



# 高速跨平台大数据存储系统Alluxio (原Tachyon)的特性与案例介绍

顾荣

南京大学 PASA大数据实验室博士研究生,  
Alluxio项目PMC, Maintainer

2016/05/20@南京大数据技术Meetup(南京)

# 内容



- **Alluxio简介**
  - Alluxio是什么
  - Alluxio的发展历程
- Alluxio的系统框架与原理
- Alluxio的重要特性与适用场景
- Alluxio的实际应用案例介绍

# Alluxio是什么



- Alluxio是世界上第一个以内存为中心 (memory-centric) 的虚拟的分布式存储系统。
- Alluxio介于计算框架和现有的存储系统之间，为大数据软件栈带来了显著的性能提升。



# Alluxio (前Tachyon) 的最初诞生

## • 问题

- 传统大数据分析流水线中通过磁盘文件系统（如HDFS）来共享数据成为影响分析性能的瓶颈；
- 大数据计算引擎的处理进程（Spark的Executor，MapReduce的Child JVM等）崩溃出错后，缓存的数据也会全部丢失；
- 基于内存的系统存储数据冗余，对象太多导致Java GC时间过长；

## • 解决

- Alluxio为大数据分析流水线提供内存级数据共享服务
- 内存中的数据存放在Alluxio中，即使计算引擎处理进程崩溃，内存中的数据仍然不会丢失
- 存放在Alluxio内存中的数据不会冗余，同时GC开销大大减小

# Alluxio的发展演变

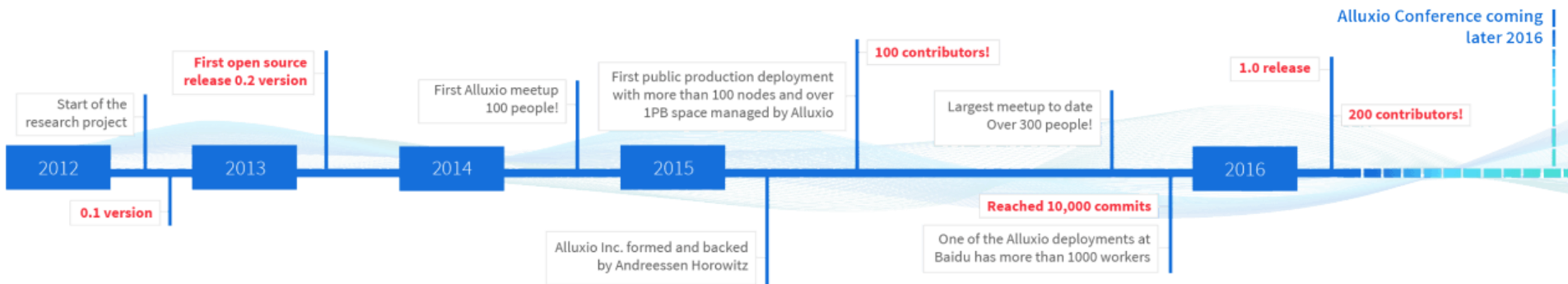
- 如今，Alluxio已发展为一个通用的分布式存储系统，将不同的计算框架和存储系统紧密联系起来。



# Alluxio的发展



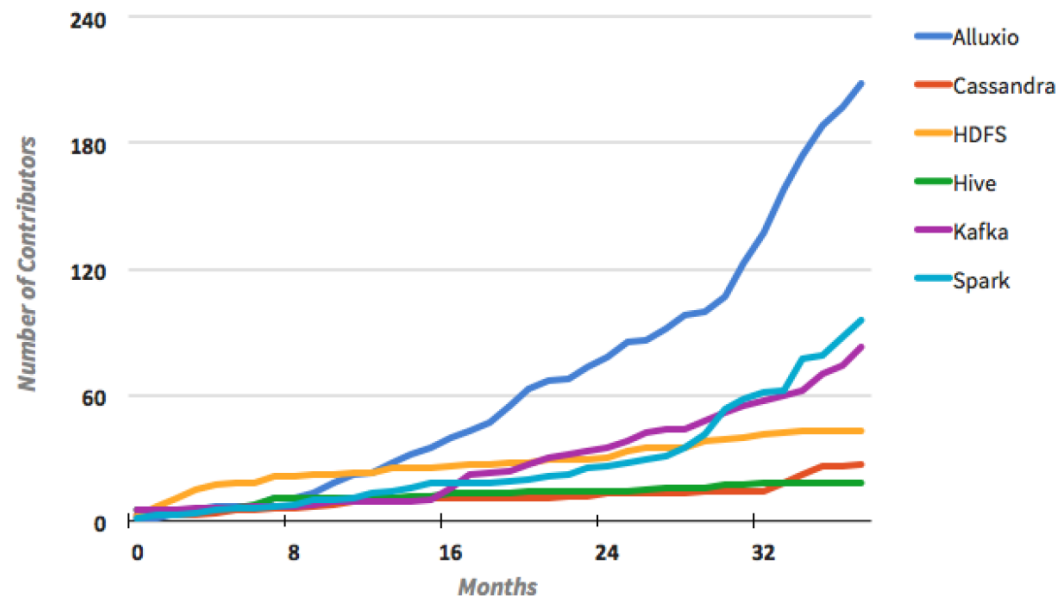
- 2012年12月，Alluxio ( Tachyon ) 发布了第一个版本0.1.0
- 2016年4月，Alluxio的最新发布版本为1.0.1，正在开发1.1.0版本



# Alluxio的发展

- 自2013年4月开源以来，已有超过**50个组织机构**的**200多贡献者**参与到Alluxio的开发中。包括阿里巴巴，Alluxio，百度，卡内基梅隆大学，IBM，Intel，**南京大学**，Red Hat，UC Berkeley和Yahoo。
- 活跃的开源社区

Open Source Contributor Comparison Based on Commit History from Github



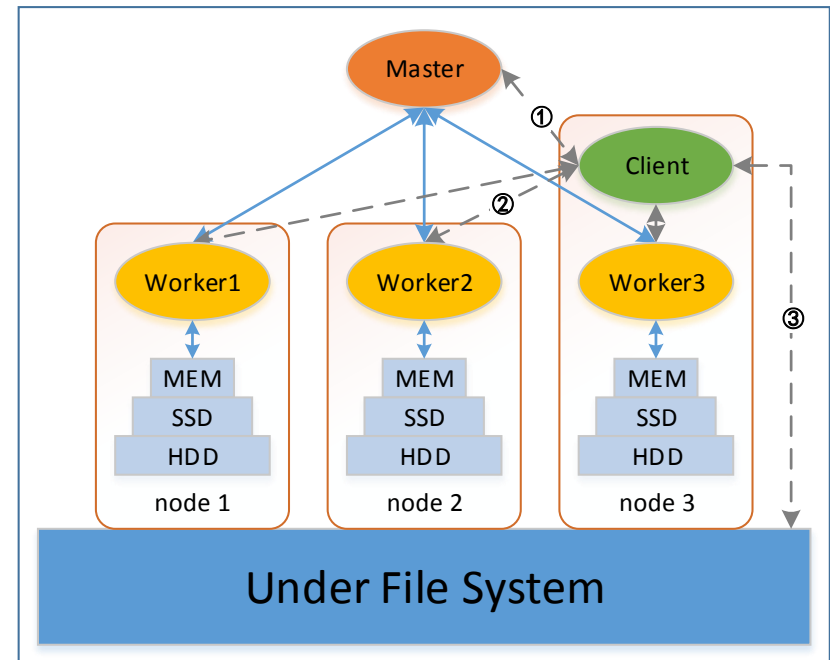
# 内容

- Alluxio简介
- **Alluxio的系统框架与原理**
  - 整体架构
  - 文件组织
  - 读写行为
  - 容错机制
- Alluxio的重要特性与适用场景
- Alluxio的实际应用案例介绍



# Alluxio整体架构

- Master-Worker
  - Master
    - 管理全部元数据
    - 监控各个Worker状态
  - Worker
    - 管理本地MEM、SSD和HDD
- Client
  - 向用户和应用提供访问接口
  - 向Master和Worker发送请求
- Under File System
  - 用于备份



