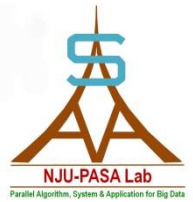
基于Alluxio提升Spark和Hadoop HDFS的系统性能与稳定性

顾 荣 博士
南京大学 计算机系 助理研究员,
Alluxio项目PMC, Maintainer

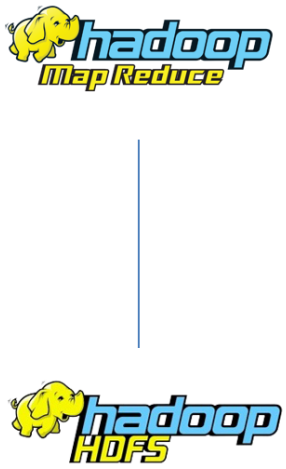
2017/03/25@China Hadoop Summit 2017(北京)

内容



- **Alluxio基本原理回顾与1.4的最新特性介绍**
- 基于Alluxio的Spark DataFrame/RDD性能调优
- 基于Alluxio提升HDFS集群的性能和SLA稳定性
- 总结

BIG DATA ECOSYSTEM Yesterday

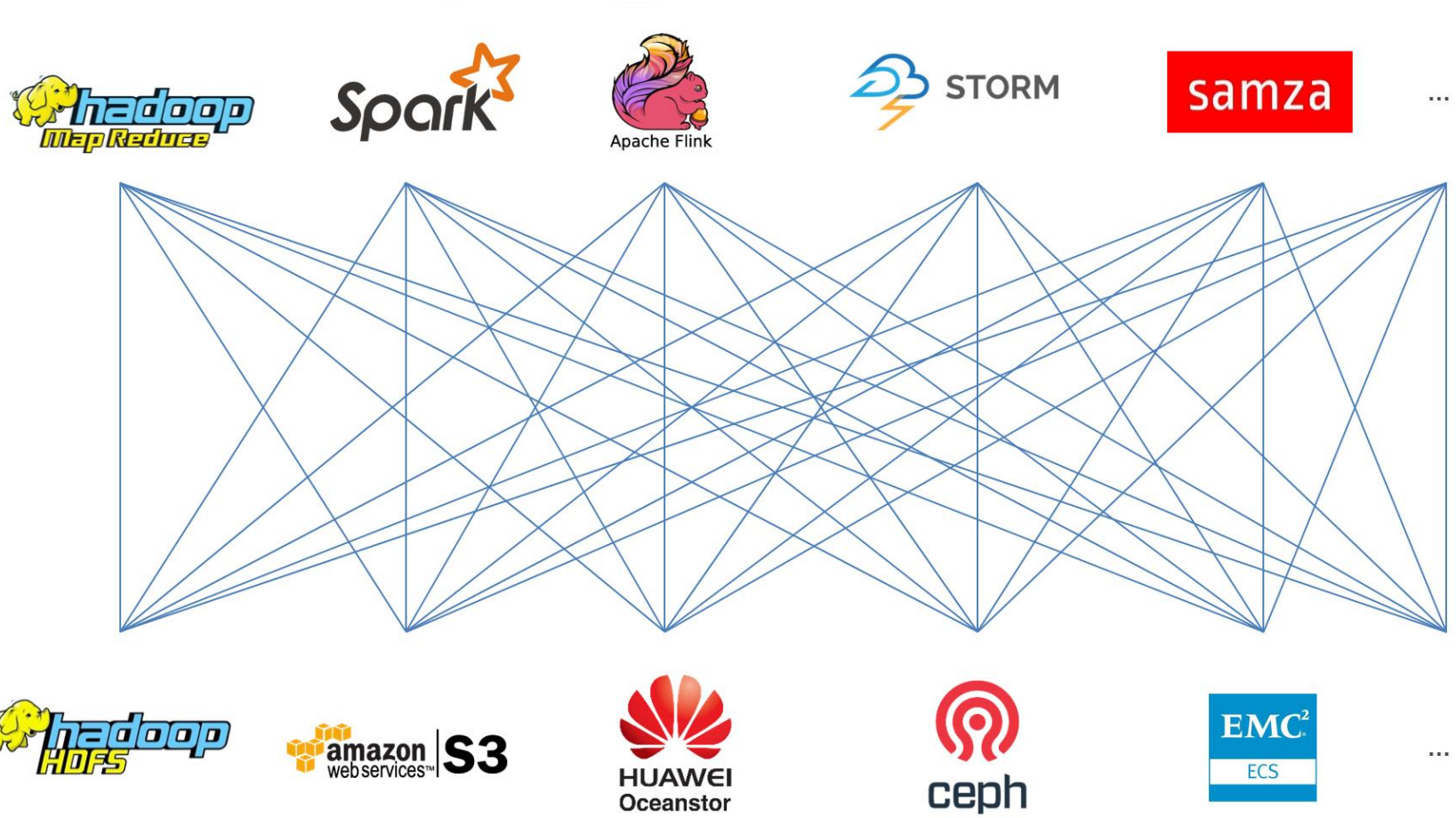


BIG DATA ECOSYSTEM Today

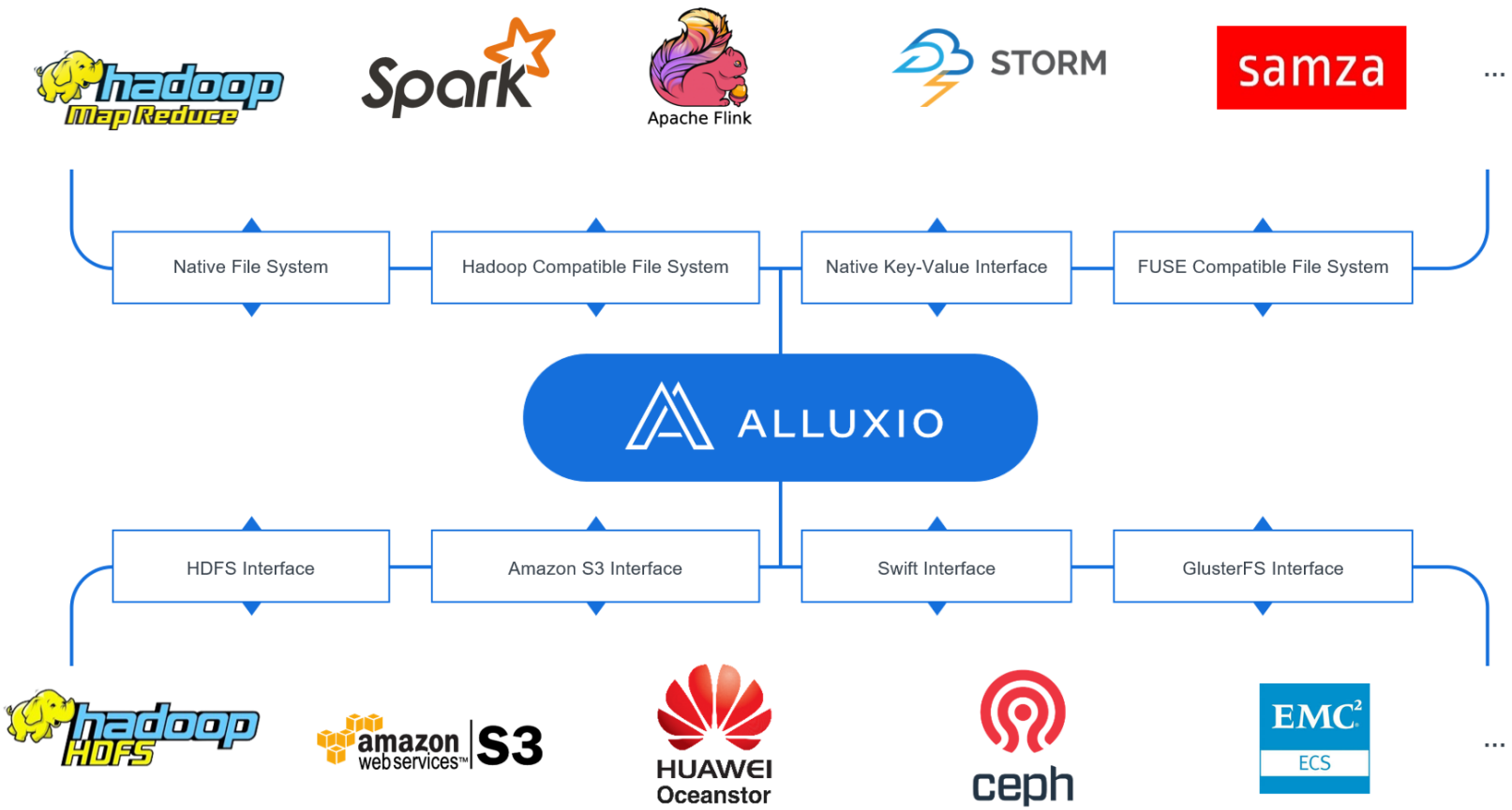
...

