Proposals for collaborated projects with Alluxio

1. Build an Alluxio Operator

Goal:

Build an Alluxio Operator to install and deploy Alluxio on Kubernetes and OpenShift clusters. Publish the operator to operatorhub.io for community use.

Details:

- Focus on running Alluxio cluster on a K8S or OpenShift cluster. Currently the Alluxio can be deployed on Kubernetes tbrough `helm` installation. We will either build a `helm` or `ansible` type of operator to automate the installation. Such operator can also monitor and manage the Alluxio cluster. More work needs to be done with OpenShift clusters since RBAC is by default enabled. The operator should provide the configurations for multi-tier storage support through persistent volume claims.

- Either extend the radanalytics-spark operator or build a new operator to install and deploy Spark + Alluxio service. Provide CLI to submit Spark applications that will run on Kubernetes cluster. Support using ConfigMap and Secrets to pass important application configurations.

2. Improve data locality for Spark Kubernetes scheduler/resource manager

Goal:

Inference Spark Kubernetes resource manager to run the executor pods on the Kuberentes node where the data is cached in Alluxio worker node.

Details:

To fully explore the caching (and short-circuit read) capability provided by Alluxio, the compute node needs to be on the same node as the Alluxio worker node caching the data. The best practice for Spark applications is to run the executor on that same node. For example, in a YARN managed Spark cluster, increasing the number of cores may lead to one executor been run on the same node Alluxio worker has the data cached.

Now with the Spark running on Kubernetes managed clusters, creating and load-balancing the executor pods are taken care by the Kubernetes scheduler. By default Kubernetes may schedule a pod that is not the same node where an Alluxio worker node is running on. This will incur networking overhead when transferring data between Alluxio worker nodes or from the UFS. Fortunately, Kubernetes resource manager allows the use of `nodeSelector` to designate the node on which the executor pod will run.

We will collect the useful file system metadata from Alluxio master node to look for the hint of the location of the node. Of course, we should handle the case when a node is truly busy. Also not clear whether a specific AlluxioRDD needs to be built and/or whether it can help.

3. Use Alluxio as intermediate storage for pipelines

Goal:

Each pipeline stage read from and write to Alluxio cluster to speed up the input and output flow.

Details:

In AI/ML pipelines, data and model are flowing from one stage to another. For example, the original data is input to the data cleansing stage. The data output from this stage may then be input to the featurization stage. These large amount of data needs to store somewhere, mostly the persistent storage such as S3. Alluxio clearly becomes the good candidate to help as an intermediate storage. Specially when the pipeline service is one of the Kubernetes native services: Kubeflow Pipelines or Pachyderm. Kubeflow Pipelines as well as Pachyderm runs each pipeline stage in a pod running container images.

The data output stage can write to Alluxio and if the amount of data exceeds the Alluxio storage configuration, data will be persisted to UFS. The data input stage can read from Alluxio. And similarly, if desired, data locality can be implemented to run the pod selectively.

4. Use Alluxio in R

Goal:

- Create R language binding for Alluxio with its REST APIs.
- Create a R package to use Alluxio as the memory for data.frame.
- Allows R with GPUs to use Alluxio.

Details:

R has a large amount of packages and user base in data science and machine learning areas. Its limitation however is single machine execution. The size of a R data.frame is limited to the RAM size of the machine. With Alluxio multi-tier storage, we will explore the option to extend the data.frame to use the hard disk as well. A new R package providing a new data.frame implementation can hold data far more than the machine's RAM limit. And also with the data movement between Alluxio worker nodes, such data.frame will not be only loaded data physically persistent to the local machine.

One good reason for such support in R is due to the reality that today, many good ML algorithms and data visualization tools are not implemented or possible on existing distributed R frameworks, such as SparkR or sparklyr. Therefore, computing on a single machine still has its use case.

We will also look at the GPU support in R. Providing code running in a GPU access to Alluxio cache is also a great addition.