# Alluxio整体架构



- 基本组件

- Master-Worker

- 定时心跳通信，管理系统状态

- Worker内部

- 利用Cache机制，将热数据置于MEM

- Alluxio-UFS

- 定期备份数据

- Client-Alluxio

- ① Client从Master获得文件元数据信息
    - ② 若文件数据在Alluxio中，从本地或远程Worker上读取
    - ③ 若文件数据不在Alluxio中，从UFS上读取

# Alluxio文件组织

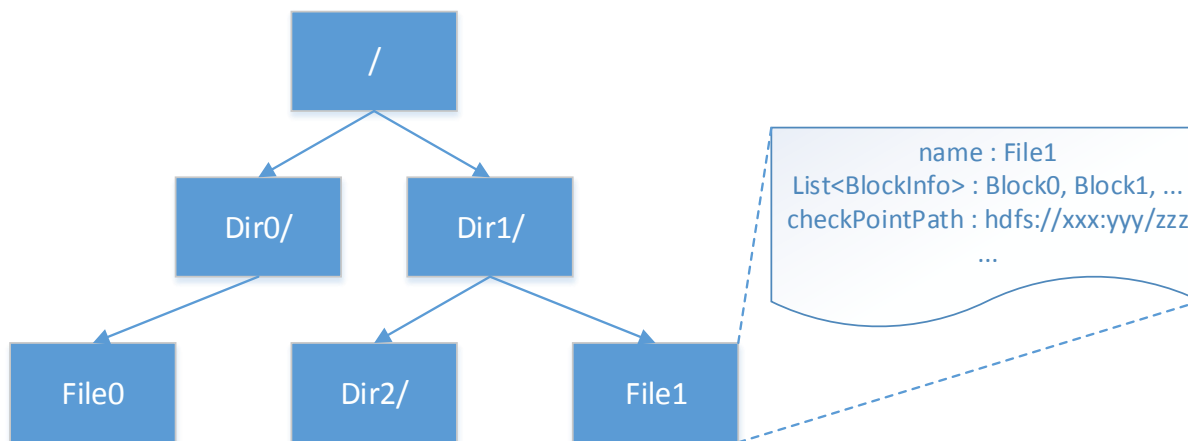
- 元数据为Inode Tree形式

- 树状结构

- 文件和目录都为Inode

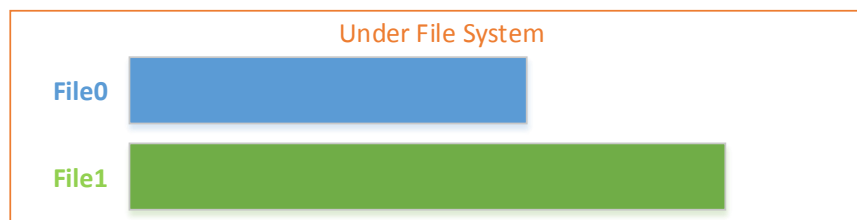
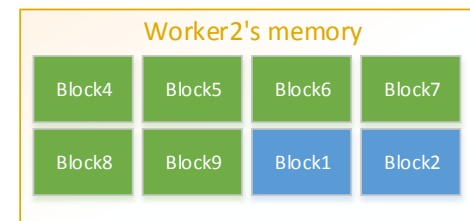
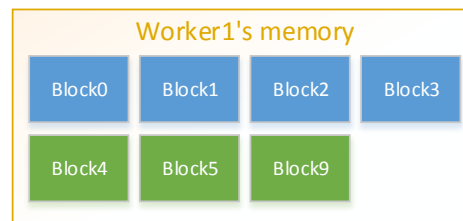
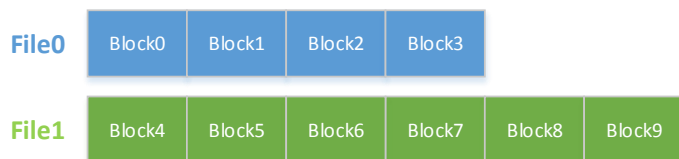
- 文件属性

- 构成Inode的基本信息：id、名称、长度、创建/修改时间等
- 块信息
- 备份信息



# Alluxio文件组织

- Alluxio中，文件数据按块（Block）组织
  - 文件&块信息保存在Master
  - 块数据保存在Worker
- Worker中以块为粒度进行存储和管理
- UFS中以文件为粒度进行存储和管理



# Alluxio读写行为

- 使用读写类型控制数据的存储层位置
  - ReadType --- 控制读数据时的行为
  - WriteType --- 控制写数据时的行为

| 类型               | 取值                    | 含义   |
|------------------|-----------------------|--|
| 读类型<br>ReadType  | CACHE_PROMOTE<br>(默认) | 如果读取的数据块在Worker上时，该数据块被移动到Worker的最高层。如果该数据块不在本地Worker中，那么就将一个副本添加到本地Worker中。 |
|                  | CACHE                 | 如果该数据块不在本地Worker中，那么就将一个副本添加到本地Worker中。                                      |
|                  | NO_CACHE              | 不会创建副本。  |
| 写类型<br>WriteType | CACHE_THROUGH         | 数据被同步地写入到Worker和底层存储系统。  |
|                  | MUST_CACHE<br>(默认)    | 数据被同步地写入到Worker，但不会写入底层存储系统。   |
|                  | THROUGH               | 数据被同步地写入到底层存储系统，但不会写入Worker。   |
|                  | ASYNC_THROUGH         | 数据被同步地写入到Worker，并异步地写入底层存储系统。  |

# Alluxio读写行为

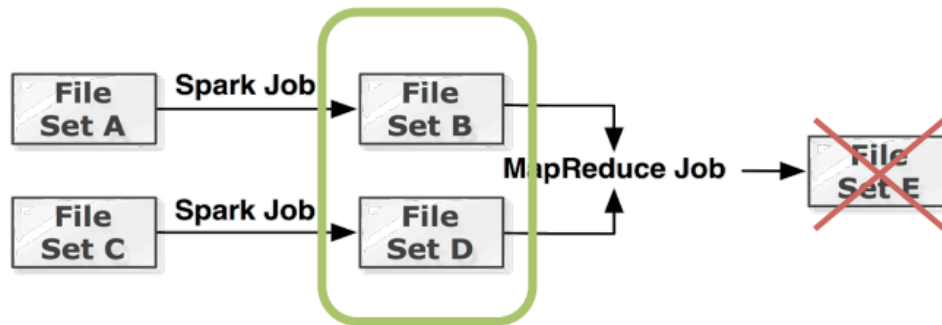


- 使用定位策略控制目标Worker
  - 需要向Worker中写数据块时，决定使用哪个Worker

| 取值                       | 含义  |
|--------------------------|---|
| LocalFirstPolicy<br>(默认) | 首先尝试使用本地Worker，如果本地Worker没有足够的容量容纳一个数据块，那么就从有效的Worker列表中随机选择一个Worker。 |
| MostAvailableFirstPolicy | 使用拥有最多可用容量的Worker。  |
| RoundRobinPolicy         | 以循环的方式选取存储下一个数据块的Worker，如果该Worker没有足够的容量容纳一个数据块，就将其跳过。                |
| SpecificHostPolicy       | 返回指定主机名的Worker。   |

# Alluxio容错机制

- Master
  - 支持使用ZooKeeper启动多个Master
  - 日志 Journal : EditLog + Image
- Worker
  - 由Master监控, 失效时自动重启
- 备份Checkpoint & 世系关系 Lineage



# 内容

- Alluxio简介
- Alluxio的系统框架与原理
- **Alluxio的重要特性与适用场景**
  - 命令行接口
  - 文件系统API
  - Alluxio-FUSE
  - 世系关系API
  - 键值存储库API
  - 分层存储
  - 更多的底层存储系统
  - 统一命名空间
  - 与计算框架相结合
  - 安全性
  - Web界面
  - 配置项设置
  - 度量指标系统
- Alluxio的实际应用案例介绍



# 命令行接口

- 运行方式

- bin/alluxio fs [command]

- cat
    - chmod
    - copyFromLocal
    - copyToLocal
    - fileInfo
    - ls
    - ...
  - mkdir
    - mv
    - rm
    - touch
    - mount
    - unmount

- 路径表示

- alluxio://<master-address>:<master-port>/<path>
  - 支持通配符 \* , 如 : `bin/alluxio fs rm /data/2014\*`

# 命令行接口适用场景

- 方便用户直接进行文件/目录操作，不用编写代码
  - 快速地了解目录结构
  - 预览文件信息
  - 在脚本中操作Alluxio
  - 系统管理者所用
  - ...

