analytics Service Line)
- 4+ years of experience in Big Data and Machine Learning in multiple verticals
- Recent Deloitte projects in Australia's biggest Telecom company:
  - Architecting one of the largest Hadoop deployments in cloud employing in-memory computation technologies
  - Developing enterprise-grade stream processing system based on Hadoop stack employing NoSQL data-stores
- Premium Udemy Instructor with 12,000+ students from 131 countries
- Technical Reviewer of an upcoming Hadoop book published by APress
- Lets connect: ielahi@deloitte.com.au | linkedin.com/in/irfanelahi | @elahi_irfan

Agenda

- Big Data – The Evolution and Beyond
- In-Memory Computation Trend – Overview
- Unaddressed Challenges in Big Data
- Introduction and Deep Dive Comparison of Alluxio and Apache Ignite

Big Data

The Evolution and Beyond



Name of the game:

Memory is the new disk!

In-Memory Trend

Overview

Driving Factor: Economics

- Memory is much faster than disk (approx. 3000x)
- Cost of memory decreasing
- Memory per node increasing

Driving Factor: Traditional paradigms' Limitations

Intermittent disk I/O and serialization cost in traditional computing platforms (e.g. MapReduce) causes:

- High Latency
- In-efficiency in iterative algorithms execution in analytics (e.g. machine learning, graph and network analysis)
- In-efficiency in interactive data mining
- Infeasibility for innovative use-cases like stream processing

Impact: Innovative technologies and processing patterns

- New processing patterns:
  Batch -> Event Driven
- New processing technologies:
  Map Reduce -> Spark
  Hive -> Impala
- New storage technologies:
  HDFS -> Alluxio | IGFS
- New Use-cases:
  Real-time stream processing
  IoT

Challenges:

- Memory is still expensive than disk (approx. 80x)
- Memory is still limited
- Not all data is memory-worthy

and that's not all…

Un-addressed Challenges:
Overview

On-Heap Memory Constraints:

- On-Heap memory in memory-centric platforms (e.g. Spark) is limited thus causing resource pressure
- Data resilience is compromised in the event of application crashes and causes expensive disk I/O
- Inter-process data/state sharing still relies on HDFS I/O thereby causing performance issues

Many Compute to Storage Integration Paradox:

- Managing increasing number of compute and storage platforms increases complexity
- Adding/Removing respective systems require application changes thus impacting DevOps lifecycle
- Data locality gets compromised

Missing SQL and Transactional Support on Hadoop

Many leading Big data platforms still don't support:

- ACID compliant Transactions
- ANSI SQL compliance
- Indexing
- In-place mutation

Potential Missing Pieces of Puzzle:

- Alluxio
- Apache Ignite

Alluxio

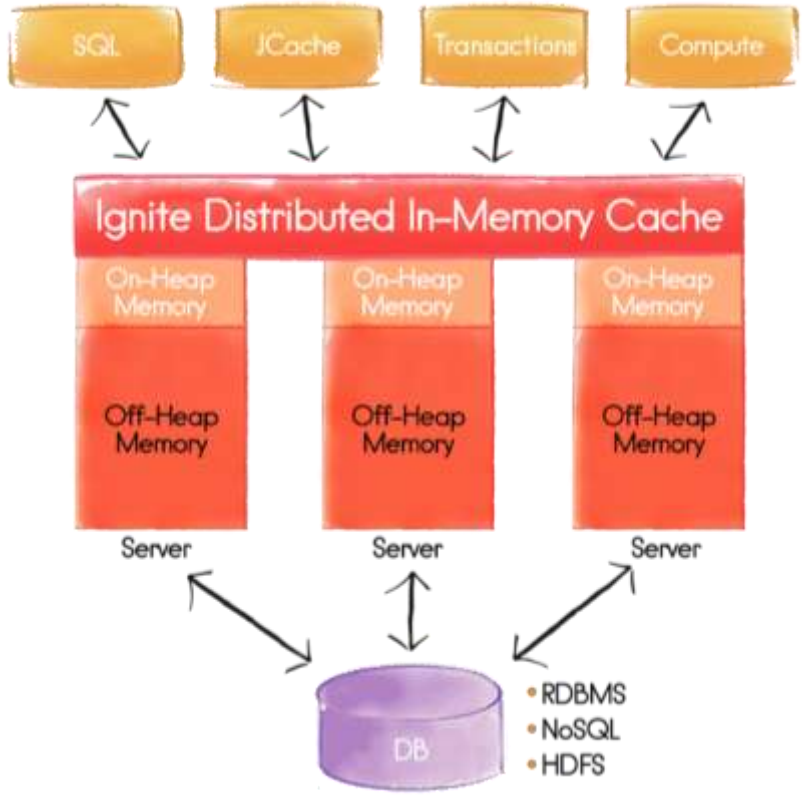
A distributed and scalable storage virtualized across multiple storage systems under unified namespace to facilitate data access at memory speed

