Managing Lustre on AWS

Andy Pollock and Aurélien Degrémont
Amazon FSx for Lustre

May 16, 2019
Agenda

- Amazon FSx for Lustre Introduction
- Connecting to S3 with Lustre and HSM
- Example Deep Learning workflow
- What’s next?
Introducing:

Amazon FSx for Lustre
Why Lustre?

- At AWS we work backwards from the customer
- Our customers needed hundreds of GB/s throughput and sub-ms latencies
- They wanted Lustre by name
Presentation

• Fully managed Lustre filesystem
• Lustre clients are customer-managed
• Can be attached to an S3 bucket
  Encrypted at rest.

Customer Instances

AWS services

Lustre clients

Amazon S3 Bucket
Characteristics

- From 3.6 TiB to more than 1,000 TiB
- Bandwidth: 200 MB/s per TiB
- Running in virtual machines, on AWS EC2
- 1 MDT - 3 % of total filesystem size
- Many OSTs - 1.1 TiB each, NVMe based
- Lustre 2.10
Easy to start

Create with a simple CLI command (Web GUI and API available)

```
aws fsx create-file-system

--file-system-type LUSTRE --storage-capacity 3600
--subnet-ids ... --security-group-ids ...
```

```json
{
  "FileSystem": {
    "FileSystemType": "LUSTRE",
    "StorageCapacity": 3600,
    "Lifecycle": "CREATING",
    "DNSName": "fs-0bd0cc7e8.fsx.us-east-1.amazonaws.com",
    ...
  }
}
```
Easy to access

1. Start an EC2 instance, with the proper networking configuration
2. Install the Lustre client (documentation available for major distros)
3. Mount the filesystem using standard command:

```
$ mount -t lustre fs-0bd0cc7e8.fsx.us-east-1.amazonaws.com@tcp /fsx
$ lfs df -h

<table>
<thead>
<tr>
<th>UUID</th>
<th>bytes</th>
<th>Used</th>
<th>Available</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsx-MDT0000_UUID</td>
<td>102.8G</td>
<td>2.6M</td>
<td>102.8G</td>
<td>0%</td>
<td>/fsx[MDT:0]</td>
</tr>
<tr>
<td>fsx-OST0000_UUID</td>
<td>1.1T</td>
<td>4.5M</td>
<td>1.1T</td>
<td>0%</td>
<td>/fsx[OST:0]</td>
</tr>
<tr>
<td>fsx-OST0001_UUID</td>
<td>1.1T</td>
<td>4.5M</td>
<td>1.1T</td>
<td>0%</td>
<td>/fsx[OST:1]</td>
</tr>
<tr>
<td>fsx-OST0002_UUID</td>
<td>1.1T</td>
<td>4.5M</td>
<td>1.1T</td>
<td>0%</td>
<td>/fsx[OST:2]</td>
</tr>
</tbody>
</table>

filesystem_summary: 3.3T 13.5M 3.3T 0% /fsx
```
Monitoring with Amazon CloudWatch

Several metrics available like space usage, bandwidth, iops, …
Connect to S3 with Lustre/HSM
Highly performant Amazon S3 access

Amazon FSx For Lustre can be connected to S3 as an HSM backend
Spin-up / spin-down Workflow

Typical workflow is to start, restore, compute and archive back

1. Start
2. Read
3. Compute
4. Archive

Import

Restore

Archive

Amazon S3 Bucket

© 2019, Amazon Web Services, Inc. or its Affiliates.
Importing millions of files

- There is no limit to the number of objects that can be stored in S3 buckets
- Customers will frequently have millions or even billions of objects
- Challenge is to create a full namespace using an object list from S3
- We create empty, released files in the Lustre filesystem for each of them
- Lustre client performance sustains S3 speed
  - Be careful with very large directories (LU-8047)

s3://bucket/file1.txt
s3://bucket/file2.txt
s3://bucket/folder1/file3.txt
s3://bucket/folder2/file4.txt

/fsx
  file1.txt
  file2.txt
  folder1/
      file3.txt
  folder2/
      file4.txt
Restoring and archiving large buckets

Files are accessed using standard mechanisms:
• Opening a released file
• Using standard Lustre lfs hsm_restore commands

Convenient to restore all files, prior to start working
Periodically, or at the end, archive all their files back to S3.

