ALLUXIO

Accelerating Ad-hoc Analysis with Spark SQL and Alluxio

What's Inside

WHITEPAPER

1 / Abstract

MOMO, a leading pan-entertainment social platform in China, has deployed Alluxio to accelerate ad-hoc query analytics. in the course of evaluating the best fit for Alluxio in their infrastructure they conducted several performance tests to understand how ad-hoc query analytics behaved in several scenarios. These tests give real-world insight to the performance benefits Alluxio provides. The MOMO findings include:

- With Alluxio, performance was improved 3-5x over the current mode
- Even when initially reading 'cold' data Alluxio delivered superior performance in most cases
- Alluxio can effectively scale-out to improve performance as requirements grow

2 / Trend

The hadoop ecosystem makes many distributed system/algorithms easier to use and generally lowers the cost of operations. However, enterprises and vendors are never satisfied with that, so higher performance becomes the next issue. We considered several options to address our performance needs and focused our efforts on Alluxio, which improves performance with intelligent caching. Alluxio clusters act as a data access accelerator for remote data in connected storage systems. Temporarily storing data in memory, or other media near compute, accelerates access and provides local performance from remote storage. This capability is even more critical with the movement of compute applications to the cloud and data being located in object stores separate from compute. Caching is transparent to users, using read/write buffering to maintain continuity with persistent storage. Intelligent cache management utilizes configurable policies for efficient data placement and supports tiered storage for both memory and disk (SSD/HDD).

3 / Alluxio Deployment Considerations

1. There are several examples of master-slave architectures in the big data ecosystem. Theses centralized system have a common problem, they have various states and significant amounts of metadata. Factoring in the working data, a master could face a heavy workload that degrades performance. Like NameNode in HDFS, it is a typical heavyweight service. But if a service is stateless or lightweight, its states and metadata it stores are not an issue, because they are easily recovered. However, Alluxio is a centralized service that puts performance pressure on our systems that the dev/ops team needs to address.

2. Alluxio is designed with a memory-centric architecture. Both reading from and writing to memory are definitely fast without doubt, but several questions need further consideration.

    - Since cold reading will trigger fetching data from remote sources, do tasks running on alluxio still perform well?
    - Do we need to load all data fetched from remote sources into alluxio?
    - If we ultimately write data to remote storage (like HDFS), why do we prefer to write through Alluxio rather than directly write to HDFS? The former obviously adds some overhead.
    - If we do write data to Alluxio first, what will happen in the event of a failure to an Alluxio master or worker.

The Alluxio distributed architecture enables the workload to be spread across multiple nodes in a scale-out fashion addressing that aspect of performance. For the metadata, we expect the master to scale as necessary, but in the event of a failure a secondary master can take over. Even if a master or worker node fails we can just format, restart the cluster and reload data from remote, avoiding any data loss.

Many variables affect the write performance so it is difficult to predict the best method for addressing it. In our case, stability is the priority so we chose not to explicitly address scenarios that were write intensive.

Alluxio is well suited to workloads where data is frequently accessed and applications can take advantage of memory caching. This avoids performance limitations from accessing data over the network and over-provisioning memory resources for data that is rarely accessed.