...

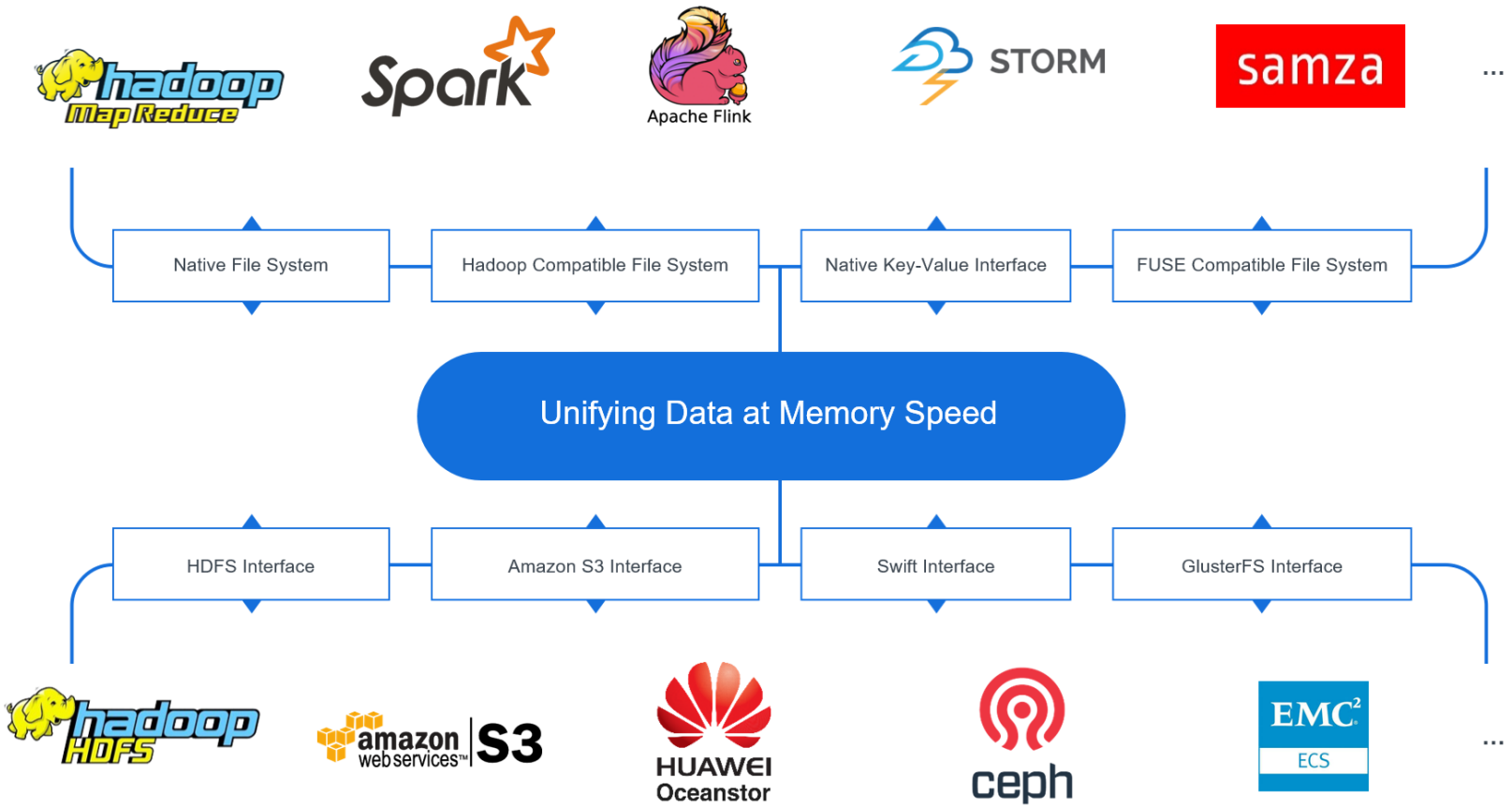
BIG DATA ECOSYSTEM Issue



BIG DATA ECOSYSTEM With Alluxio



BIG DATA ECOSYSTEM With Alluxio

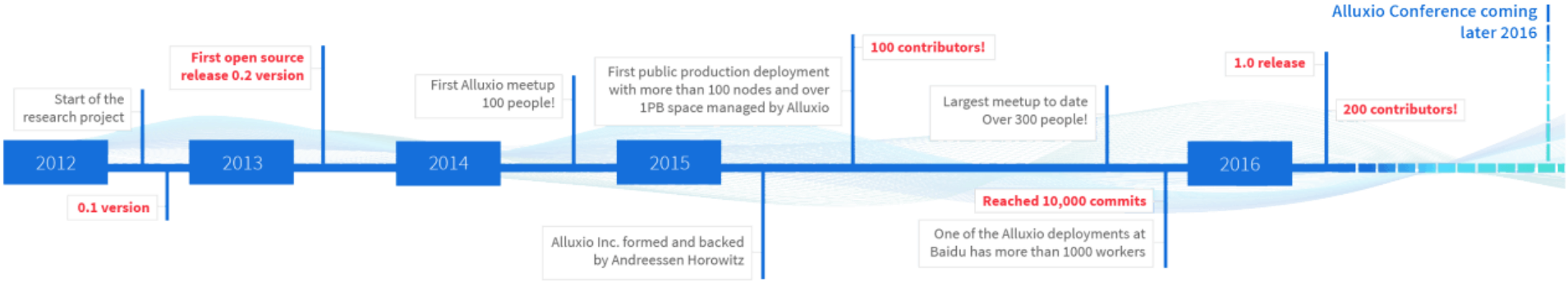


Alluxio是什么

- 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。

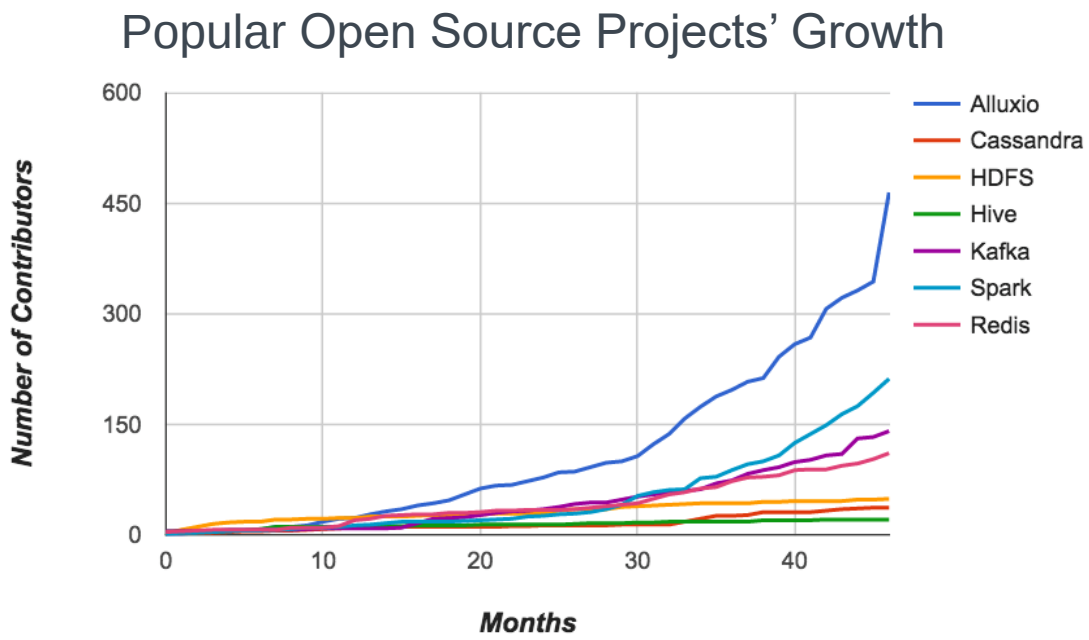
Alluxio的发展

- 2012年12月，Alluxio（Tachyon）发布了第一个版本0.1.0
- 2017年1月，Alluxio的最新发布版本为1.4

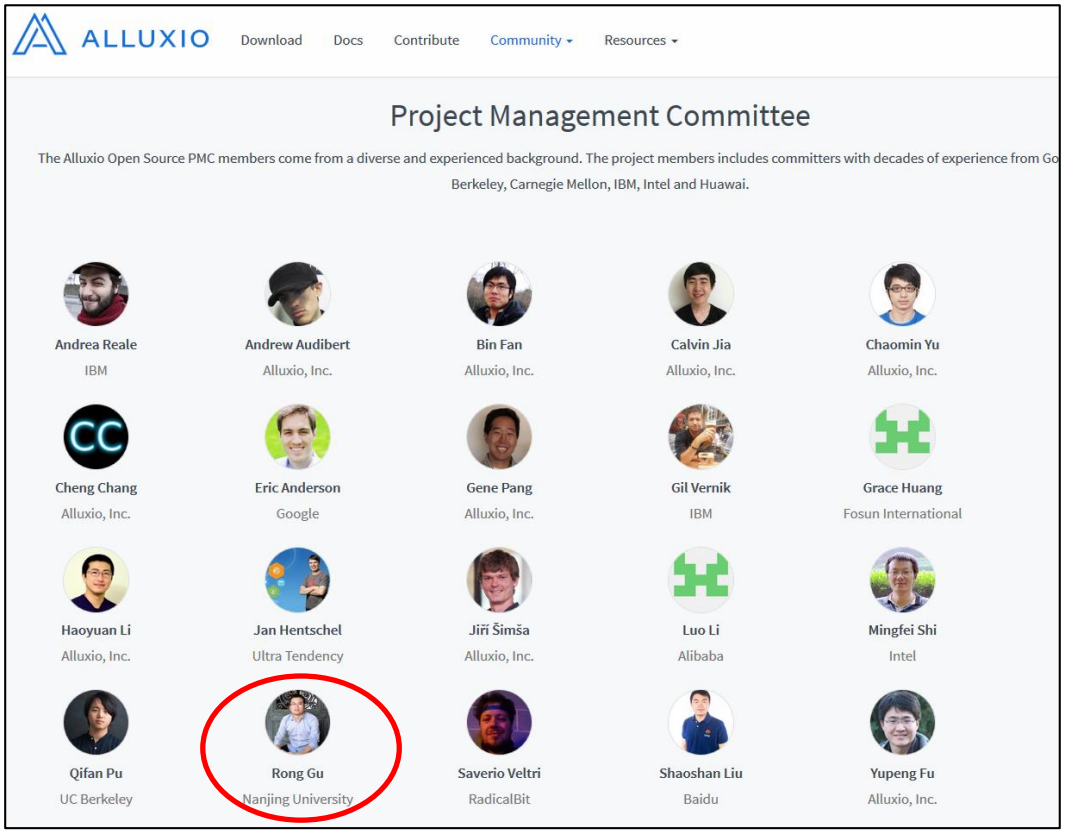


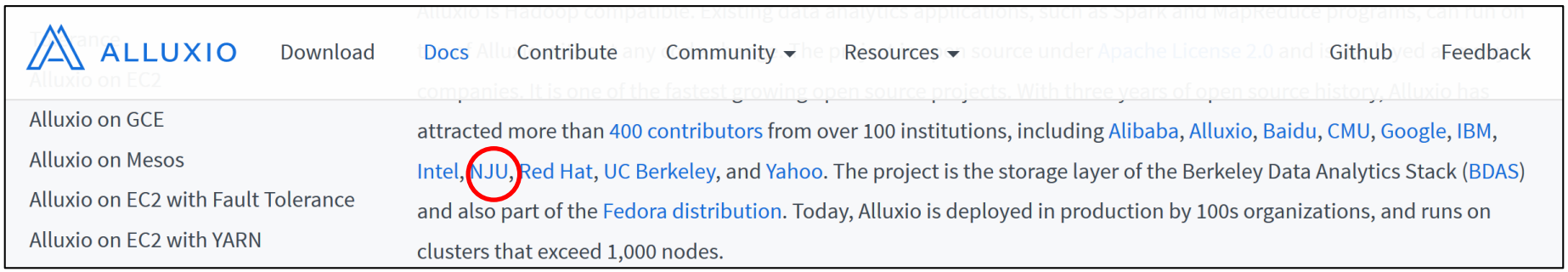
Alluxio的发展

- 自2013年4月开源以来，已有超过100个组织机构的400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

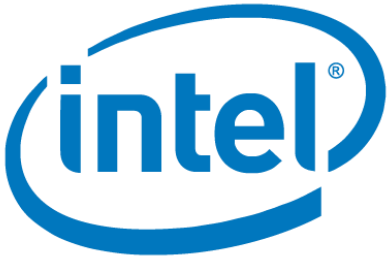


关于南大PASALab和我的贡献情况





INDUSTRY ADOPTION



The Register®
Biting the hand that feeds IT

Data Center ▸ Storage

Multi-silo data-sucker Alluxio inks deal with Dell EMC

