Harvesting, Processing, Storing data from HPCM Systems

Ben Lenard
Cray Users Group 2025

Acknowledgement

- This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH1135

U.S. DEPARTMENT OF ENERGY | Office of Science

THE UNIVERSITY OF CHICAGO

The ALCF environment

- Aurora
  - Exascale class supercomputer
  - 10,624 compute nodes and each compute node having 2 Intel Xeon CPU Max Series processors: 64GB HBM on each, 512GB DDR5 each; 6 Intel Data Center GPU Max Series, 128GB HBM on each, RAMBO cache on each; Unified Memory Architecture; 8 SlingShot 11 fabric. It ranked number 3 on the Top 500 list in the Fall of 2024
- Polaris
  - 44 Petaflop Machine
  - Hewlett Packard Enterprise (HPE) supercomputer system
  - 560 AMD EPYC Milan processors and 2,240 NVIDIA A100 Tensor Core GPUs
- Crux
  - Petascale class supercomputer
  - 256 nodes, each node having 2 AMD EPYC Milan processors with HPE Slingshot 11 fabric endpoints.
- Sirius
  - A T&D system

The problems we were looking to solve

- We have multiple HPCM systems potentially operating on different releases of the management software
- In addition to the standard data generated by HPCM, we generate data from:
  - PBS Pro
  - XALT
  - Telegraph
  - Custom Scripts / applications
- Our compute nodes are stateless or ephemeral and some of these applications only write to log files
  - While a NFS server might handle 560 Polaris nodes, it would likely not scale to Aurora's size
- We have multiple consumers of the same data
- We needed near real-time use of the data
  - Instead of the ETL'ing files on a set interval
- We needed to bring order to the different data sources and have a repeatable method for deployment
- We need a redundant and fault tolerant system since we seldomly take a complete outage

The Data to capture

- We wanted to capture:
  - The data from the HPCM Kafka topics
    - Each HPCM system has its own Confluent Kafka Cluster
  - user build and execution data via XALT, which writes json files on each node
  - The PBS logs from the PBS server as well as the logs on each node
  - Data from custom applications

What is XALT?

- XALT is an open source High Performance Computing (HPC) application build and execution tracking framework. Highly configurable, XALT produces Javascript Object Notation (JSON) records detailing the libraries and other runtime information that enable users to build and execute their HPC applications.

PBS Pro

- PBS Pro is the job scheduler that ALCF uses:
  - Accounting Logs
  - Server Logs (pbs_server)
  - Scheduler (pbs_sched)
  - Mom (pbs_mom)
  - Comm (pbs_comm)

Kafka Cluster within ALCF

- We deployed Apache Kafka within ALCF out of the need to move messages from Point A to Point B from these various systems; we also use this cluster for non-HPCM systems as well.
- We have 5 physical servers for our cluster
- We deployed our Kafka cluster with Kraft instead of ZooKeeper
- We have 5 nodes, all of which are voting member in the quorum, we also have replication set to 3, resulting in 2 nodes can be down at the same time
- We require SSL traffic is required to connect to the cluster as well as username and password.
- Topics are secured with ACL's by username – by default there is no ACL's and anyone can do anything.
- We're currently running 3.6 and moving to 3.9

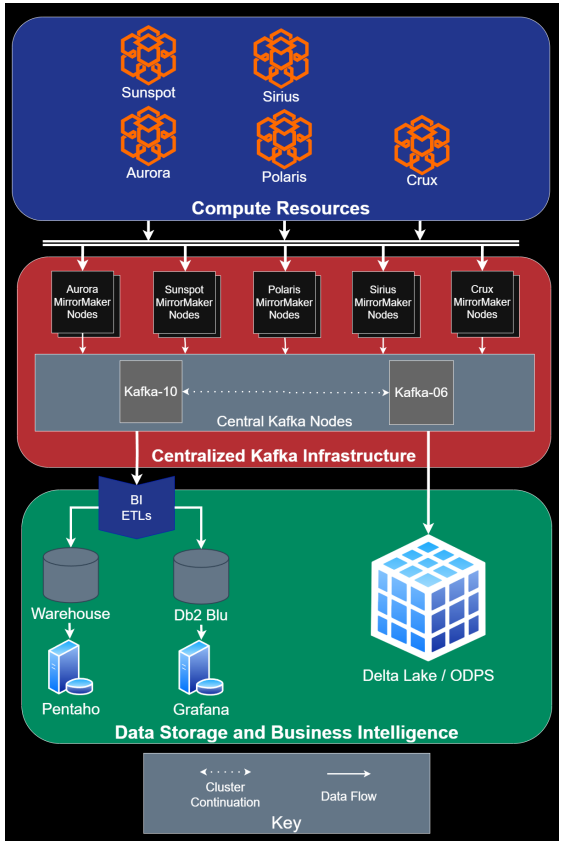
What is Kafka MirrorMaker and Filebeat?