- Launched in 2012 by UC Berkeley AMPLab
- Formerly known as Tachyon
- Licensed under Apache License 2.0
- Approximately 500 contributors
- Deployed in Yahoo, Baidu, Intel, Samsung to name a few

Ignite

An distributed key-value store and scalable in-memory computing platform with powerful SQL, key-value and processing APIs

- First release in early 2015
- August 2015: second fastest project to graduate after Spark
- Licensed under Apache License 2.0
- Approximately 120 contributors
- Deployed in IBM, Siemens, Citibank, Barclays, Nielsen to name a few

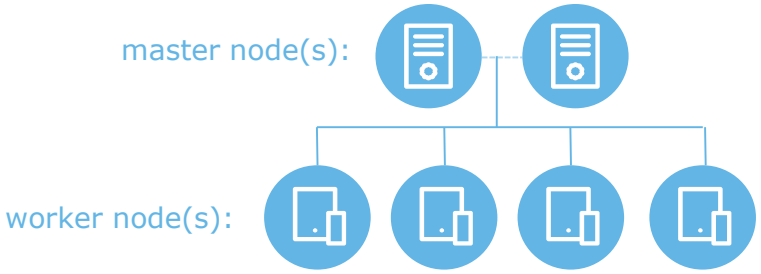


Deep Dive Comparison

- Alluxio
- Apache Ignite

Architecture

Alluxio



Master Nodes:
- Manage File System Metadata
- Can be Primary or Secondary
- HA supported via ZooKeeper

Worker Nodes:
- Store data in the form of blocks
- No rebalancing of blocks upon addition of new nodes
- Send heartbeats to Master Nodes

Require Under File System (UFS) (e.g. HDFS, S3) for operation

Apache Ignite

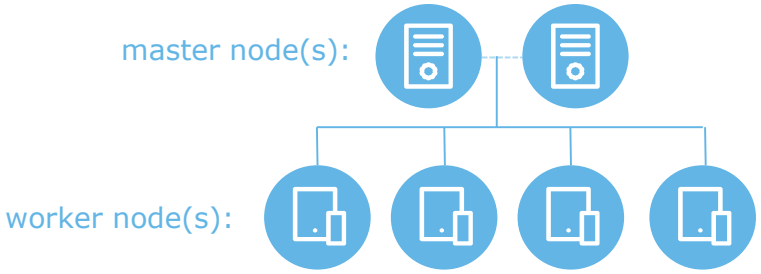
Optional Node Roles:
- Servers (Default | Equal by design | Multiple servers on one host)
- Clients (Explicitly defined | Connect to servers for computation)

Logical Grouping:
- User configurable node roles via attributes registered by nodes at start-up
- Registered attributes can be leveraged for dynamic logical grouping based on predicates (e.g. CPU Utilization > 50%) for localizing processes and jobs

No Name-Node Architecture:
- When used as IGFS, no centralized metadata management (e.g. like HDFS NameNode or Alluxio's Master Nodes) is needed
- Hashing is used for data locality determination

Architecture (Continued)

Alluxio



Configuration:
- Requires explicitly specifying:
  - Master Node(s)
  - Worker Node(s)

  in the configuration files
- Addition of nodes requires restarting cluster

Interfaces:
- Alluxio Shell
- Web interface (also enables to browse Alluxio FS)