EMC² ECS

Follows startup's agreement with Huawei

HUAWEI

Huawei and Alluxio Jointly Release Big Data Storage Acceleration Solution, Speeding Up Big Data Application Popularization

DATA + FOLLOW THIS TOPIC

8 data trends on our radar for 2017

From deep learning to decoupling, here are the data trends to watch in the year ahead.

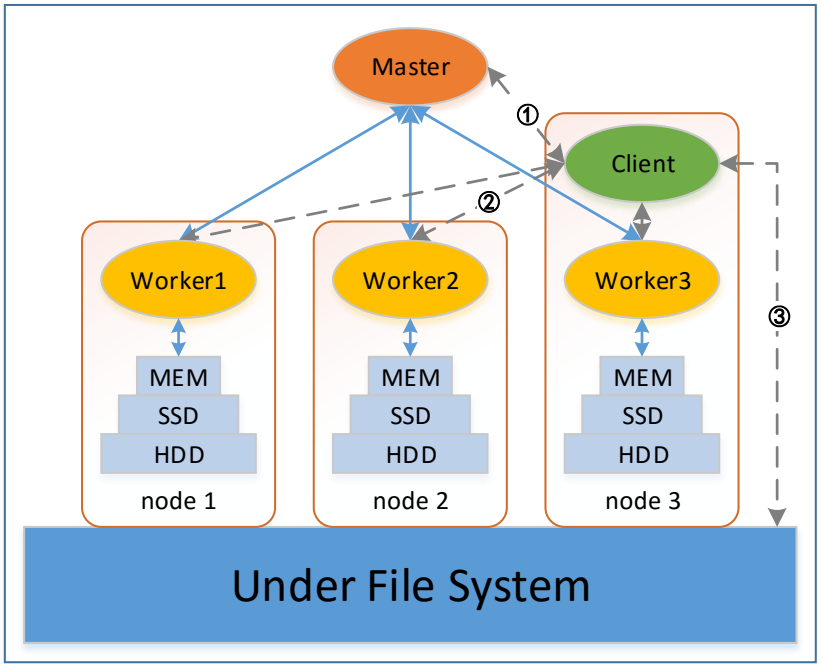
By Ben Lorica, January 3, 2017

6. The decoupling of storage and computation will accelerate.

The UC Berkeley AMPLab project ended last November, but the team behind Apache Spark and Alluxio are far from the only ones to highlight the separation of storage and computation. As noted above, popular object stores in the cloud and even some recent deep learning architectures emphasize this pattern.