# 文件系统API



- 以Java API的方式提供Alluxio的访问接口

- 创建、写文件

```
FileSystem fs = FileSystem.Factory.get();  
AlluxioURI path = new AlluxioURI("/myFile");  
FileOutputStream out = fs.createFile(path);  
out.write(...);  
out.close();
```

- 读文件

```
FileSystem fs = FileSystem.Factory.get();  
AlluxioURI path = new AlluxioURI("/myFile");  
FileInputStream in = fs.openFile(path);  
in.read(...);  
in.close();
```

- 最新版本Java API Doc

( <http://alluxio.org/documentation/master/api/java/> )

- 兼容现有的Hadoop FileSystem接口

- 在MapReduce、Spark等作业中以 "alluxio://" 代替 "hdfs://"

# 文件系统API适用场景

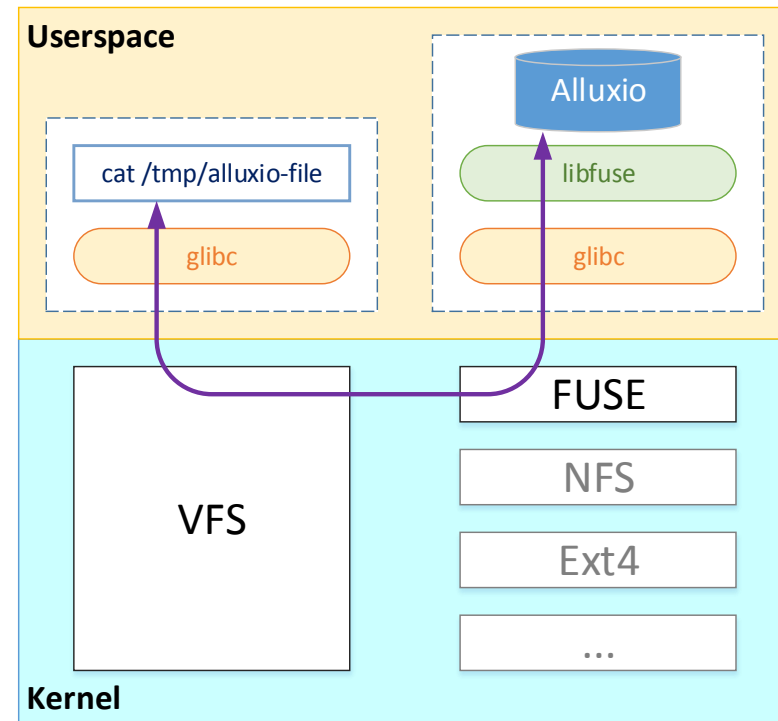


- 在项目、计算任务中细粒度操作Alluxio
  - MapReduce、Spark Job
  - 其他需要分布式文件系统支持的项目
  - 利用Alluxio的内存特性进行加速
  - ...

# Alluxio-FUSE



- 能够在Linux的本地文件系统中挂载Alluxio
  - 利用Linux libfuse功能包
  - 挂载为Linux本地文件系统中的目录
- 方便快捷的使用方式
  - \$ alluxio-fuse.sh mount <dir>
  - 像使用本地文件系统一样使用<dir>
  - 支持的操作
    - open
    - read
    - lseek
    - write



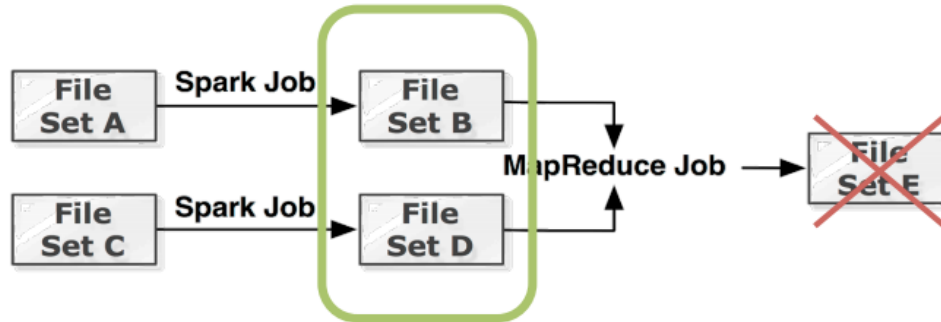
# Alluxio-FUSE适用场景

- 将传统并行计算迁移至Alluxio，利用内存加速
  - CephFS、Glusterfs、Lustre -> Alluxio
- 对单机应用使用多节点内存进行加速
- 为普通用户提供一个大量容量的内存文件系统
- ...

# 世系关系API

- 世系关系 Lineage

- 记录文件之间的世系关系，在数据丢失时通过重计算进行恢复



- 例：构造并创建世系关系

```
AlluxioLineage tl = AlluxioLineage.get();  
List<AlluxioURI> inputFiles = Lists.newArrayList(new AlluxioURI("/inputFile"));  
List<AlluxioURI> outputFiles = Lists.newArrayList(new AlluxioURI("/outputFile"));  
JobConf conf = new JobConf("/tmp/recompute.log");  
CommandLineJob job = new CommandLineJob("my-spark-job.sh", conf);  
tl.createLineage(inputFiles, outputFiles, job);
```

# 世系关系适用场景

- 与现有计算框架相结合
  - 将数据容错从计算层移至存储层
  - 在MapReduce、Spark等不同框架间共享数据时，保证数据可靠性
  - ...
- 对系统本身进行容错
  - 断电后内存数据丢失，但Lineage信息不丢失，能够恢复数据
- ...

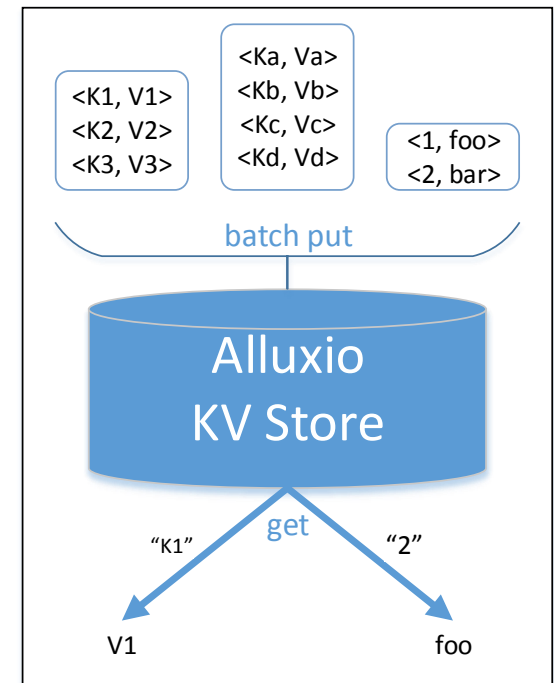


# 键值存储库API

- Alluxio的键值 ( key-value ) 存储功能
  - 创建一个键值存储库并且把键值对放入其中
  - 键值对放入存储后是不可变的
  - 键值存储库完整保存后，打开并使用该键值存储库

## • API样例

```
KeyValueSystem kvs = KeyValueSystem
    .Factory().create();
KeyValueStoreWriter writer = kvs.createStore(
    new AlluxioURI("alluxio://path/my-kvstore"));
writer.put("100", "foo");
writer.put("200", "bar");
writer.close();
KeyValueStoreReader reader = kvs.openStore(
    new AlluxioURI("alluxio://path/kvstore/"));
reader.get("100");
reader.get("300"); //null
reader.close();
```



# 键值存储库适用场景

- 提供结构化数据的存储
  - 键值对结构
  - 表结构
  - 图数据
  - ...
- 优化现有应用
  - 以键值对方式存储中间结果，避免复杂的文件解析过程
  - 为现有的键值应用提供更大的内存存储空间
  - ...