Apache Ignite

Configuration:
- Doesn't require explicit specification of nodes in configuration files
- Nodes discover themselves automatically when started
- Supported methods for nodes discovery:
  - Multi-cast
  - Static IP based
- Cluster can be scaled without restarting
- Supported deployment modes: Shared or Embedded

Interfaces:

Visor CommandLine:

For viewing topology, node metrics, cache statistics and administrating cluster

Web Interface:

Needs to be installed separately

Architecture (Continued)

Alluxio

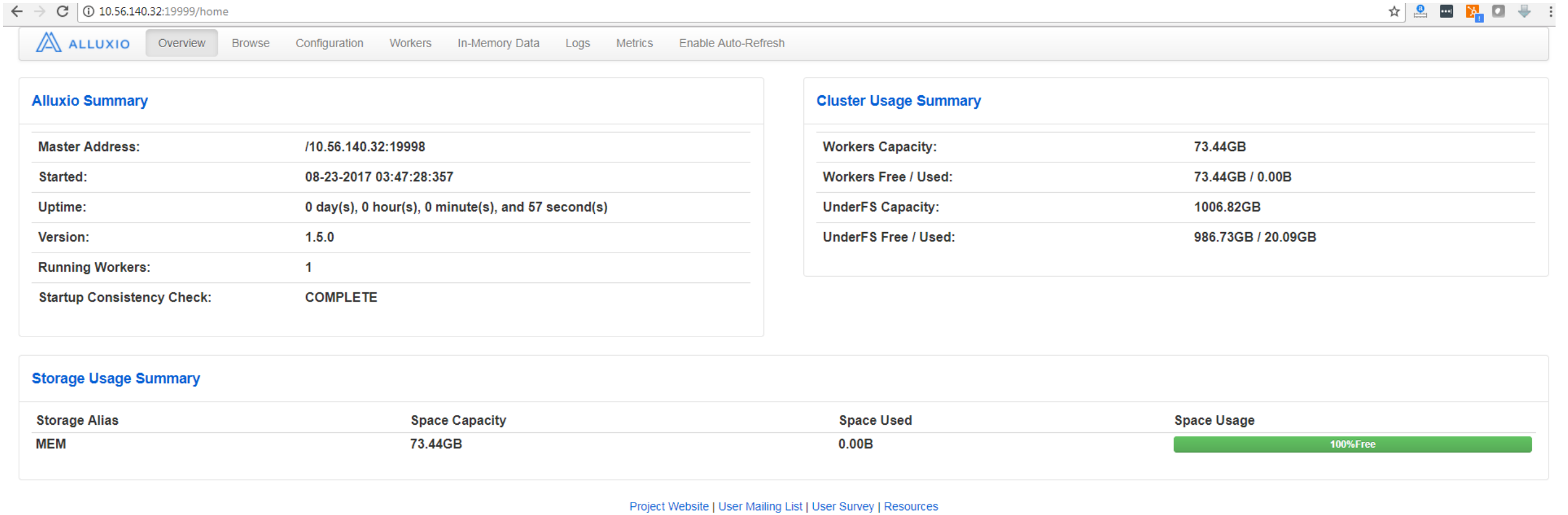
```
[Cloudera@lxapp5917 alluxio-1.5.0-hadoop-2.8]$ ./bin/alluxio fs
Usage: alluxio fs [generic options]
       [cat <path>]
       [checkConsistency [-r] <Alluxio path>]
       [checksum <Alluxio path>]
       [chgrp [-R] <group> <path>]
       [chmod [-R] <mode> <path>]
       [chown [-R] <owner> <path>]
       [copyFromLocal <src> <remoteDst>]
       [copyToLocal <src> <localDst>]
       [count <path>]
       [cp [-R] <src> <dst>]
       [createLineage <inputFile1,...> <outputFile1,...> [<cmd_arg1> <cmd_arg2>
...]]
       [deleteLineage <lineageId> <cascade(true|false)>]
       [du <path>]
       [fileInfo <path>]
       [free [-f] <path>]
       [getCapacityBytes]
       [getUsedBytes]
       [head -c <number of bytes> <path>]
       [help <command>]
       [leader]
       [listLineages]
```