- Kafka MirrorMaker
  - A tool part of the Kafka package that acts as a Consumer and Producer at the same time
  - It is used to mirror a topic from a source topic in a source cluster to destination cluster
  - Multiple MirrorMaker servers can be run concurrently for redundancy
- Elasticsearch Filebeat
  - A log file parser and tailer that can copy the log file data to ether ElasticSearch or Kafka
  - Configurable for which files to watch and exclude; this can be done at the line level too
  - ALCF has been using Filebeat for a number of years to place syslog data into Elasticsearch
  - This is lightweight compared to its counter part Logstash and is written in Go
  - The event that gets put on the topic is JSON with meta data about where the event came from

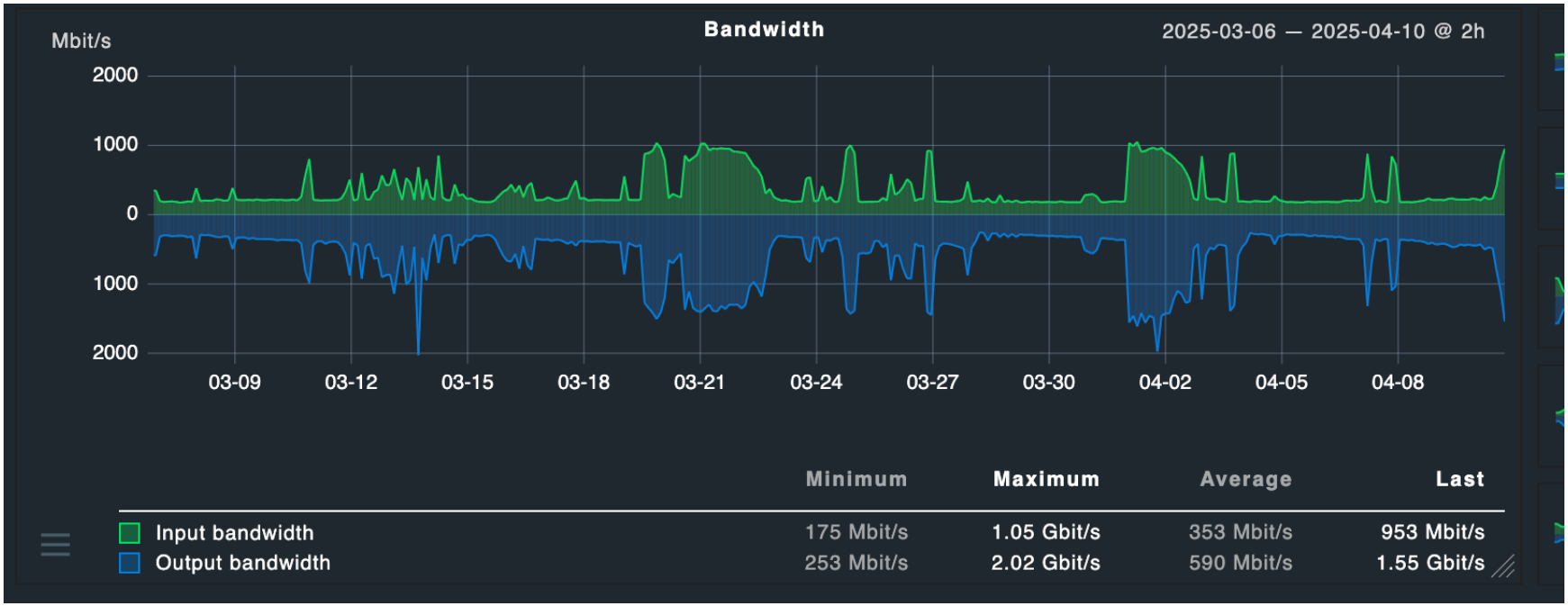
Harvesting the Data

- In terms of harvesting the data:
  - We use Filebeat on the compute nodes to read the log files and then push them into the system's kafka cluster
    - i.e. XALT logs on a Polaris Compute node is read by FileBeat and then transmitted to Polaris' Kafka cluster into its topic
  - We then clone the system's Kafka Cluster to the Centralized ALCF Kafka Cluster via MirrorMaker
    - i.e. We use two MirrorMaker servers/apps to clone the Polaris Kafka topics to the Centralized cluster. In the centralized cluster, these topics are prepended with the system name.

Kafka Cluster within ALCF (Cont.)



An idea of the bandwidth one of the ALCF Kafka nodes



Storing the Data

- We have two direct consumers of the data on the ALCF Kafka cluster
  - —Business Intelligence
    - The team consumes the data from Kafka and stores the data in Db2
    - Row and Columnar formats
    - Dedicated database for columnar format using BLU – roughly 62TB in size
    - Centralized data warehousing database with both formats – roughly 7TB in size
    - 32 Gb Fibre Channel to the Netapp AFF 400 storage system
  - —Operation Data Processing System
    - The team consumes the data from Kafka via Spark and stores the data in a Delta Lake
    - Uses Parquet files as storage
    - Stored via S3 objects via 100Gb ethernet to the Netapp AFF 400 storage system
    - Roughly 54 TB in size

What is a Data / Delta Lake?

- Data Lake
  - —is a centralized repository designed to store large-scale structured, semi-structured, and unstructured data