# 分层存储

- 为什么需要多级存储??

- 内存大小有限

- 两个概念

- StorageTier

- 存储层, 对应存储介质, 如内存、SSD、硬盘

- StorageDir

- 数据块存放在Alluxio Worker的本地目录, 通常对应一块磁盘设备
- 一个StorageTier包含一个或多个StorageDir

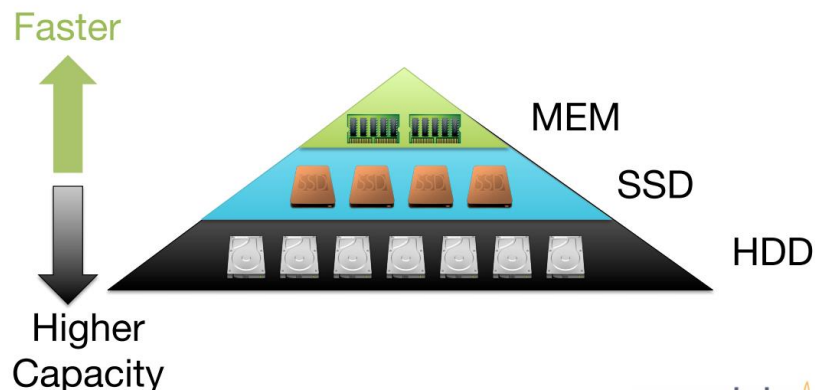
- 数据块管理

- Allocator ---- 选择分配哪个StorageDir里的空间

- GreedyAllocator、MaxFreeAllocator、RoundRobinAllocator

- Evictor ---- 选择撤销哪个StorageDir里的哪些数据块

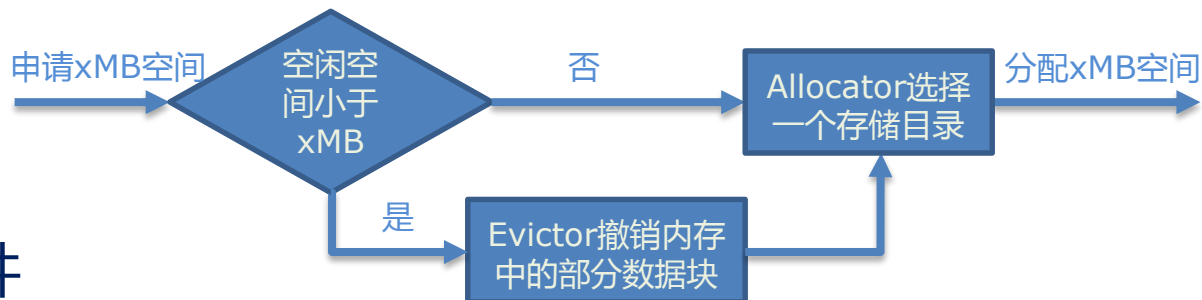
- GreedyEvictor、LRUEvictor、LRFUEvictor、PartialLRUEvictor



# 分层存储

- 写文件

- Client写一个数据块



- 读文件

- Client读一个数据块

- 读类型是CACHE\_PROMOTE，首先将数据块从SSD或者HDD移动到MEM，然后再读取MEM中的数据块
- 读类型不是CACHE\_PROMOTE，从存放该数据块的存储层直接读取数据

- Pining files

- Pining经常访问的文件在内存，永远不会被替换

- Space reserver

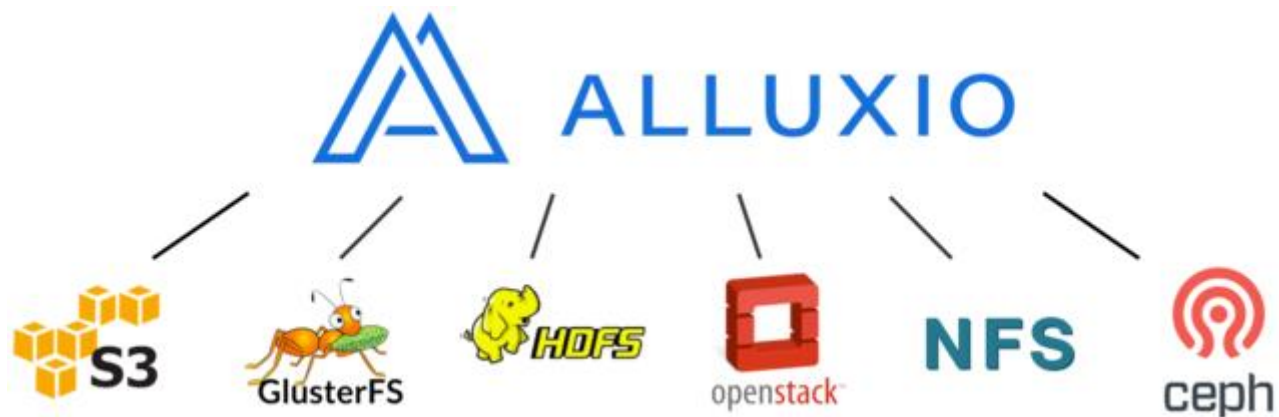
- 保留部分空闲空间 ---- 提升突发性写的性能

# 分层存储适用场景

- 提升大部分应用的I/O性能
  - 在充分利用内存的同时保证了存储容量
  - 不同的分配/替换策略适用于不同的访问模式
  - ...
- 充分利用多种存储设备
  - 除内存外，也能利用SSD进行加速
  - 使用多个存储层目录进行负载均衡
- ...

# 更多的底层存储系统

- 支持不同的底层存储，用于备份
  - GlusterFS、(secure) HDFS、NFS、Amazon S3
  - Aliyun OSS、OpenStack Swift
- 通用的接口，能够扩展更多底层存储系统
  - UnderFileSystemFactory
  - UnderFileSystem



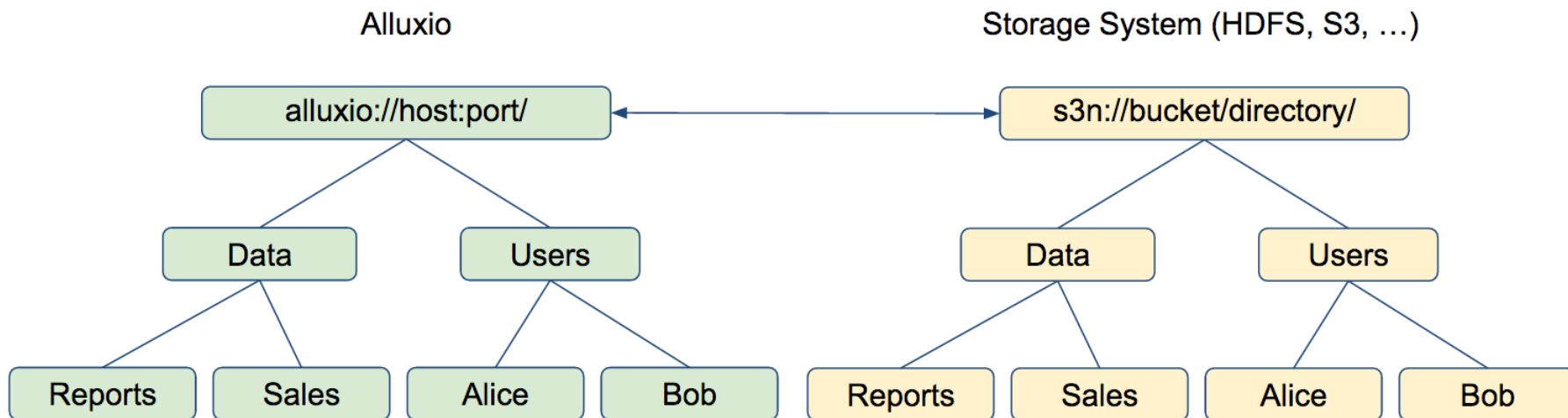
# 底层存储系统的适用场景

- 对Alluxio中的重要数据进行备份
  - 结合ASYNC\_THROUGH的写类型，异步备份数据，既保证了性能，又提供了可靠性
- 将数据迁移至Alluxio
  - 将数据从原先基于磁盘的存储迁移至Alluxio，利用内存加速
- ...