Alluxio Shell

Apache Ignite

```
(wrn) <visor>: Topology is empty.
visor> top
Empty topology.
visor> top
Empty topology.
visor> top
Hosts: 1
+=========================================================================================
=================================================+
|   Int./Ext. IPs     |   Node ID8(@)     | Node Type |                    OS
             | CPUs |       MACs        | CPU Load |
+=========================================================================================
=================================================+
| 0:0:0:0:0:0:0:1%lo | 1: A2CEF9FE(@n0) | Client    | Linux amd64 3.10.0-327.36.
3.el7.x86_64 | 16   | 00:0D:3A:D1:2C:17 | 0.00 %   |
| 10.56.140.32       |                   |          |
             |      |                   |          |
| 127.0.0.1          |                   |          |
             |      |                   |          |
+-----------------------------------------------------------------------------------------
-------------------------------------------------+

Summary:
+-------------------------------------+
| Total hosts    | 1                  |
| Total nodes    | 1                  |
| Total CPUs     | 16                 |
| Avg. CPU load  | 0.00 %             |
| Avg. free heap | 64.00 %            |
| Avg. Up time   | 00:01:02           |
| Snapshot time  | 08/29/17, 23:51:11 |
+-------------------------------------+
visor>
```

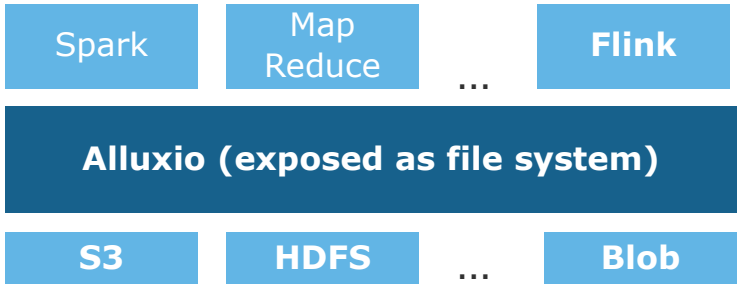
Visor CommandLine



Alluxio Web Interface

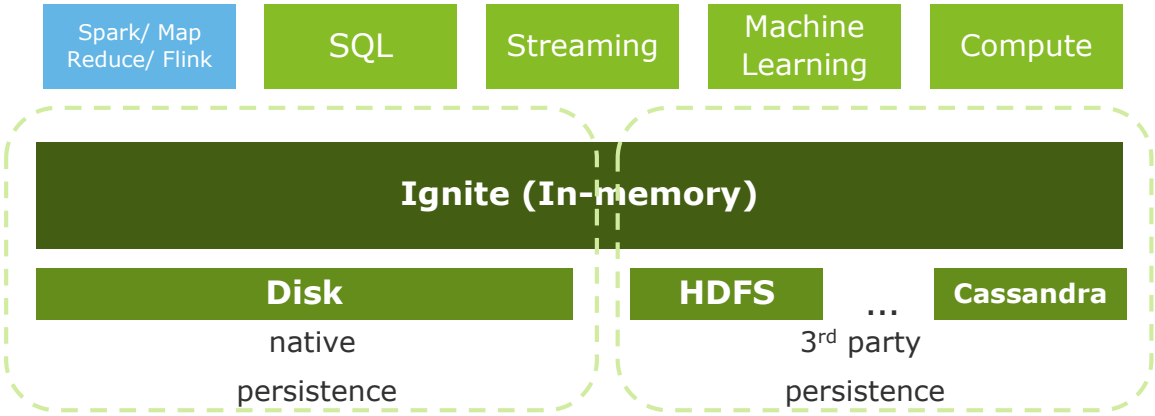
Integration with Data Stores

Alluxio



- Enables processing frameworks to interact with data from different data stores with unified namespace and API
- Conveniently supports the following data stores as UFS:
  - HDFS, Blob, S3, GCS, Minio, Ceph, Swift, MapR-FS to name a few
- Process involves mounting different UFS at different mount points in Alluxio namespace and then accessing seamlessly in applications
- Addition of more UFS storage is configurable