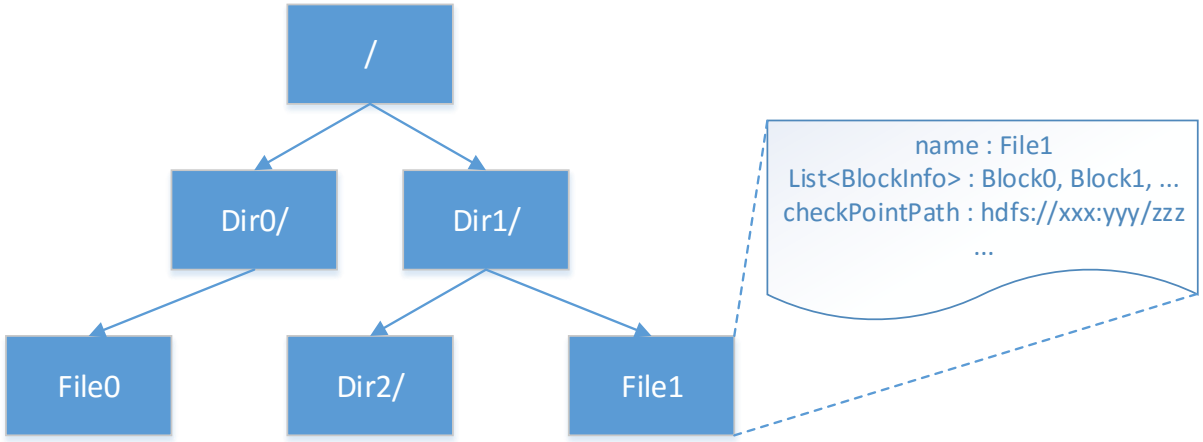
Alluxio整体架构

- 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文件组织

- 元数据为Inode Tree形式
  - 树状结构
  - 文件和目录都为一个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



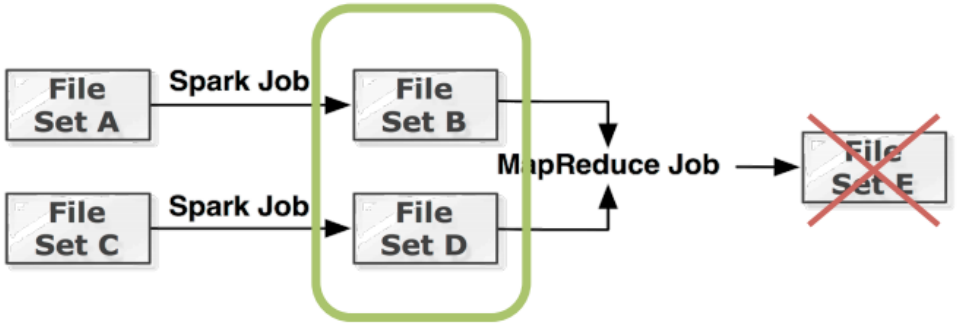
Alluxio读写行为

- 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 ReadType	CACHE_PROMOTE（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 WriteType	CACHE_THROUGH	数据被同步地写入到Worker和底层存储系统。
	MUST_CACHE（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



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*`

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");
FileOutStream out = fs.createFile(path);
out.write(...);
out.close();
```

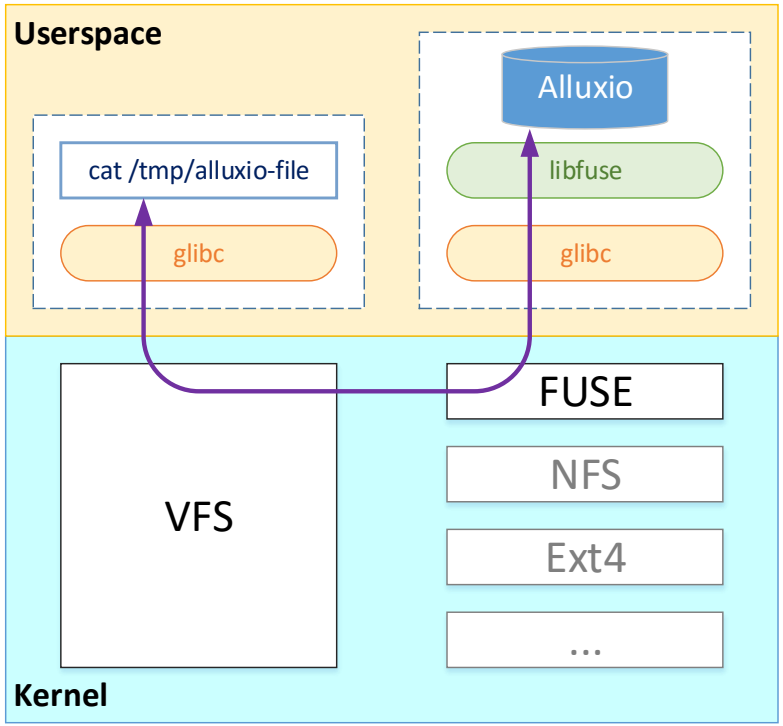
  - 读文件

```
FileSystem fs = FileSystem.Factory.get();
AlluxioURI path = new AlluxioURI("/myFile");
FileInStream 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://”

Alluxio-FUSE

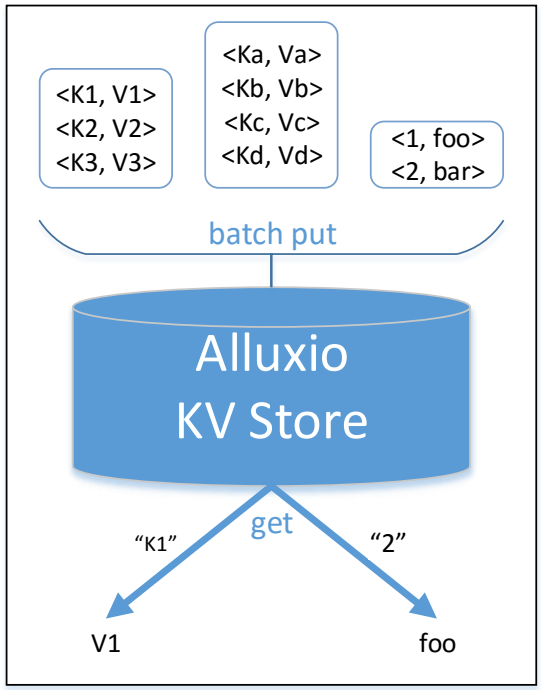
- 能够在Linux的本地文件系统中挂载Alluxio
  - 利用Linux libfuse功能包
  - 挂载为Linux本地文件系统中的一个目录
- 方便快捷的使用方式
  - $ alluxio-fuse.sh mount <dir>
  - 像使用本地文件系统一样使用<dir>
  - 支持的操作
    - open
    - read
    - lseek
    - write