# 统一命名空间

- 透明命名机制

- 保证Alluxio和其底层存储系统的命名空间是一致的
- 用户使用统一路径访问

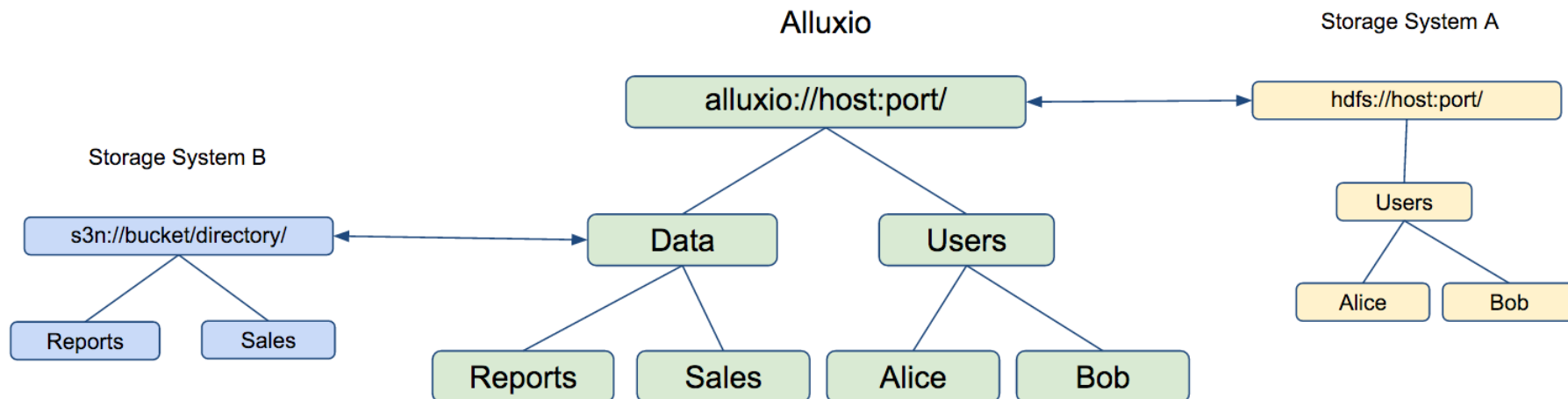




# 统一命名空间

- 统一命名空间

- 能够将多个数据源中的数据挂载到Alluxio中
- 多个数据源使用统一的命名空间
- 用户使用统一路径访问



# 统一命名空间的适用场景

- 将数据迁移至Alluxio
  - 将数据从原先基于磁盘的存储迁移至Alluxio，利用内存加速
  - 在应用中使用统一路径访问Alluxio和底层存储系统
- 管理不同数据源中的数据
  - 将数据从不同数据源迁移至Alluxio，利用内存加速
  - 在应用中使用统一路径访问不同数据源中的数据
  - 实现不同数据源之间的数据共享
- ...

# 与计算框架相结合

- 使用Alluxio作为计算框架的存储系统
  - Spark、Hadoop MapReduce、Flink
  - H2O、Impala、 ...
    - 不需改动现有应用源码
    - 存储路径 “hdfs://ip:port/xxx” -> “alluxio://ip:port/xxx”
  - Zeppelin
    - 默认集成Alluxio，使用Alluxio作为解释器



# 与计算框架结合的适用场景

- 加速大数据应用
  - 充分利用本地内存加速
  - 原有应用能够直接使用Alluxio作为数据源
  - 原有的数据能够被自动迁移至Alluxio
- 在不同计算框架间共享数据
  - 多个应用/平台使用统一的命名空间
- ...

# 安全性



- 安全认证

- Alluxio能够识别访问用户的身份，这是访问权限以及加密等其他安全特性的基础
- 当用户（客户端）连接Alluxio（服务端）时，需要特定的用户、密码或其他认证方式

- 访问权限控制

- 以文件为粒度进行访问权限控制
- 类似POSIX标准的访问权限模型
  - 用户权限、用户组权限、其他用户权限
  - 读r、写w、执行x

# 安全性适用场景

- 保护敏感/隐私数据
  - 对不同用户进行空间隔离
  - 对不同应用/计算框架进行空间隔离
  - ...
- 保证系统本身的可靠性
  - 隔离系统文件和用户文件
  - 普通用户无法访问系统文件
- ...

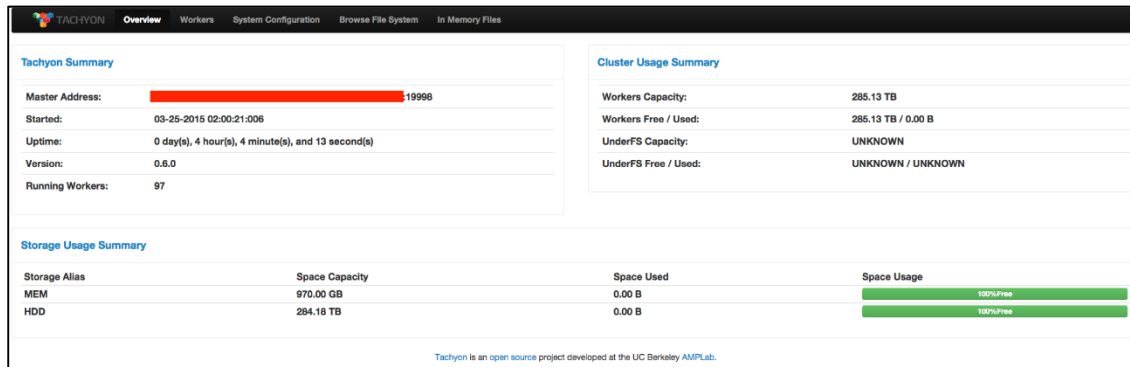
# Web界面

- 以可视化方式便于用户查看和管理
  - Master、Worker的基本运行状态和信息
  - Alluxio系统的配置信息
  - 浏览文件系统
  - 文件内容和文件块信息
  - ...
- 使用浏览器打开
  - Master WebUI [<http://<MASTER IP>:19999>]
  - Worker WebUI [<http://<WORKER IP>:30000>]