Apache Ignite



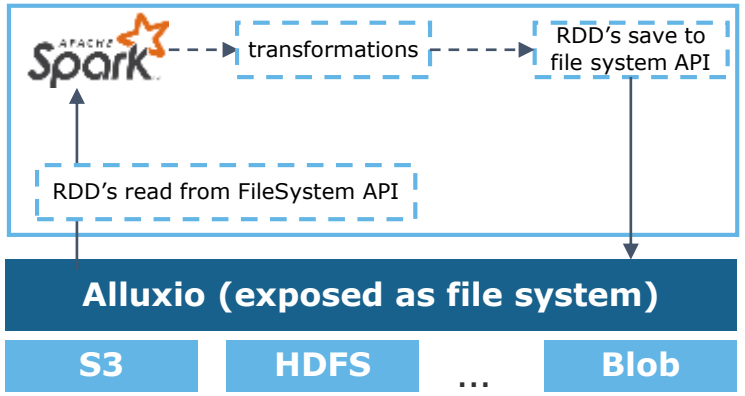
- Provides two modes of persistence in addition to in-memory:
  - Native Persistence (disk only)
  - 3rd party Persistence (pluggable)

Native persistence:
- Treats disk for persisting super-set of data
- Supports SSD, Flash, 3d Xpoint storage
- Features like ACID compliance, SQL are supported only in this mode

3rd Party Persistence:
- Data stores like HDFS, Cassandra and JDBC based are pluggable
- Involves implementing CacheStore interface for read/write through
- Supports write-behind caching for improved performance

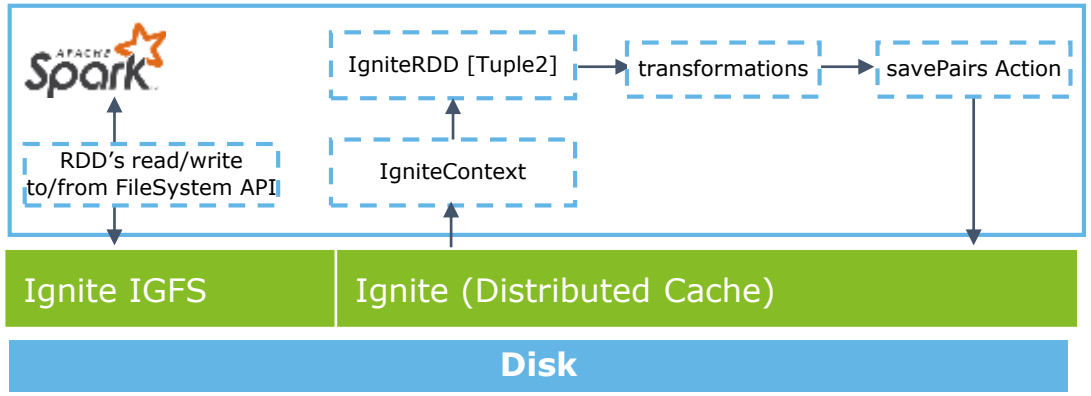
Integration with Spark

Alluxio



- Alluxio is provides Hadoop compatible file system APIs and thus data can be read/written via Spark RDD's file system related APIs
- Enables to read/write data from different data stores (configured as UFS) via Alluxio's unified Namespace and API
- Automatically manages movement of data persisted in Alluxio or UFS

Apache Ignite

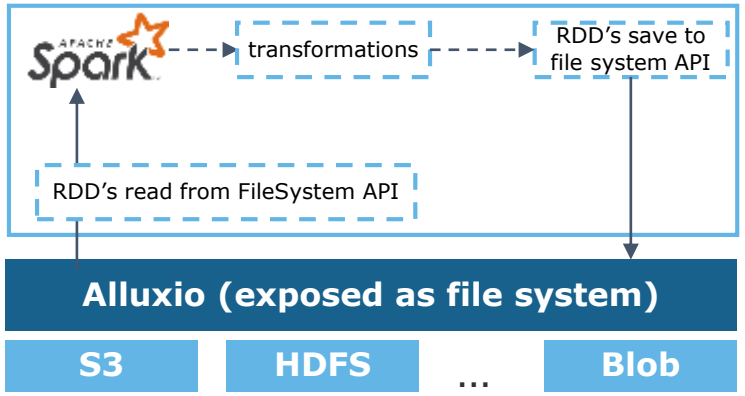


Two ways to integrate with Spark:

- **As stand-alone IGFS or caching layer on HDFS:**
  - Ignite is exposed as HDFS and thus data can be read/written via Spark RDD's File System related APIs
- **As Distributed Cache via IgniteContext:**
  - Provides implementation of Spark RDDs (supporting all transformations and actions)
  - Mutable RDDs (view over distributed cache's content)
  - Configurable lifespan depending upon Ignite's deployment mode

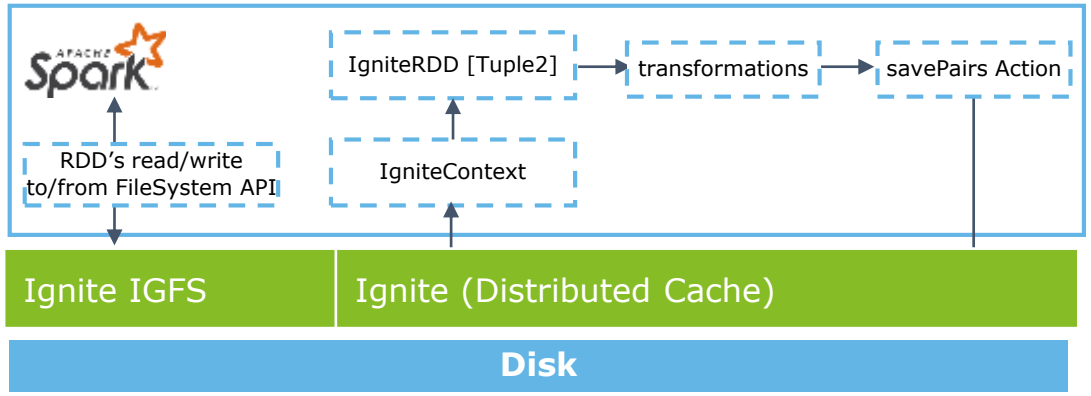
Integration with Spark (Continued)

Alluxio




```
//reading data:
val textRdd =
sc.textFile("alluxio://masternode:19998/path")

//transformations:
val textRdd2=textRdd.filter(_.contains("deloitte"))

//writing data:
textRdd2.saveAsTextFile("alluxio://masternode:19998/desti
nation_path")
```

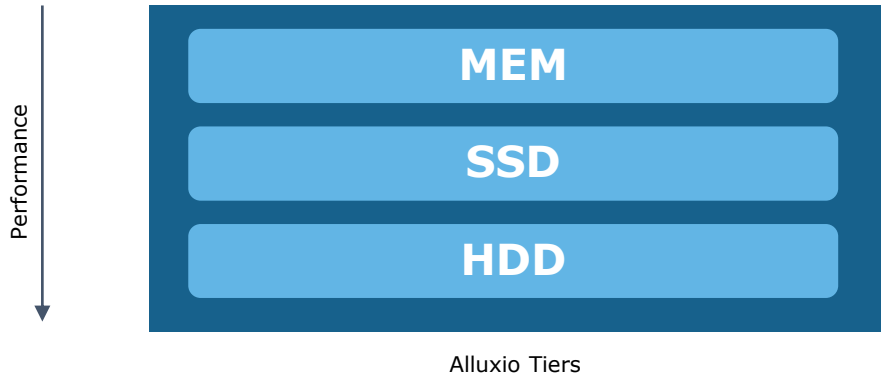
Apache Ignite




```
//creating IgniteContext
val igniteContext = new IgniteContext(sparkContext,() => new
IgniteConfiguration())

//creating IgniteRDD
 val
cacheRdd:org.apache.ignite.spark.IgniteRDD[Integer,String]=
igniteContext.fromCache("deloitte_cache")

//transformations:
val cacheRdd2=cacheRdd.filter(_._2.contains("deloitte"))

//writing data:
cacheRdd2.savePairs()
```