  - —primary advantage of storing disparate information in a data lake is centralization of the data
  - —Data lakes have faced challenges related to inconsistent data, data reliability, and poor performance
- Delta Lake:
  - —offers significant advantages over traditional data lakes, including data consistency, reliability, and scalability
  - —they provide ACID (Atomicity, Consistency, Isolation, Durability) guarantees, ensuring that data within a Delta Lake is consistent and reliable
  - —they provide scalable metadata handling by using a transaction log
  - —schema enforcement and data versioning
  - —S3-style object storage

What is ODPS?

- Operational Data Processing System (ODPS):
  - Paper at DEXA 2023 (ODPS 1.0) and SSDBM 2025 (ODPS 2.0)
  - It's a system for processing the HPC system's events, storing them into a Delta Lake, and looking criteria
    - i.e. show me the nodes where Project XYZ had an issue and between a given time period
  - It use uses a bitmask to map the system out as well as provides a quick way to search for these events
    - Think of it as bitmap index in a relation database; the speed is obtained by scanning a bitmap vs a b-tree
  - Uses Spark, Scala, and a S3 object store
  - Many improvements in ODPS 2.0

Processing the Data

- BI Team
  - Processes the data as it's being stored in the database via a traditional ETL fashion via Python applications
  - Some of these Python applications take actions based on the data values, such as sending messages to Slack
- ODPS
  - ODPS 2.0 uses Spark to read from the Kafka bus
    - Each partition of each topic has a worker process
  - Stored in the Delta Lake as 3 levels
    - Level 0: Raw data
    - Level 1: Decoded data from Level 0 from each partition
    - Level 2: The merger of various level1's into a single table

Uses of this data

- BI team uses Pentaho and Grafana
  - Tradition reporting for the DoE
  - Usage of the systems by science domain
  - Usage of the systems by project
  - Usage of the systems
  - Power consumption of Aurora
- ODPS 2.0 uses of the data
  - JFA debugging reason for a job that failed
  - Debugging why a job would not start and was re-queued
  - Debugging why a piece of hardware went down
  - Figuring out the impact of a piece of hardware going down
  - Finding a very specific error across many days across all nodes

Questions?