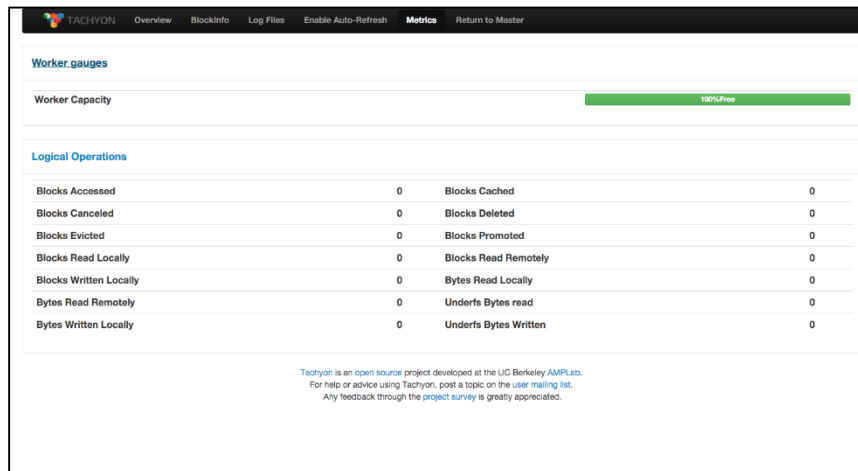
# Web界面



- Master WebUI例



- Worker WebUI例



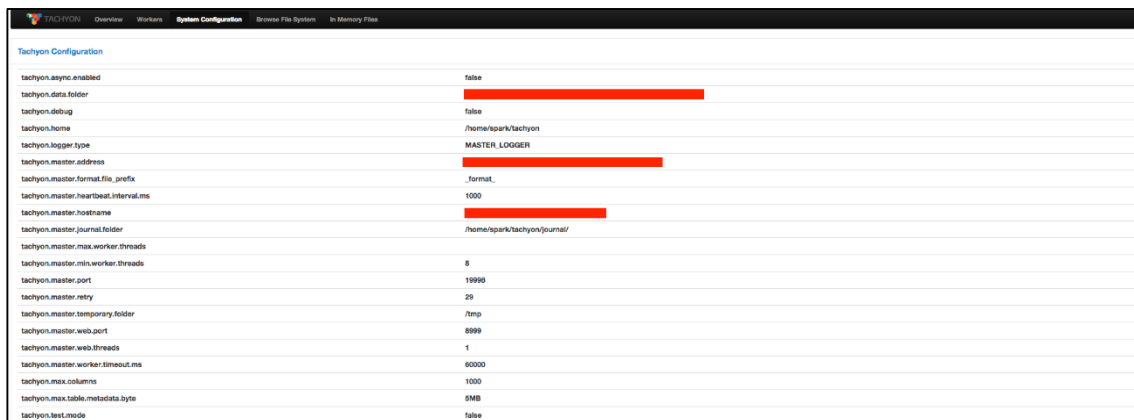


# Web界面的适用场景

- 方便快捷的查看整个系统状态
  - 运行状态，节点个数，系统版本等基本信息
  - 整体容量使用情况
  - 每个Worker的状态，容量使用等
- 浏览文件系统
  - 快速查看文件/目录结构
  - 预览文件内容
  - 下载Alluxio文件到本地
- ...

# 配置项设置

- 配置属性
  - 在alluxio-default.properties文件中的“alluxio.xx.yy”项
  - 共有配置项、Master配置项、Worker配置项、用户配置项、集群管理配置项、安全性配置项
- 系统环境属性
  - 在conf/alluxio-env.sh文件中定义为环境变量
- 在Web界面上查看



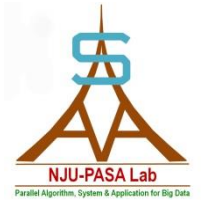
The screenshot shows the Tachyon Configuration web interface. It displays a list of configuration properties and their values. The interface has a dark theme with a navigation bar at the top containing 'TACHYON', 'Overview', 'Workload', 'System Configuration', 'Browse File System', and 'In Memory Files'. The main content area is titled 'Tachyon Configuration' and contains a table of properties.

| Property Name                        | Value                        |
|--------------------------------------|------------------------------|
| tachyon.async.enabled                | false                        |
| tachyon.data.folder                  | [REDACTED]                   |
| tachyon.debug                        | false                        |
| tachyon.home                         | /home/spark/tachyon          |
| tachyon.logger.type                  | MASTER_LOGGER                |
| tachyon.master.address               | [REDACTED]                   |
| tachyon.master.format.file.prefix    | ._format_                    |
| tachyon.master.heartbeat.interval.ms | 1000                         |
| tachyon.master.hostname              | [REDACTED]                   |
| tachyon.master.journal.folder        | /home/spark/tachyon/journal/ |
| tachyon.master.max.worker.threads    | 8                            |
| tachyon.master.min.worker.threads    | 8                            |
| tachyon.master.port                  | 19998                        |
| tachyon.master.retry                 | 20                           |
| tachyon.master.temporary.folder      | /tmp                         |
| tachyon.master.web.port              | 8899                         |
| tachyon.master.web.threads           | 1                            |
| tachyon.master.worker.timeout.ms     | 60000                        |
| tachyon.max.columns                  | 1000                         |
| tachyon.max.table.metadata.byte      | 5MB                          |
| tachyon.test.mode                    | false                        |

# 配置项设置的适用场景

- 各种特性的具体配置
  - 层次化存储
  - 键值存储库
  - 安全性
  - ...
- 应用/系统的调优
  - 块大小
  - 备份方式
  - 连接时长
  - ...

# 度量指标系统



- 实时统计整个Alluxio系统的度量信息
  - 常规信息：集群的整体度量信息（如：CapacityTotal）
  - 逻辑操作：执行的操作数量（如：FilesCreated）
  - RPC调用：每个操作的RPC调用次数（如：CreateFileOps）
- 支持以不同方式显示这些信息
  - ConsoleSink：输出到控制台
  - CsvSink：每隔一段时间将度量指标信息导出到CSV文件中
  - JmxSink：查看JMX控制台中寄存器的度量信息
  - GraphiteSink：给Graphite服务器发送度量信息
  - MetricsServlet：显示到Web UI中

# 度量指标系统



- WebUI中的度量信息

TACHYON Overview Browse File System System Configuration Workers In-Memory Files Log Files Enable Auto-Refresh Metrics

### Master gauges

|                         |          |
|-------------------------|----------|
| Master Capacity         | 100%Free |
| Master Underfs Capacity | 81%Free  |

### Logical Operations

|                     |   |                      |   |
|---------------------|---|----------------------|---|
| Directories Created | 0 | File Block Infos Got | 0 |
| File Infos Got      | 0 | Files Completed      | 0 |
| Files Created       | 0 | Files Freed          | 0 |
| Files Persisted     | 0 | Files Pinned         | 0 |
| New Blocks Got      | 0 | Paths Deleted        | 0 |
| Paths Mounted       | 0 | Paths Renamed        | 0 |
| PathsUnmounted      | 0 |                      |   |

### RPC Invocations

|                  |   |                      |   |
|------------------|---|----------------------|---|
| CompleteFile Ops | 0 | CreateDirectory Ops  | 0 |
| CreateFile Ops   | 0 | DeletePath Ops       | 0 |
| FreeFile Ops     | 0 | GetFileBlockInfo Ops | 0 |
| GetFileInfo Ops  | 0 | GetNewBlock Ops      | 0 |
| Mount Ops        | 0 | RenamePath Ops       | 0 |
| SetState Ops     | 0 | Unmount Ops          | 0 |

Tachyon is an [open source](#) project developed at the UC Berkeley AMPLab.  
For help or advice using Tachyon, post a topic on the [user mailing list](#).  
Any feedback through the [project survey](#) is greatly appreciated.

# 度量指标系统的适应场景

- 让用户深入了解集群上运行的任务
  - I/O密集型
  - I/O是否为瓶颈 ( RPC or 数据传输 )
  - ...
- 进一步分析
  - 将度量数据作为数据源，输入到其他分析型应用/框架中
- ...

# 内容



- Alluxio简介
- Alluxio的系统框架与原理
- Alluxio的重要特性与适用场景
- **Alluxio的实际应用案例介绍**
  - **Barclays Personal and Corporate Bank**
  - 更多使用案例