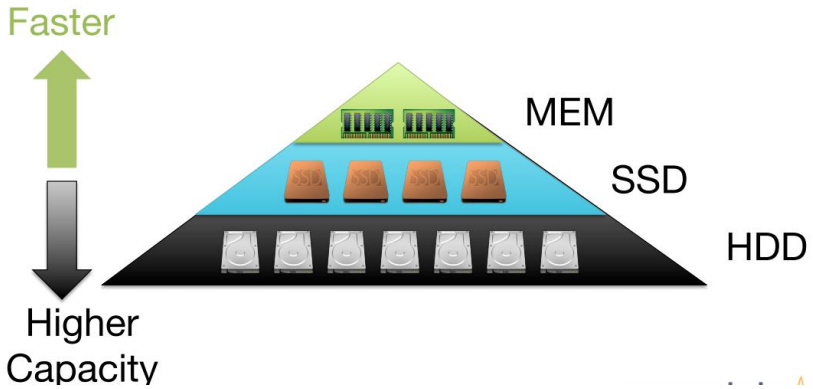
键值存储库API

- 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();
```



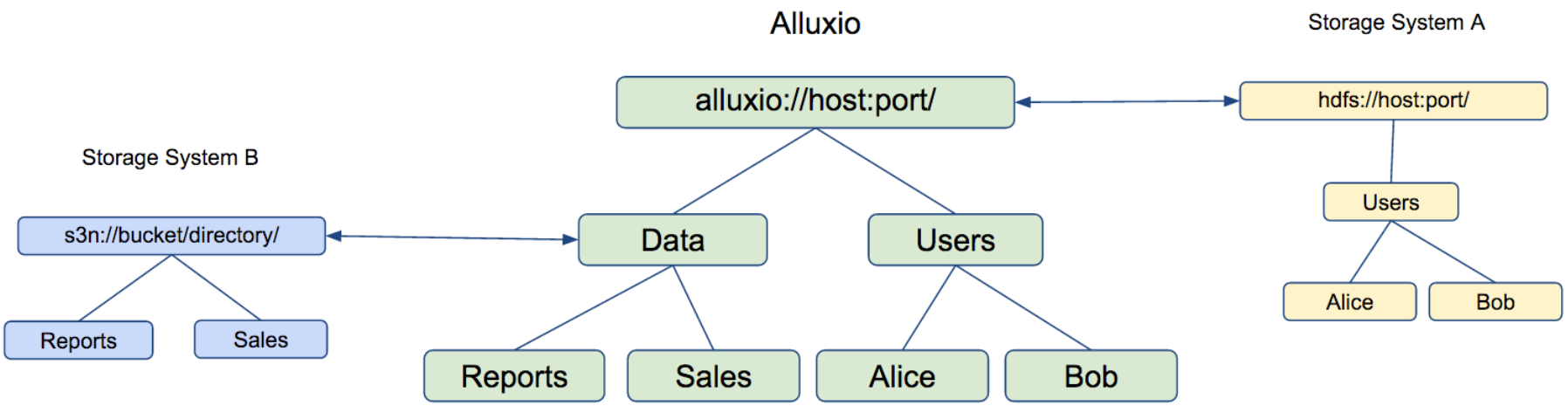
分层存储



- 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

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

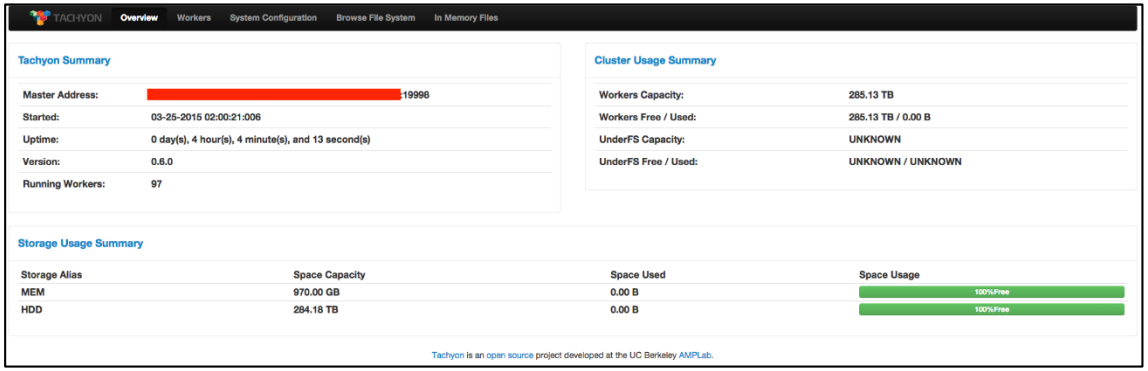
- 使用Alluxio作为计算框架的存储系统
  - Spark、Hadoop MapReduce、Flink
  - H20、Impala、…
    - 不需改动现有应用源码
    - 存储路径 “hdfs://ip:port/xxx” -> “alluxio://ip:port/xxx”
  - Zeppelin
    - 默认集成Alluxio，使用Alluxio作为解释器

安全性

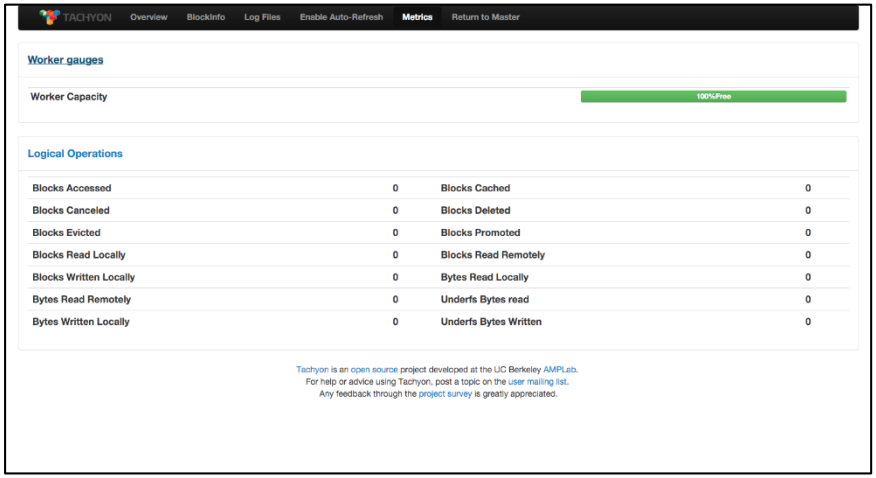
- 安全认证
  - Alluxio能够识别访问用户的身份，这是访问权限以及加密等其他安全特性的基础
  - 当用户（客户端）连接Alluxio（服务端）时，需要特定的用户、密码或其他认证方式
- 访问权限控制
  - 以文件为粒度进行访问权限控制
  - 类似POSIX标准的访问权限模型
    - 用户权限、用户组权限、其他用户权限
    - 读r、写w、执行x

Web界面

- Master WebUI例



- Worker WebUI例



技术特点小结

- *Co-located compute and data with memory-speed access to data*
- *Virtualized different storage systems under a unified namespace*
- *Scale-out architecture*
- *File system API, software only*

系统优势小结

Unification

New workflows across any data in any storage system

Performance

Orders of magnitude improvement in run time

Flexibility

