京东大数据平台

规模

集群规模

服务器规模30000台+，离线集群总规模18000+ , 用户6000+

计算能力

离线数据日处理40PB+，日运行Job数100万+

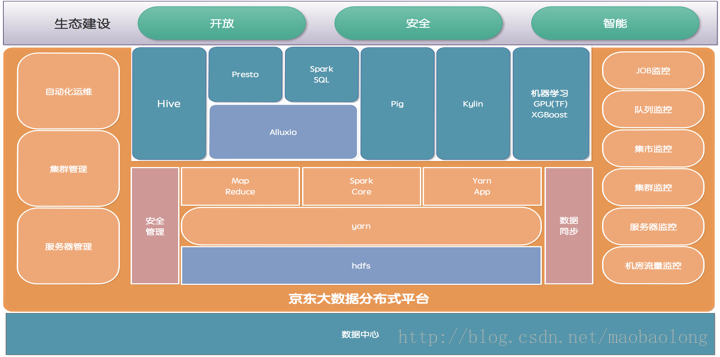
存储能力

总数据量400PB+，日增数据量500TB+

业务能力

业务主题40+，数据模型450+

大数据平台架构



Alluxio介绍

Alluxio（之前的Tachyon）。是世界上第一个以内存速度统一不同存储系统的系统。在大数据生态中，Alluxio位于计算框架和各种存储系统之间。此外，Alluxio存储为中心的架构，使数据访问速度的数量级比现有解决方案快得多。

Alluxio社区的优势

Alluxio社区的以下优势使得Alluxio越来越多的被很多知名企业所应用，也吸引越来越多的贡献者。

活跃开源社区

开源是大势所趋。800多Contributor，1723次fork。6000多次PR。社区Maintainer活跃度高，提出的PR很快就有回复

创始人为华人

华人contributor特别多，中文文档资料特别多。加入门槛底，还有新手教程，对于新contributor非常友好。

CI完备

alluxio-bot(PR title)、 AmplabJenkins作为PR builder(License header\Checkstyle\findbugs\test)

Alluxio in JD

Alluxio在京东的应用主要在JDPresto、Kylin、HDFS。本文主要介绍JDPresto在京东的应用。

JDPresto on Alluxio

Alluxio作为可插拔的容错组件应用于京东体系内诸多计算框架。

利用Alluxio优秀的缓存能力提供对ADHOC， 实时流计算天生的支撑，降低集群对于网络消耗的依赖。

JDPresto on Alluxio已经带来了10倍平均性能提升。Alluxio作为可插拔的优化组件，当Alluxio服务不可用时，JDPresto可以直接访问HDFS。我们的工作是扩展Alluxio，增强Alluxio和HDFS的一致性。