# Barclays的应用需求

- 数据来源

- 银行事务数据
- 存储在关系型数据库中

- 计算任务

- 批量处理某一时间段的数据
- 多个应用使用同一批数据

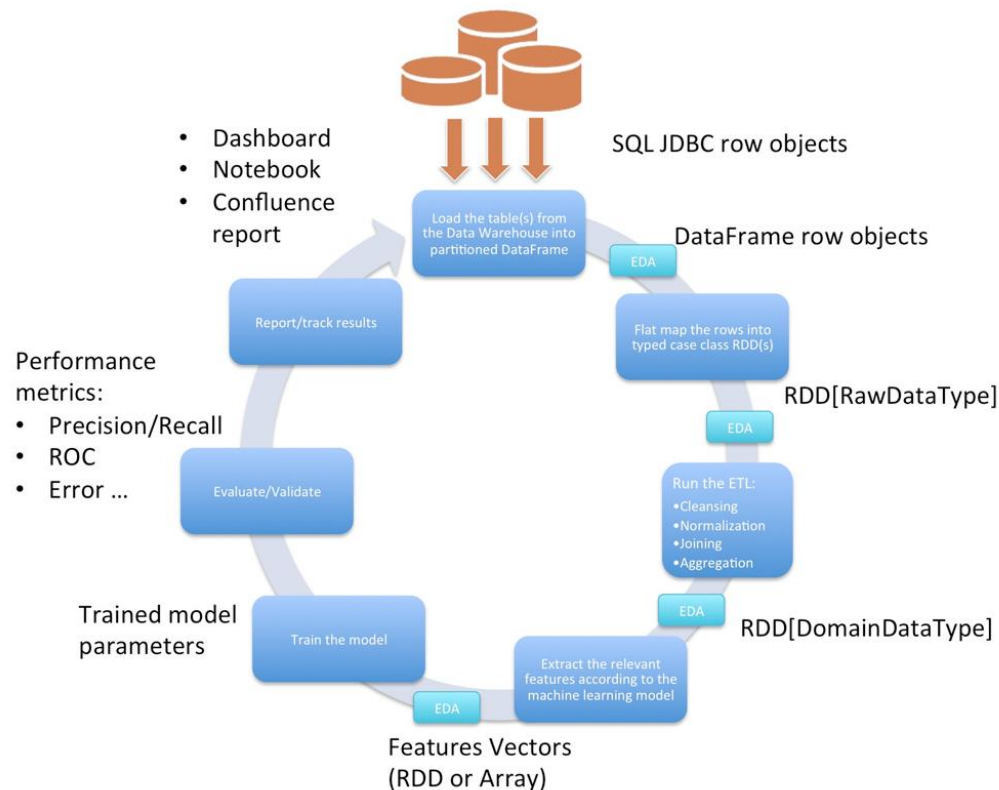
- 时间需求

- 日常应用，每天都要对前一天的数据进行分析



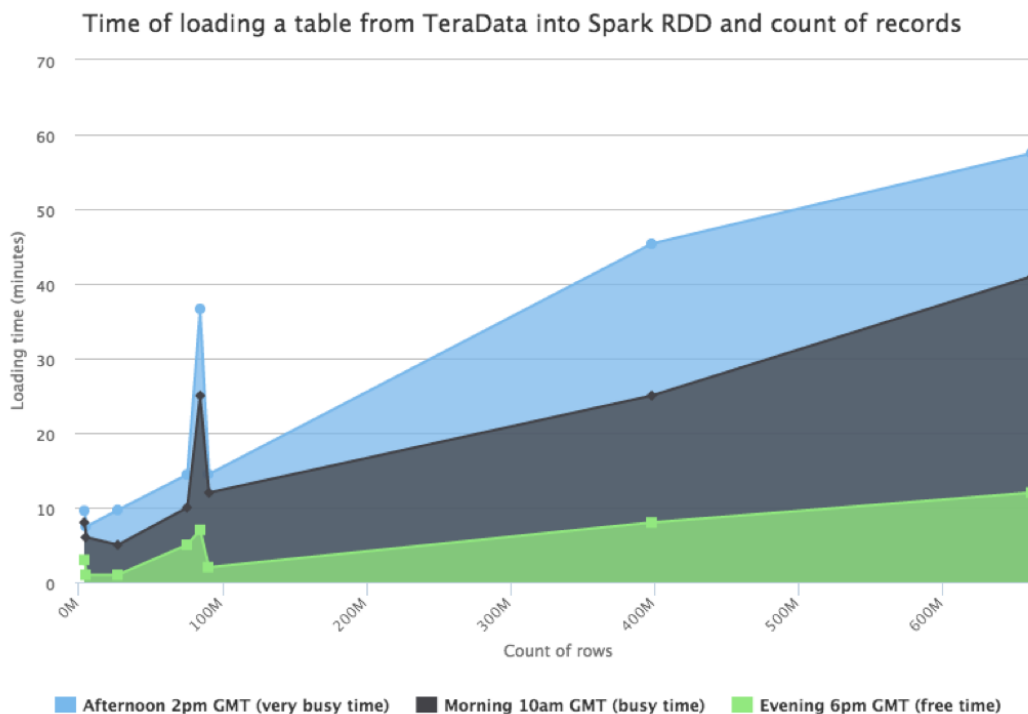
# 原先的解决方案

- 使用Spark进行计算
  - 并行的JDBC从数据库中读取数据
  - 使用DataFrame格式进行计算和分析



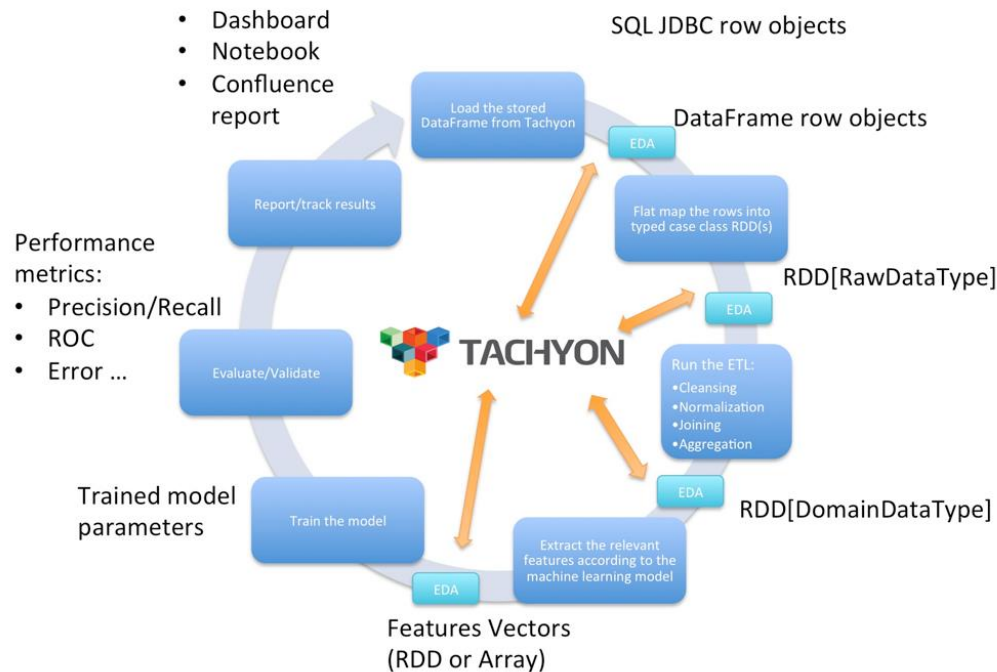
# 原先解决方案的缺陷

- 数据读取过程成为瓶颈
  - 将大规模的数据表从JDBC读入，转换成DataFrame耗时巨大
  - 不同Spark任务间不能共享内存，每个应用都要从JDBC读一遍数据



# 进一步解决方案

- 使用Alluxio ( Tachyon ) 管理数据
  - 输入的大规模数据表存储在Alluxio中，所有应用共享
  - 计算中产生的中间结果也存储在Alluxio中，Spark任务间共享
  - 使用Alluxio对Spark任务本身进行加速



# 示例



- 将DataFrame存储到Alluxio

```
dataframe.write.save("alluxio://master_ip:port/mydata/mydataframe.parquet")
```