Choice in compute and storage – grow each independently, buy only what is needed

Alluxio 1.4版本的重要新特性介绍

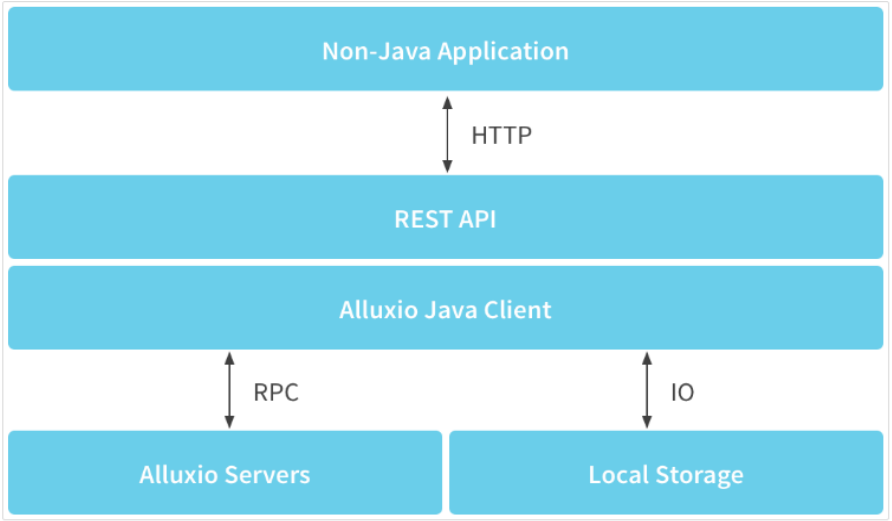
- 优化Alluxio底层对象存储API
  - 优化的对象存储连接器
    - Alluxio 1.4.0在对象存储上的小文件和小规模读取的性能方面有了重大改进。改进UFS API提升了对象存储中获取元数据的性能。

	创建	删除
S3A	5 倍提升，与 v1.3 比较	3 倍提升，与 v1.3 比较
Ceph	10 倍提升，与 v1.3 比较	15 倍提升，与 v1.3 比较

  - 简化的对象存储集成
    - 如今集成新的对象存储只需要原先实现的方法中的一半即可。
    - 在源代码行数方面，现在仅需要不到400行代码即可完成新的对象存储的集成，不到原来的一半。

Alluxio 1.4版本的重要新特性介绍

- 文件系统原生REST接口
  - 新引入的REST接口提供了同Alluxio native Java API的对等性，其目的是促进非Java环境与Alluxio的交互。
    - REST接口通过Alluxio代理进程实现，该进程使用一个内部Alluxio Java客户端代理REST接口和Alluxio服务器之间的通信
    - 为了获得最佳性能，推荐将Alluxio代理与Alluxio服务进程放在一起。这可以让非java应用以内存级的速度访问Alluxio中的数据，同时最小化Alluxio代理和Alluxio服务之间额外网络的开销。



Alluxio 1.4版本的重要新特性介绍

- 数据包流（Packet Streaming）
  - Alluxio 1.4.0引入了一种新的网络传输协议，旨在充分利用Alluxio组件之间的可用网络带宽
    - 在标准网络中高达2倍的IO性能改进，以及在高延迟吞吐量产品环境下取得更好的结果
  - 减少了网络传输期间使用的缓存，并且依赖于使用连续流传输协议取代数据传输中的请求-响应协议
    - 通过使用这种方法，能确保网络管道持续饱和，因为我们不需要发送周期性的额外数据请求

Alluxio 1.4版本的重要新特性介绍

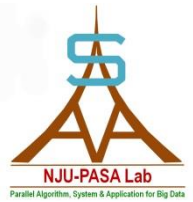
- 支持Apache Hive (Contributed By PASALab)
  - 将Apache Hive集成运行在Alluxio上,具体看使用文档 (http://www.alluxio.org/docs/master/en/Running-Hive-with-Alluxio.html)