Memory Architecture

Alluxio



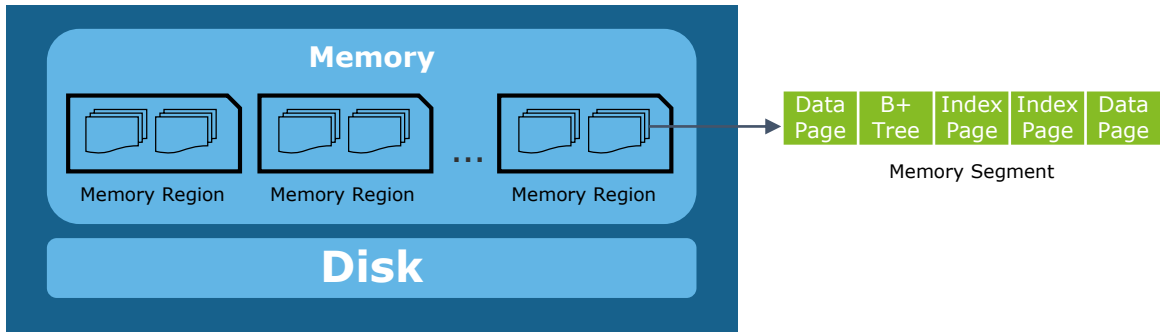
Alluxio Tiers

Alluxio storage is divided into three ordered tiers as follows:

- MEM (memory)
- SSD
- HDD

- Allows to store data greater than the available Memory in cluster
- Automatically manages data between tiers
- Data is written to top tier by default

Apache Ignite



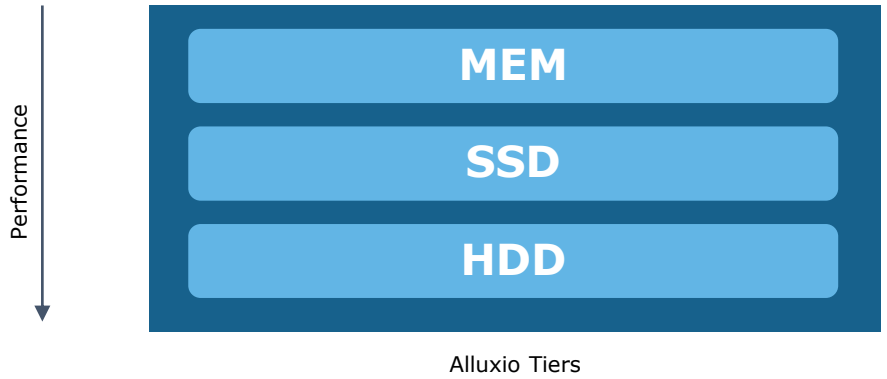
In Native Persistence, data and index storage is divided into:

- Memory (subset of data)
- Disk (superset of data)

- Data can be stored both off-heap and on-heap.
- When stored off-heap, less constraints on volume of data to be stored and less GC pauses
- Memory is further divided into Memory Regions
- Memory Regions consist of Memory Segments which comprise of Data Page, B+ Tree Page, Index Page and FreeList Structures

Advanced Memory Management

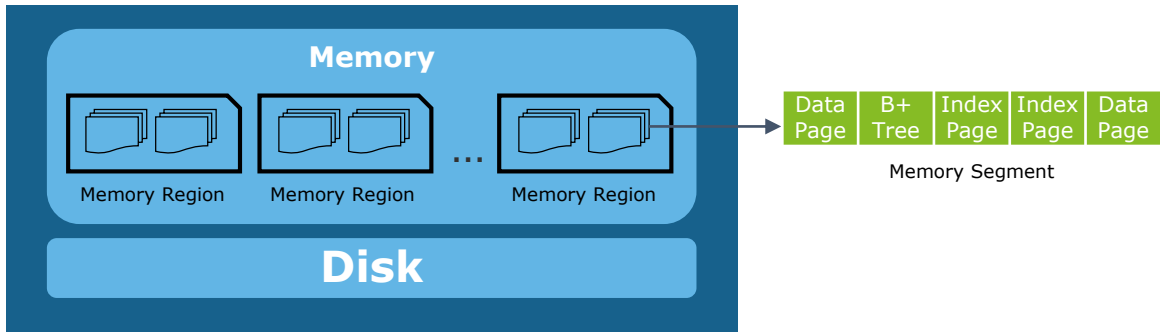
Alluxio



Alluxio Tiers

- **Pinning/Unpinning:**
  To enforce data locality in a specific tier
- **Allocators:**
  For choosing locations to write new data blocks.
- **Evictors:**
  For choosing which data to move to lower tier for freeing space. Supported algorithms: Greedy, LRU, LRFU, Partial LRU
- Evictors and Allocators are applied globally
- Write may fail if space cant be freed or if data exceeds the size of top tier