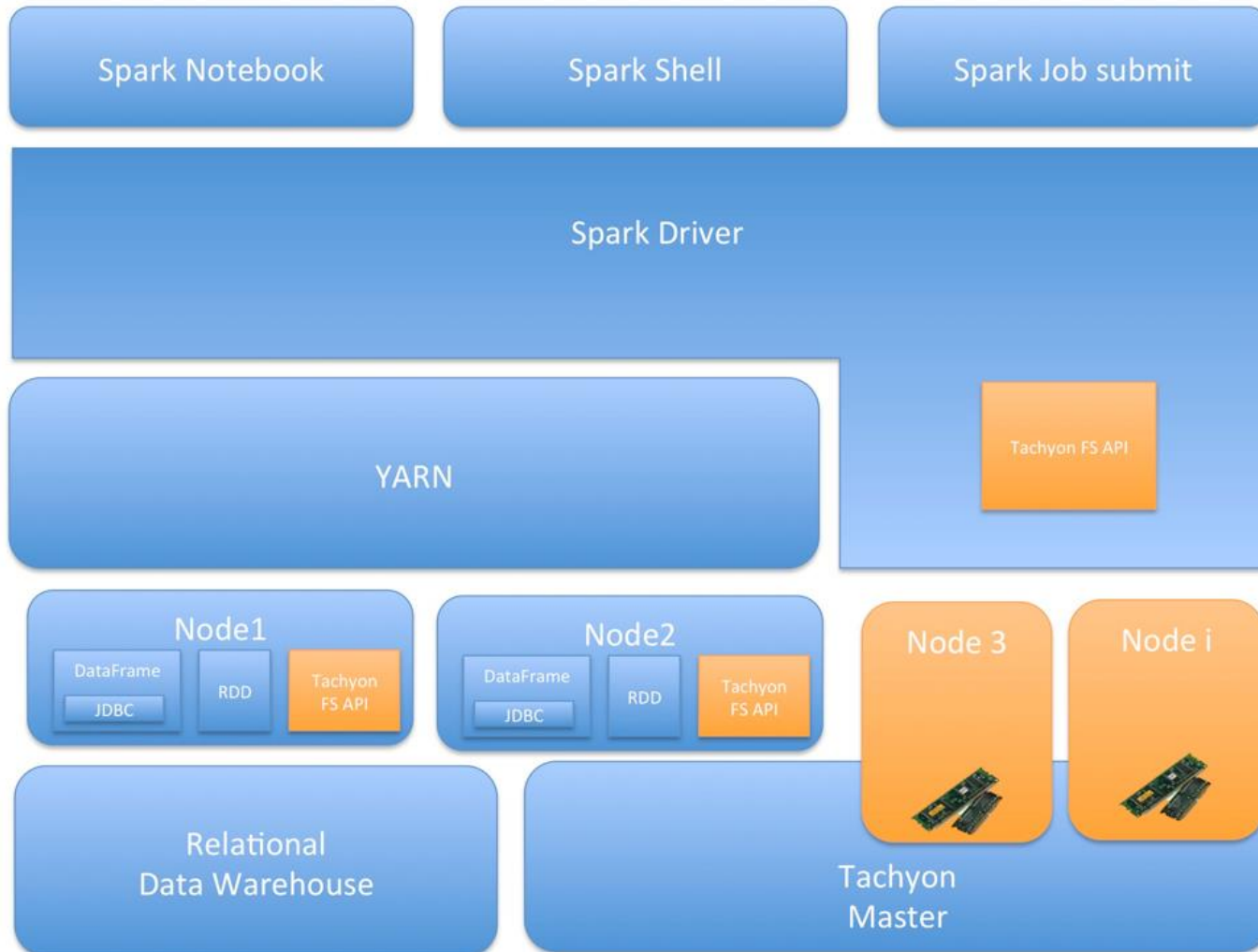
```
val dataframe : DataFrame =  
    sqlContext.read.load("alluxio://master_ip:port/mydata/mydataframe.parquet")
```

- 将Spark RDD存储到Alluxio

```
rdd.saveAsObjectFile("alluxio://master_ip:port/mydata/myrdd.object")
```

```
val rdd : RDD[MyCaseClass] = sc.objectFile[MyCaseClass]  
    ("alluxio://master_ip:port/mydata/myrdd.object")
```

# 完整的解决方案架构



# 在Barclays应用中达到的效果

- 从不可能到可能

工作流的迭代时间**从数小时缩短到了秒级别！**

- 大规模的数据表只需要载入一次，大大降低了从数据库读数据的时间
- 不同应用之间通过内存共享数据，减少了磁盘和网络开销
- 在一天时间内可以运行更多的分析应用

# 更多的使用案例

- Baidu US 查询系统；
- 阿里云OSS整合Alluxio；
- 更多地见[这里](#)



ALLUXIO  
1.1.0-SNAPSHOT

概览 用户指南 ▾

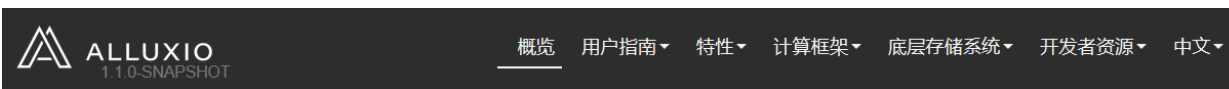
## Powered By Alluxio

目前有许多公司和组织都在使用Alluxio，下面列出了其中一部分。如果你的组织诉我们(project.alluxio@gmail.com)，或者填写[调查](#)让我们知道。

- Adatao
- Alibaba
- Atigeo
- Baidu
- Barclays
- Huawei
- IBM Research
- Intel
- Neusoft
- Alluxio
- UC Berkeley AMPLab
- Ultra Tendency

# 欢迎使用Alluxio中文文档！

- <http://alluxio.org/documentation/master/cn/index.html>



## 概览

Alluxio是世界上第一个以内存为中心的虚拟的分布式存储系统。它统一了数据访问的方式，为上层计算框架和底层存储系统构建了桥梁。应用只需要连接Alluxio即可访问存储在底层任意存储系统中的数据。此外，Alluxio的以内存为中心的架构使得数据的访问速度能比现有常规方案快几个数量级。

在大数据生态系统中，Alluxio介于计算框架(如Apache Spark, Apache MapReduce, Apache Flink)和现有的存储系统(如Amazon S3, OpenStack Swift, GlusterFS, HDFS, Ceph, OSS)之间。Alluxio为大数据软件栈带来了显著的性能提升。以百度为例，使用Alluxio后，其数据处理性能提升了30倍。除性能外，Alluxio为新型大数据应用作用于传统存储系统的数据建立了桥梁。用户可以以独立集群方式(如Amazon EC2)运行Alluxio，也可以从Apache Mesos或Apache YARN上启动Alluxio。

Alluxio与Hadoop是兼容的。这意味着已有的Spark和MapReduce程序可以不修改代码直接在Alluxio上运行。Alluxio是一个已在多家公司部署的开源项目(Apache License 2.0)。Alluxio是发展最快的开源大数据项目之一。自2013年4月开源以来，已有超过50个组织机构的200多贡献者参与到Alluxio的开发中。包括阿里巴巴, Alluxio, 百度, 卡内基梅隆大学, IBM, Intel, 南京大学, Red Hat, UC Berkeley和Yahoo。Alluxio处于伯克利数据分析栈(BDAS)的存储层，也是Fedora发行版的一部分。

[Github](#) | [版本](#) | [下载](#) | [用户文档](#) | [开发者文档](#) | [Meetup 小组](#) | [JIRA](#) | [用户邮件列表](#) | [Powered By](#)

## 现有功能

### 灵活的文件API

Alluxio的本地API类似于 `java.io.File` 类，提供了 `InputStream` 和 `OutputStream` 的接口和对内存映射I/O的高效支持。我们推荐使用这套API以获得Alluxio的最好性能。另外，Alluxio提供兼容Hadoop的文件系统接口，Hadoop MapReduce和Spark可以使用Alluxio代替HDFS。

### 可插拔的底层存储

在容错方面，Alluxio备份内存数据到底层存储系统。Alluxio提供了通用接口以简化插入不同的底层存储系统。目前我们支持Amazon S3, OpenStack Swift, Apache HDFS, GlusterFS以及单节点本地文件系统，后续也会支持很多其它的文件系统。

### 层次化存储

通过分层存储，Alluxio不仅可以管理内存，也可以管理SSD和HDD，能够让更大的数据集存储在Alluxio上。数据在不同层之间自动被管理，保证热数据在更快的存储层上。自定义策略可以方便地加入Alluxio，而且pin的概念允许用户直接控制数据的存放位置。

### 统一命名空间

Alluxio通过挂载功能在不同的存储系统之

### 世系(Lineage)

通过世系(Lineage)，Alluxio可以不受容错

### 网页UI & 命令行

用户可以通过网页UI浏览文件系统。在调



# The End & Thank you!



项目主页：<http://alluxio.org/>

个人微博：顾荣\_NJU

电子邮箱：[gurongwalker@gmail.com](mailto:gurongwalker@gmail.com)