- 提升了与基于YARN的应用的集成工作
  - 支持在用户/权限打开的模式下将YARN的应用运行在Alluxio上

Alluxio 1.4版本的重要新特性介绍

- 提升了Alluxio Master中的锁管理机制
  - 使得基于MapReduce的计算框架能够有更高并发的元数据操作性能
  - 对缓慢底层文件系统进行元数据操作（例如在云存储中进行重命名）的性能能够提升1个数量级左右
- 指定层级写操作
  - 用户能够将数据显式地写到Alluxio存储层级中的指定层，而不是原先默认的最顶层

内容



- Alluxio基本原理回顾与1.4的最新特性介绍
- **基于Alluxio的Spark DataFrame性能调优**
- 基于Alluxio提升HDFS集群的性能和SLA稳定性
- 总结

实验环境

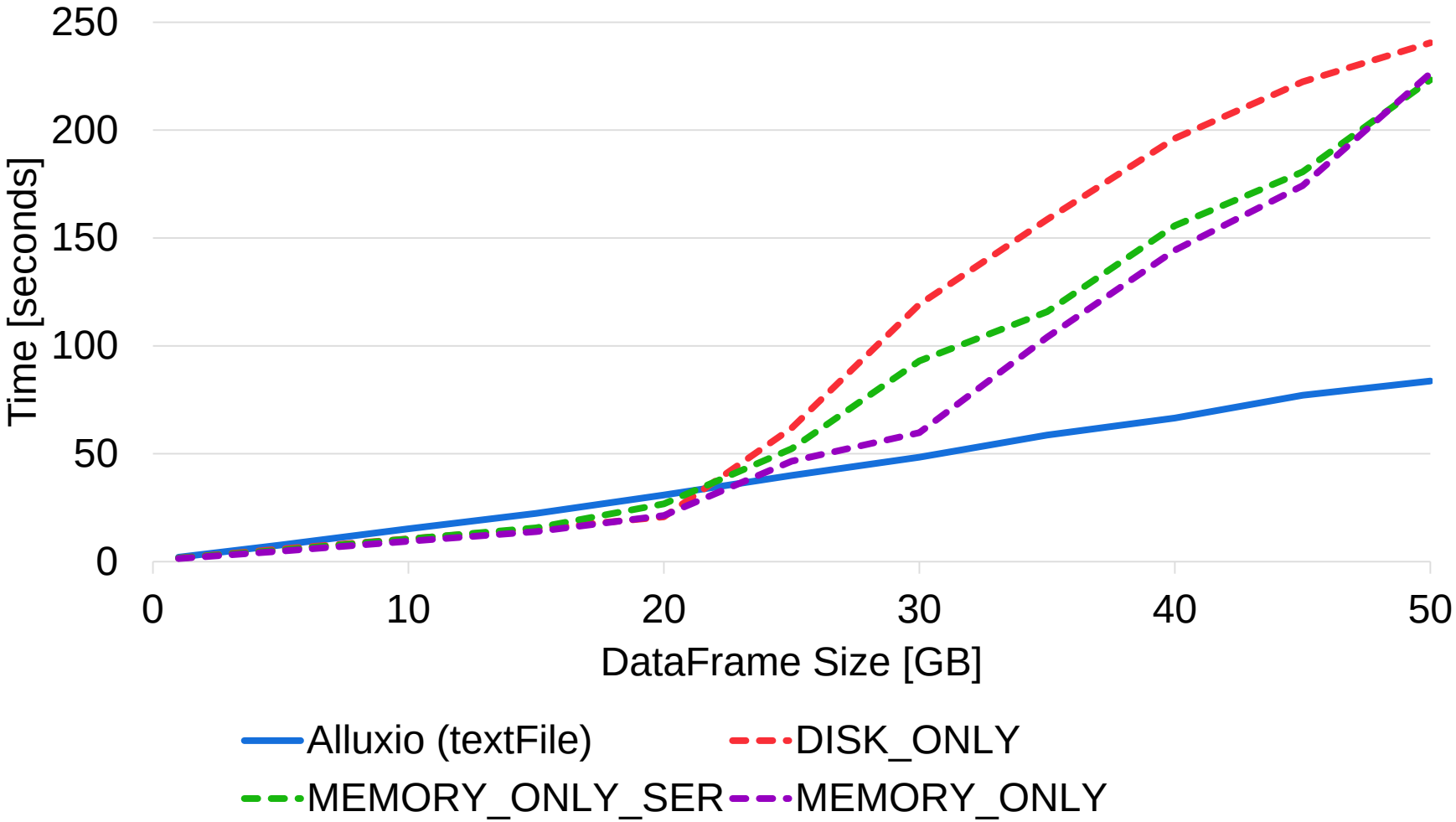
Spark 2.0.0 + Alluxio 1.2.0

Single worker: Amazon r3.2xlarge（61 GB MEM, 8-core CPU）

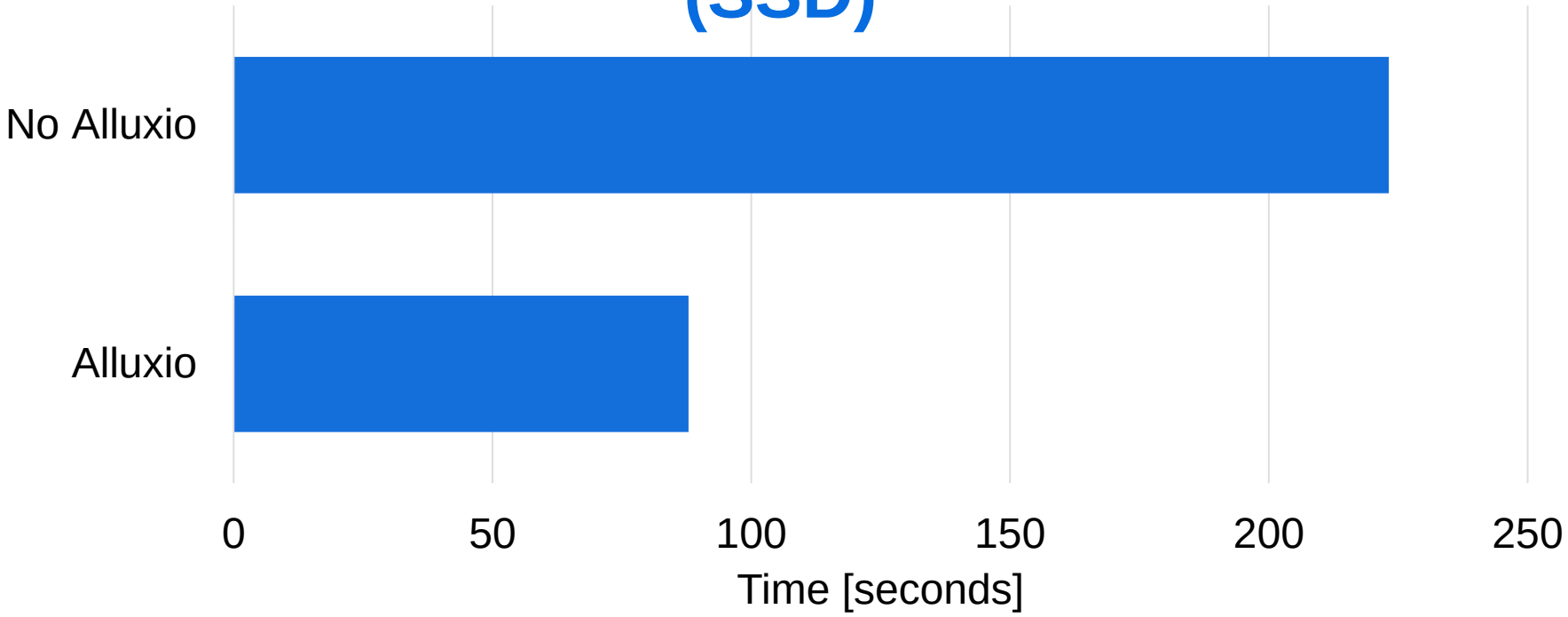
Comparisons:

- Alluxio
- Spark Storage Level: MEMORY_ONLY
- Spark Storage Level: MEMORY_ONLY_SER
- Spark Storage Level: DISK_ONLY

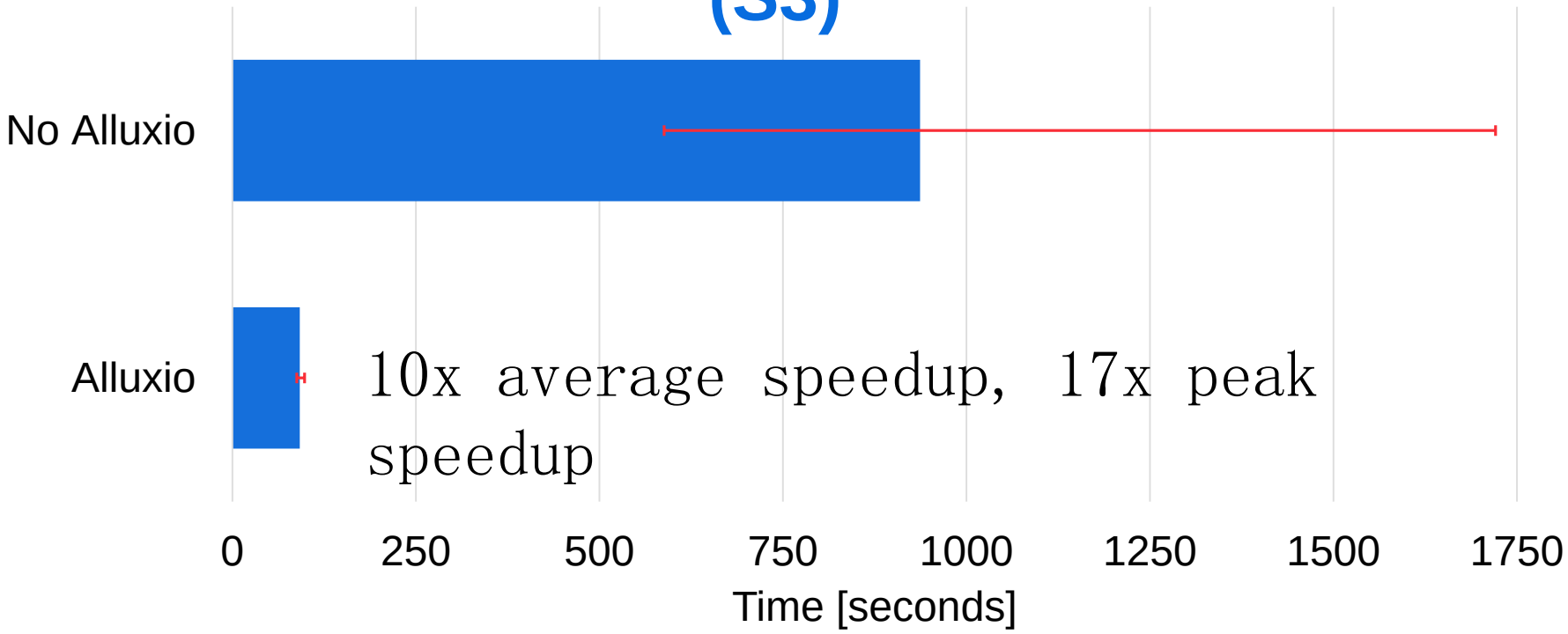
READING CACHED DATAFRAME (PARQUET)



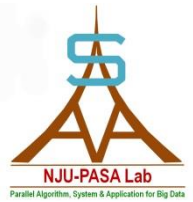
新场景：READ 50 GB DATAFRAME (SSD)



新场景：READ 50 GB DATAFRAME (S3)



内容

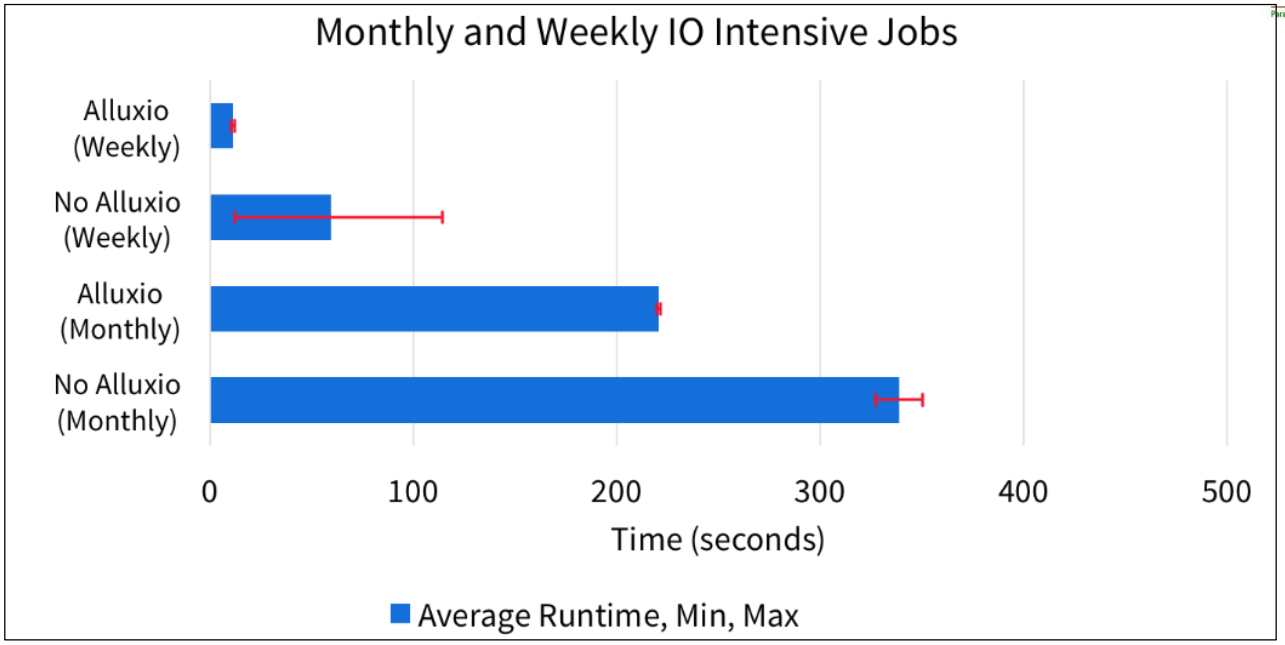


- Alluxio基本原理回顾与1.4的最新特性介绍
- 基于Alluxio的Spark DataFrame/RDD性能调优
- **基于Alluxio提升HDFS集群的性能和SLA稳定性**
- 总结

使用Alluxio提升HDFS集群的性能和SLA稳定性

- Alluxio可以给与HDFS共同部署的计算集群的两大好处
  - 提升高达10倍的性能提升
  - 性能的高可预测性使得SLA（service-level agreement服务级别协议）很容易满足
    - 例：作业运行时间的变化范围从100秒以上缩短至2秒

场景 1