Apache Ignite



Supports memory policies (e.g. eviction) to be applicable at:
- Memory Region Level (for off-heap caching)
- Entry Level (for on-heap caching)

thus providing more granular control

Eviction:
Supported algorithms for Page Based Eviction:
Random-LRU, Random2-LRU

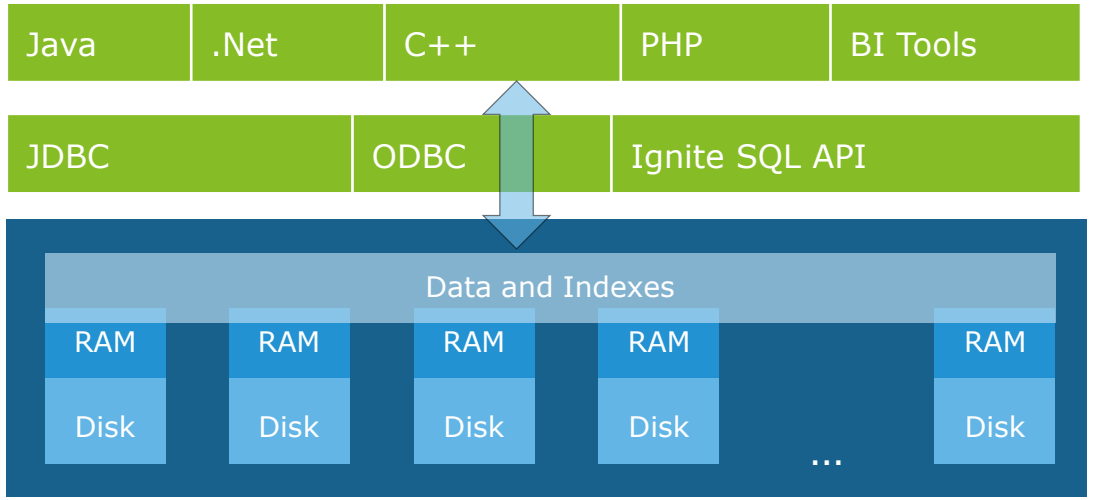
Supported algorithms for Entry Level Eviction:
FIFO, LRU, Random

Additional Capabilities: SQL Support

Alluxio

Not supported

Apache Ignite



- Supports distributed and Horizontally scalable SQL Database capabilities
- Supports indexing
- SQL ANSI-99 compliant
- Supports all SQL DDL and DML commands including UPDATE, DELETE, MERGE queries
    - Resembles Kudu's capabilities
    - Counters limitation of HDFS
- Supports running queries on data spanning on memory or disk. All of the data need not be in memory for processing unlike in Impala or Spark

Additional Capabilities (Continued)

Alluxio

Key Value APIs (not transactional/ACID compliant)

Apache Ignite

Key Value APIs (transactional - ACID compliant)

Compute Grid: Distributed and parallel computation

MLGrid: Machine learning library on top of Apache Ignite. Currently supports limited vector and matrix algebra operations and other algorithms are on the roadmap.

Streaming ingestion: Ingesting real time streams of data into ignite in distributed, scalable and fault tolerant manner

Key Takeaways:

- For inter-process state sharing (e.g. across Spark jobs), both provide adequate functional capabilities.
- Both platforms provide automatic hot data management whereas Apache Ignite provides more granular control courtesy of its per memory region policies.
- For convenience in use-cases involving interacting with data in multiple storage systems at memory speed, **Alluxio** makes more sense.
- For building real-time data and analytics pipelines, Apache Ignite makes more sense as a sink.
- For analytical use-cases involving relational processing and in-place mutation at high speed, Apache Ignite makes more sense.

Questions