Alluxio已经在我们的生产环境运行1年多了。

优势

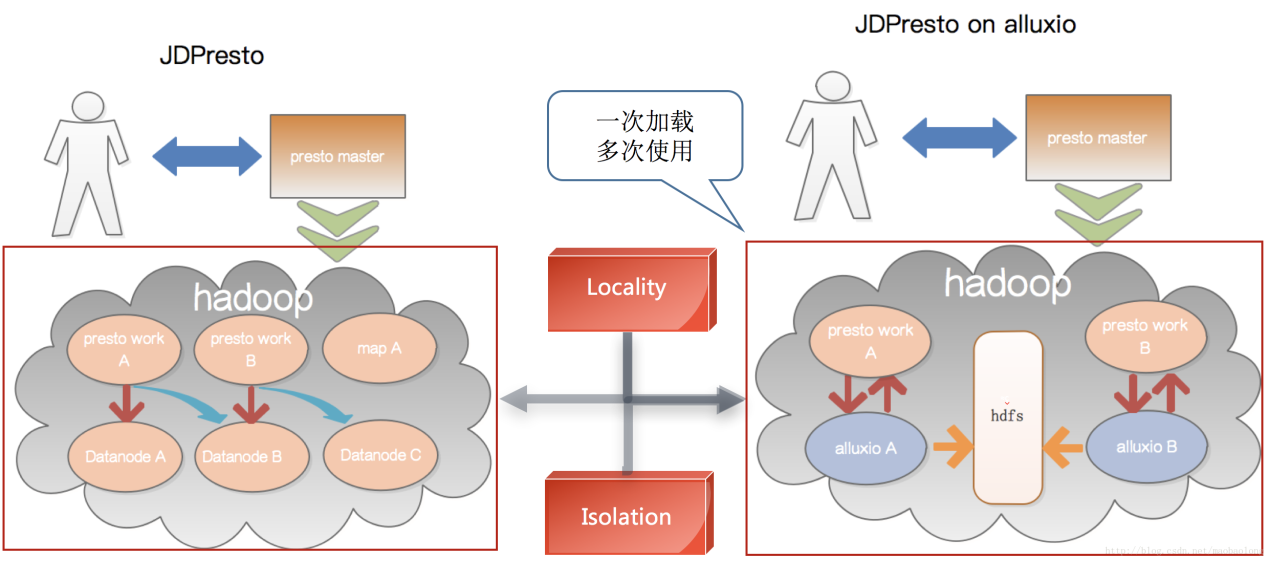
可插拔

容错

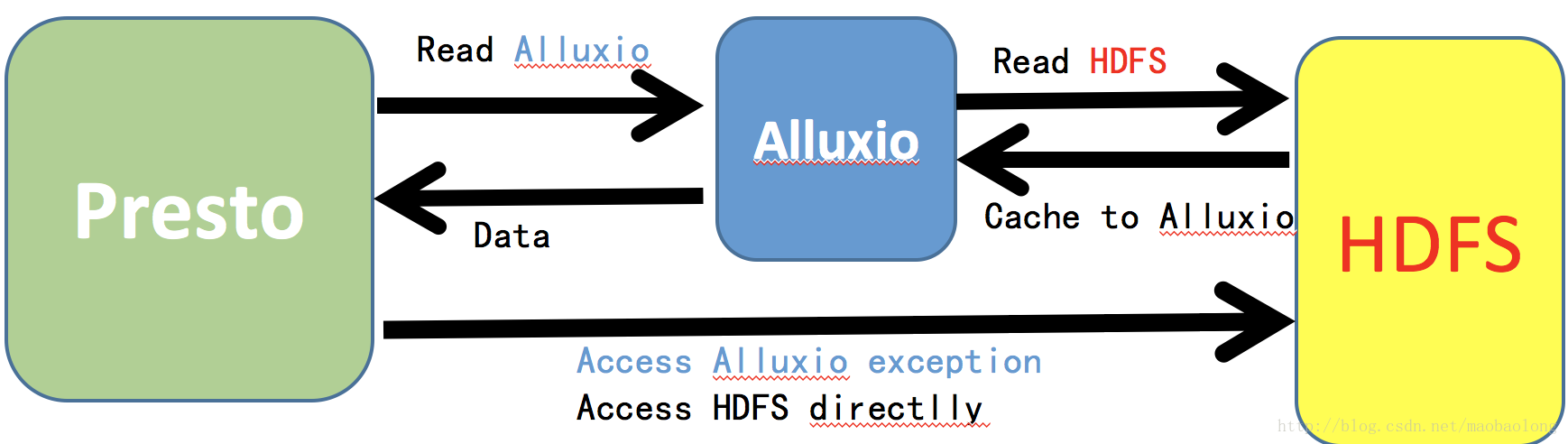
增强本地性

架构和流程介绍

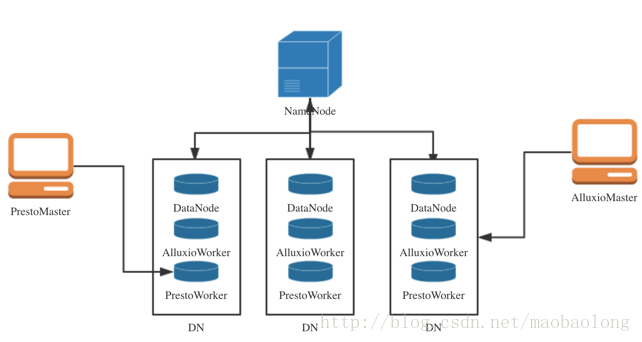
下图可以看出用Alluxio与不用Alluxio的Presto集群增加了隔离性和本地性