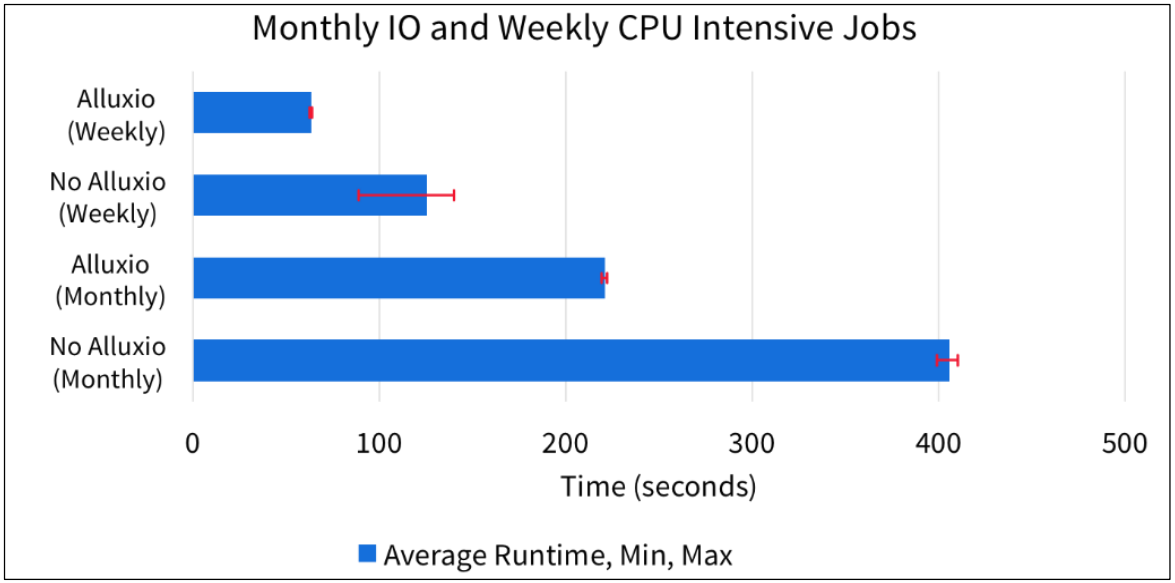


周作业和月作业都是IO密集型

结论：

1. Alluxio对这两种作业的性能提升都很明显。
2. 在不使用Alluxio的情况下，作业执行性能的波动范围较大（见图中用红线标出的最小和最大范围），甚至可能比使用Alluxio的情况**慢10倍**以上。

场景 2

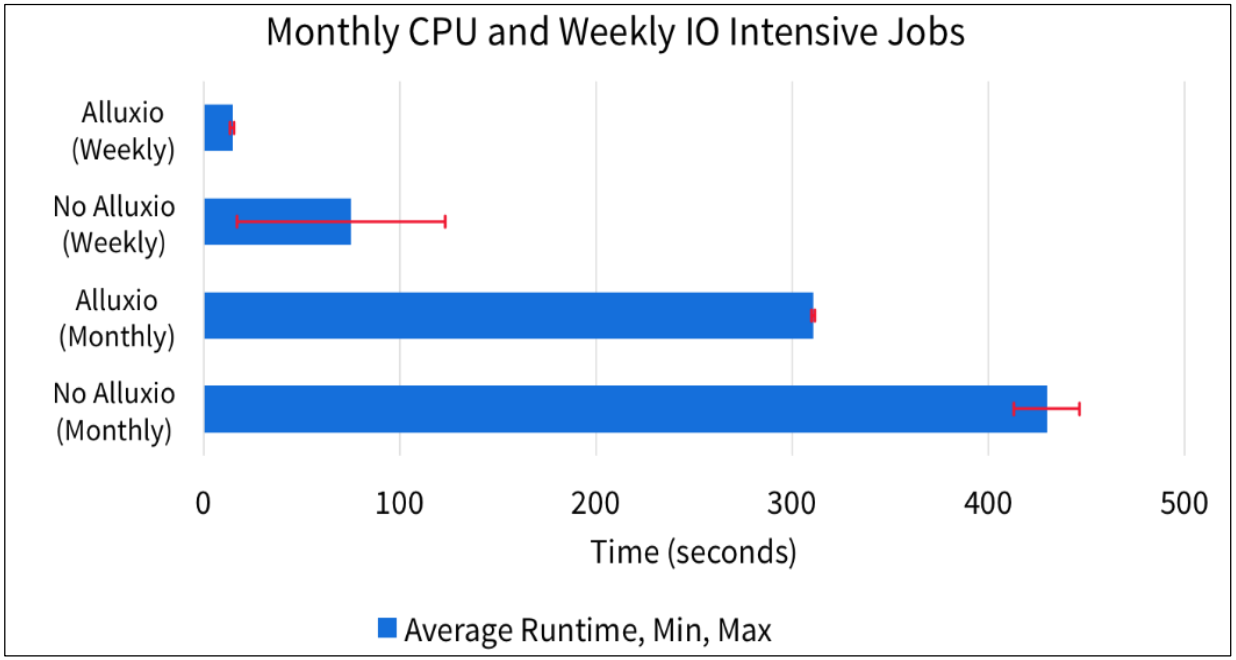


月作业I/O密集型，周作业CPU密集型

结论：

1. Alluxio还是同时提高了两种任务的性能
2. 周作业得益于Alluxio带来的内存级速度的I/O，但是性能提升没有之前的IO密集型作业明显（此时性能主要受机器的CPU吞吐量影响）
3. 月作业在使用Alluxio的情况下表现出极大的优势，因为场景1中月作业的性能影响因素在此场景下仍然适用

场景 3

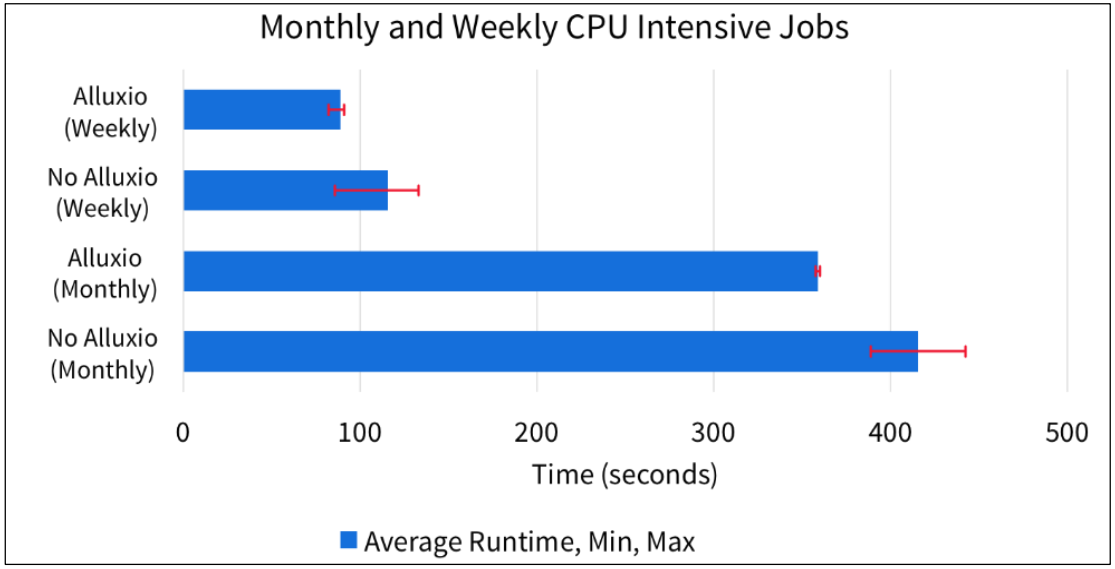


月作业是CPU密集型，周作业是I/O密集型

结论：

1. Alluxio极大地加速了周作业，因为周作业的数据完全在内存中
2. CPU密集型的月作业的执行性能也有所提升，因为使用Alluxio避免了周作业与月作业竞争磁盘资源。

场景 4

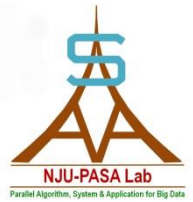


月作业和周作业都是CPU密集型

结论：

1. Alluxio带来的性能提升不明显，这是因为两种任务中I/O吞吐量都不是瓶颈。
2. 然而，通过持续地管理在内存中的数据，Alluxio使得作业的性能更为稳定。

内容



- Alluxio基本原理回顾与1.4的最新特性介绍
- 基于Alluxio的Spark DataFrame/RDD性能调优
- 基于Alluxio提升HDFS集群的性能和SLA稳定性
- **总结**

总结

- Alluxio可以在多个方面帮助Hadoop/Spark应用
  - Alluxio可以直接在内存中保存大规模的数据来加速Spark应用
  - Alluxio能够共享内存中的数据
  - Alluxio可以提供稳定和可预测的性能

欢迎使用Alluxio中文文档！

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

NJU-PASA Lab
Parallel Algorithm, System & Application for Big Data

The End & Thank you!

项目主页： http://alluxio.org/

个人微博：顾荣_NJU
电子邮箱：gurong@nju.edu.cn