Using standard Lustre lfs commands, you can enqueue thousands of requests into the coordinator queue.

Workload is distributed across all filesystem servers to maximize bandwidth
Operating Lustre/HSM at S3 scale

- We have done a lot of stress testing of HSM and are helping to maintain it
- Large-scale imports, restores and full system archives have exercised Lustre/HSM a lot
  - Example: memory leak fix (LU-11892)
  - Example: identified opportunity to optimize import/export performance by avoiding linear scans
- Copytools not reading incoming work (KUC) can deadlock the coordinator
  - We have had to make sure that our copytool does not deadlock the coordinator
  - Further opportunities for coordinator-side enhancements
- Coordinator activity reporting:
  - Status is available per file (using lfs hsm_action or lfs hsm_state)
  - Opportunity to have an overall status report for the global archiving or restore progress
Example Use Case: Amazon FSx for Lustre and Deep Learning
Example Use Case: Amazon FSx for Lustre and Deep Learning

- Integrating Lustre with AWS batch computing services and GPU hosts for deep learning training jobs

- Deep learning training using ImageNet2012 dataset and Tensorflow
  - Annotated image database

- Deploy an infrastructure with a workflow using several AWS Services
  - Schedule compute tasks with AWS Batch
  - Efficient I/O with Amazon FSx for Lustre and Amazon S3

- Based on [AWS Compute Blog post](https://aws.amazon.com/blogs/compute/)

© 2019, Amazon Web Services, Inc. or its Affiliates.
Architecture

Multi-node Tensorflow container registry

Multi-node parallel

P3/P3dn container Instances

DL placement group

Lustre kernel driver

commit

hydrate

Amazon S3

Amazon ECR

AWS Batch

Amazon FSx for Lustre

Amazon FSx
Prepare images

1. Prepare an EC2 instance image using Ubuntu 16.04, with Nvidia drivers, Lustre client and Docker.
   - Save it as an Amazon Machine Image (AMI) for later use
2. Prepare a TensorFlow container with TensorFlow, Horodov, Cuda libraries and OpenMPI
   - Push it to ECR registry
Workflow

1. Start a 3.6-TB filesystem, attached to S3 bucket with ImageNet dataset
   $ aws fsx create-filesystem ... --ImportPath s3://mybucket/imagenet

2. Start a AWS Batch environment
   1. Compute environment
   2. Compute resources
      • Instance type: p3 family (Tesla V100)
      • vCPUs: 0 to 4096
   3. Proper network access to the Lustre filesystem

3. Create the job definition that uses these definitions and the docker container

4. And run!
Results

- Using: 20 x p3.16xlarge instances (8 Tesla V100, 64 cores, 128 GB RAM)
- 100,000 images/sec, 90-100% GPU utilization
What’s Next?
Network namespace enhancement (1/2)

- Containers are very popular, especially in the Cloud
- Based on *cgroups* and *user namespaces*
- *Network user namespaces* own specific network interfaces, IP addresses, routing and firewalling tables, ...
- LNET only supports the default network namespace
Network namespace enhancement (2/2)

- We are enabling the use of any network namespace and not only the root namespace.
  - LU-12236 – Support more than the default root network namespace.
Working with the community

- We are sharing our Lustre modifications
  - LU-11892 – Memory leak in MDT Coordinator
  - LU-12227 – Lustre init script does not check if ZFS devices are already mounted
  - LU-12236 – Support more than the default root network namespace
- We plan to do more!
Thank you!