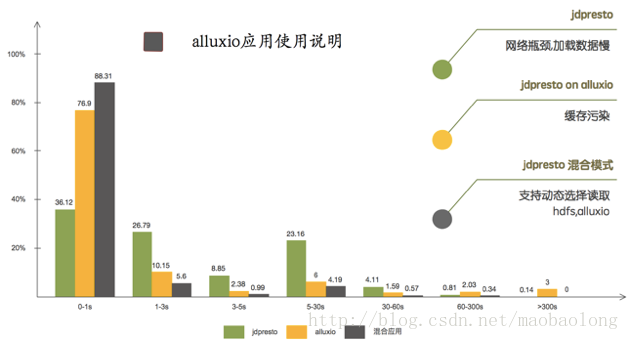
下图可以看出Presto会优先读取本地的缓存，如果读取Alluxio异常，则会直接读HDFS



下图是部署图

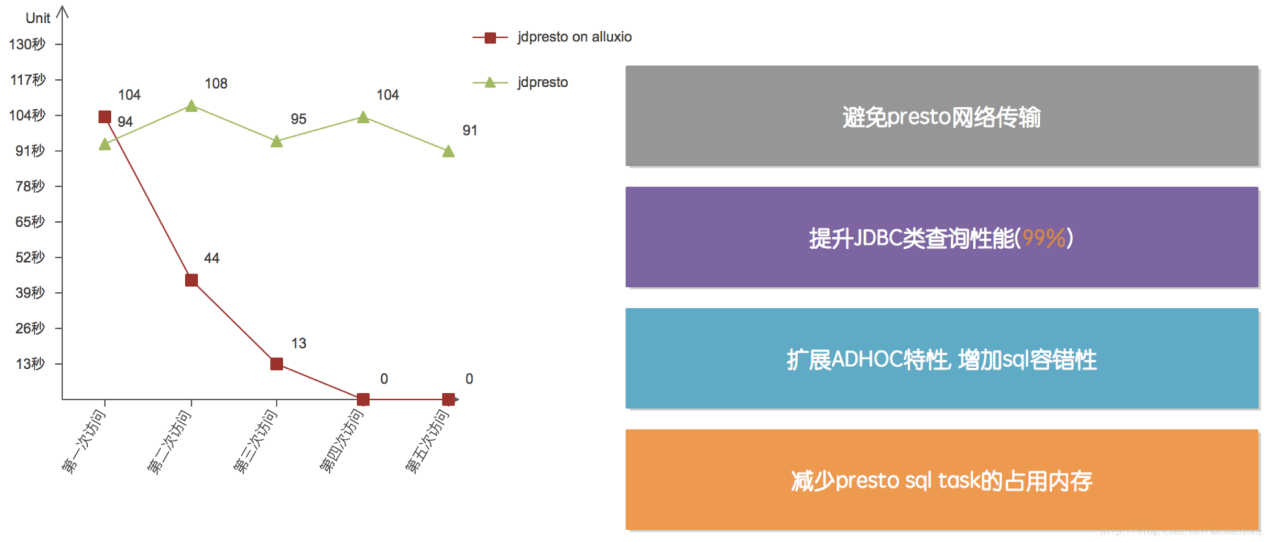


下图为使用JDPresto、JDPresto on alluxio、 JDPresto混合模式的对比图

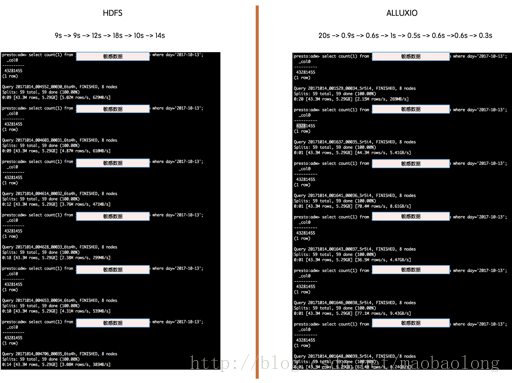


下图是测试5次Presto读时间用Alluxio和不用Alluxio的对比

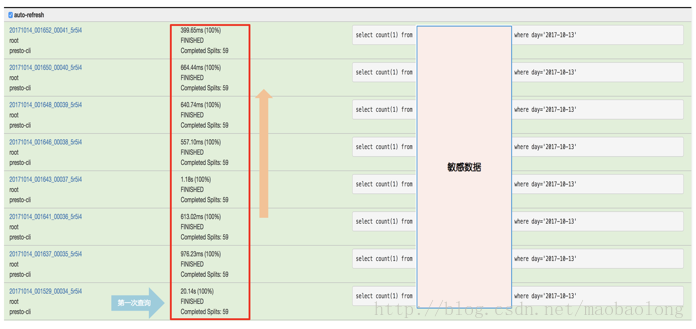
对比测试——sql查询速度



下图是在Presto的交互终端执行sql查询



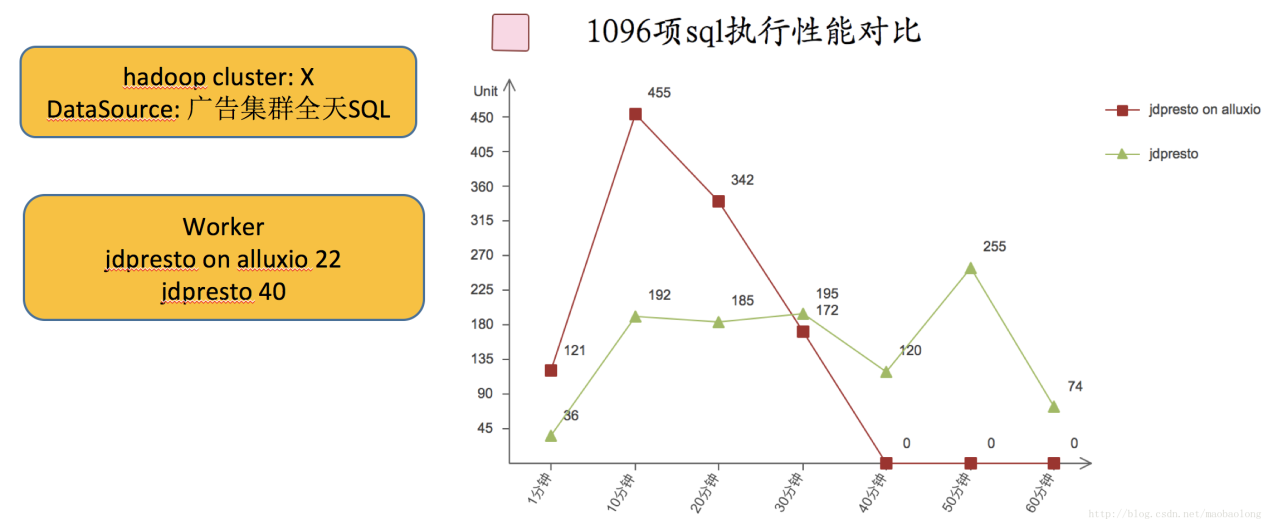
下图是在Presto的Web界面看到sql的执行情况



通过上边的查询结果可以看出采用Alluxio进行缓存，由于增加了本地性减少网络消耗以及使用内存进行存储减少IO，所以查询sql速度提升显著。需要说明的是采用Alluxio进行缓存，第一次查询需要从底层文件系统读取数据内容并缓存到Alluxio的Worker中，并且当时测试的版本Alluxio为1.4版本，采用的是同步缓存，所以速度反而会慢，1.5版本以后的Alluxio已经采用异步缓存，优化了第一次读取的速度。

性能压测——转发线上一天查询请求

我们开发了性能压测工具，转发线上sql到两个presto集群，一个使用Alluxio，为测试，另一个不使用Alluxio为对照。两个Presto集群在同一个HDFS集群中。



通过上图可以看出，红色线（使用Alluxio）在40分钟时已经执行完所有sql，而绿色线在60分钟也没有执行完。

我们的工作

为了让Alluxio在JD更好的发挥其作用，我们在Alluxio、Presto以及on Yarn方面做的工作如下图所示:



京东的贡献

Item Count

PMC 1

Contributor 6

PR 53

Merged PR 50

Merged Commit 221

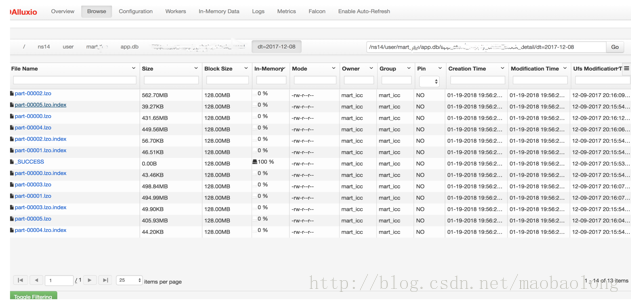
Additions / Deletions +4153 / -2254

新界面

基于ui-grid

支持排序、分页、过滤

可以指定位置显示文件或文件夹内容



很遗憾，新界面Patch没有合并到主分支了，需要的话，可以自行打PATCH。ALLUXIO-2557

一致性检查

启动检查：在启动的时候进行一致性检查，将不一致的文件或文件夹重新加载元数据

每次访问都检查：每次访问文件或文件夹时，都会与底层文件系统进行比对，不一致则更新元数据。已经演变成目前的指纹(FingerPrint)功能。

基于水位的空间置换策略

当Alluxio存储空间不足时，需要采用一定的空间回收策略进行回收。同步回收是旧版的回收实现。它当客户请求比当前在worker上可用空间更多空间时启动回收程序释放足够的空间来满足要求。 这导致许多小的回收尝试，并不高效。而采用基于高低水位的异步空间回收器，当空间空闲达到高水位时，进行空间释放，释放到空间空闲达到低水位时停止释放空间。

JVM暂停监视器

这是一个监视JVM暂停情况的服务。该服务建立一个简单的线程。在此线程中，在循环中运行sleep一段时间方法，如果sleep花费的时间比传递给sleep方法的时间长，就意味着JVM或者宿主机已经出现了停顿处理现象，可能会导致其它问题，如果这种停顿被监测出来达到一定的阈值，线程会打印相应级别的消息。还可以把额外sleep的时间进行metrics统计，外部的监控系统可以监控Alluxio的健康状态以及进行报警。

Shell命令

由于现有的Shell命令不能满足我们的需求，于是改进了现有的命令以及实现了一些新的实用的shell命令。

整合CopyFromLocal、CopyToLocal、Cp

ls命令增加 -h 参数显示带单位的文件大小，默认显示字节大小，与标准的文件系统命令实用习惯一致。

修改bin/alluxio，支持-debug参数开启的时候，增加ALLUXIO\_USER\_DEBUG\_JAVA\_OPTS环境变量的值到JVM参数中，该环境变量可以定位成远程调试参数，例如ALLUXIO\_SHELL\_DEBUG\_JAVA\_OPTS="-agentlib:jdwp=transport=dt\_socket,server=y,suspend=y,address=5603"

修改load命令，增加--local参数，如果该文件已经存在在Alluxio中，设置了--local选项，并且有本地worker，则数据将移动到该worker上。原有的逻辑如果其它worker已经存在该文件，则不会将文件load在本地worker。

修改rm命令，增加-alluxioOnly参数只删除Alluxio的元数据和缓存数据，并不会删除底层文件系统内容。

动态修改日志级别

日志库将日志分为5 个级别，分别为DEBUG、INFO、WARN、ERROR 和FATAL。这5 个级别对应的日志信息重要程度不同，它们的重要程度由低到高依次为DEBUG < INFO < WARN < ERROR < FATAL。日志输出规则为：只输出级别不低于设定级别的日志信息。比如，级别设定为INFO，则INFO、WARN、ERROR和FATAL 级别的日志信息都会被输出，但级别比INFO 低的DEBUG 则不会被输出。

我们有时希望在不重启服务的情况下动态设置某个Logger的日志级别。

所以，我们实现了Alluxio shell的一个logLevel命令，可以在特定实例上获取或更改特定类的日志级别。

语法是alluxio logLevel --logName = NAME [--target = <master | worker | host：port>] [--level = LEVEL]，其中logName表示日志的名称，target列出了需要设定的Alluxio master或worker列表 。 如果提供了参数level，则命令更改日志级别，否则将获取并显示当前日志级别。

例如，以下命令将foo.Bar类的日志级别在master和192.168.100.100：30000的worker上设置为调试级别。

alluxio logLevel --loggerName = foo.Bar --target = master，192.168.100.100：30000 --level = DEBUG

1

以下命令获取foo.Bar类的所有worker的日志级别

alluxio logLevel --loggerName = foo.Bar --target = workers

1

问题修复

修改了若干问题，不一一列举。

未来工作

跟踪Alluxio社区PATCH，向Alluxio提高代码

完善压测工具

完善监控和预警

稳定性提升

可调试支持

扩展其他框架使用

Fix Bug