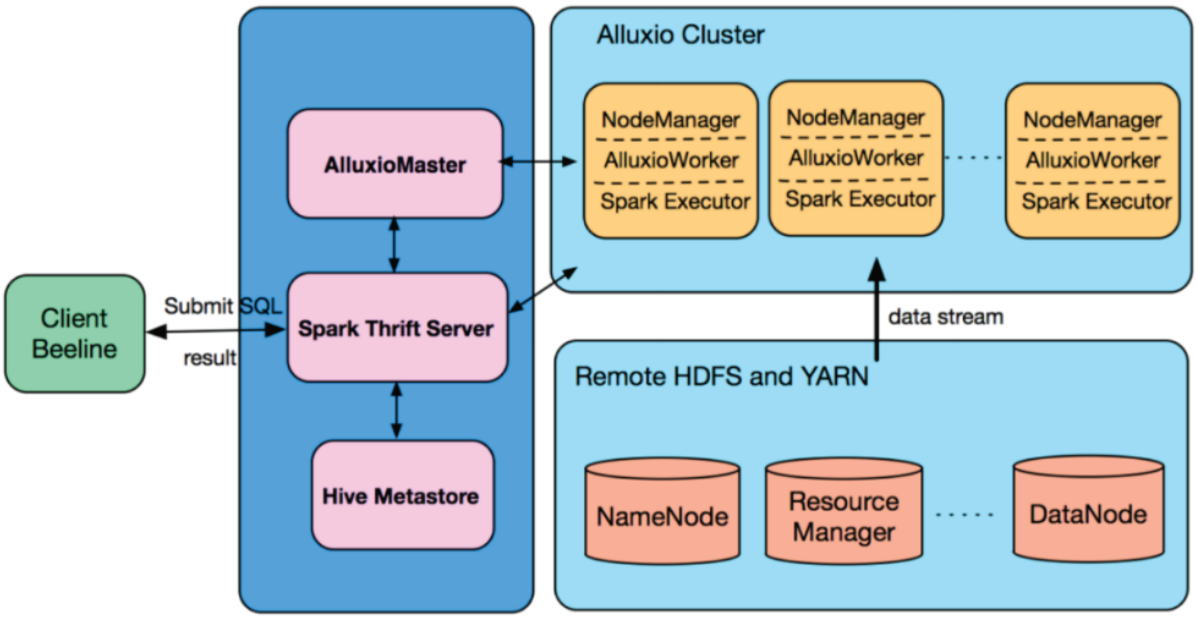
The best fit for our infrastructure was ad-hoc query. The data is frequently accessed, read-intensive, and easily recovered if necessary.

4 / MOMO Architecture

The first decision was to separate the Datanodes from the Alluxio workers to address the following issues:

1. Both processes require hard disk to store data, and the heavy I/O would likely lead to an increase in disk failure which is already an issue in production.
2. Separating the Alluxio workers provides dedicated HDD resources for caching. The HDD on the Datanodes are typically at over 80% capacity, so this is an effective way to manage resources independently and provide the best performance.

Additionally, we wanted to avoid irrelevant online tasks which do not recognize the Alluxio scheme being scheduled on Alluxio workers. As the node-label feature of yarn address our concern, we set up an independent and exclusive label cluster for Alluxio by marking it “ad_hoc”. The following figure shows our architecture.



5 / Performance Evaluation

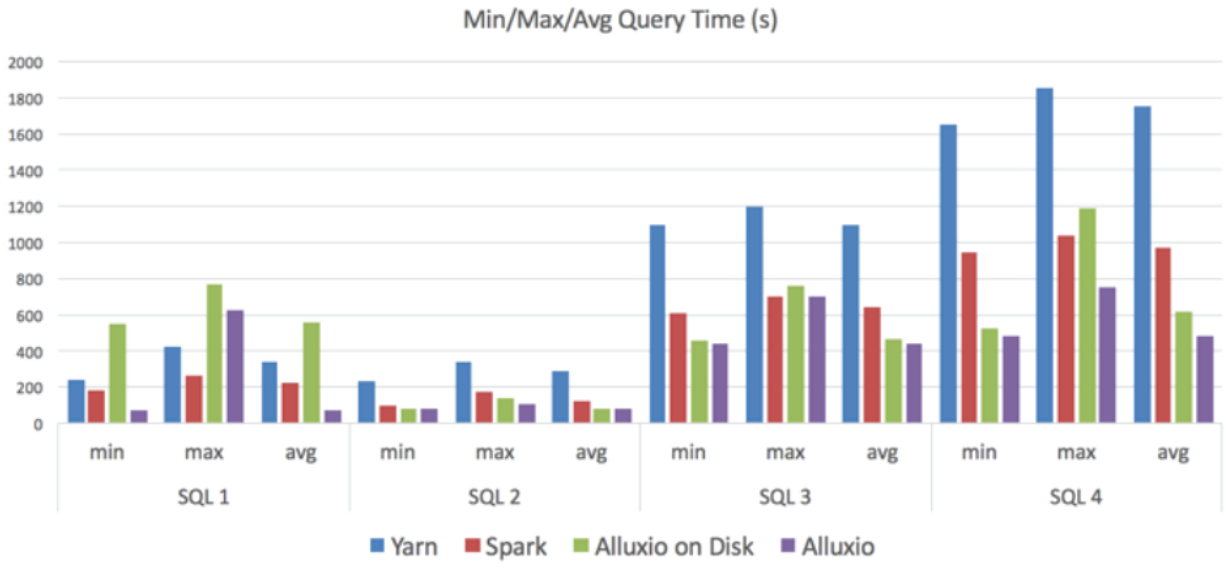
We designed an experiment with four typical queries of different sizes, and ran the queries in four different modes:

1. Yarn mode, which is our current online mode.
2. Spark mode, with tasks running on the label cluster but without Alluxio as a middle layer.
3. Alluxio mode running on the label cluster with both RAM and HDD layers configured.
4. Alluxio on Disk mode is nearly the same as the third mode with HDD cache only.

The main comparison we were interested in was the current Yarn mode vs. Alluxio with both RAM and HDD caching. The Spark mode and Alluxio on Disk mode were used as a `control test` for additional insight.

The following table shows the query information, and the chart displays the performance results. The x-axis is time measured in seconds, faster is obviously better.

Online SQL	Input Size
SQL 1	300G
SQL 2	1T
SQL 3	1.5T
SQL 4	5T



5 /

We can draw several inferences from the test results:

- Overall, Alluxio provides a significant performance boost as expected, which is 3-5x fastern than Yarn mode and 1.5-3x faster than Spark mode.
- Even with cold reading, Alluxio mode still performs better in most cases.
- The more Alluxio workers, the better performance can be achieved.
- By comparing Alluxio mode and Alluxio on Disk mode, the difference in performance between caching with Alluxio in memory vs. HDD was not significant enough to require a memory layer, considering the cost of memory.
- In certain small input size cases, Spark mode is as good as or better than Alluxio mode due to Spark memory management capabilities. However, once the size exceeds the JVM memory Spark cannot keep pace because Allux io uses off-heap memory techniques.

6 / MOMO Deployment

For the Spark thrift server, we developed a whitelist feature, allowing Alluxio to load data accordingly. With this method space in place Alluxio is fully utilized without unnecessary loading and eviction.

Additionally, to make Alluxio transparent to users, meaning without modifying any codes or sqls from client side, automatic switching between schemes is developed.

Therefore, if sql is a query and table involved is in whitelist, then the path of table will be transformed to an uri with an Alluxio scheme, and applications can read from Alluxio. If sql is an execution, it remains the same as the origin and writes to remote (HDFS in our case) directly.

7 / Summary and Future

There's no doubt that Alluxio improves performance in most cases, and we are planning to deploy it more widely and with more applications. In addition, we will be focusing more on security, stability and job monitoring issues and keeping pace with the community.

Future considerations:

- Fetching data from remote storage is constrained by network bandwidth and limits performance.
- Size of label cluster is limited compared to the online cluster comprised of thousands of machines, however the label cluster is much smaller with a limited amount of RAM.
- How to deploy Alluxio with write-intensive workloads.

8 / About MOMO

MOMO Inc (Nasdaq: MOMO) is a leading mobile pan-entertainment social platform in China. It has reached approximately 100 million active users and over 300 million users worldwide. Currently the MOMO big data platform has over 25,000 cores and 80TB memory which support approximately 20,000 jobs every day. And MOMO data infrastructure team works on providing stable and efficient batch and streaming solutions to support services like personalized recommendation, ads business data analysis demand.