Performance Visualization of ROOT I/O on HPC Storage Systems

Lightning Talks

Rui Reis

06/09/2021
Contextualization

Current state-of-the-art

- In order to analyse the tremendous amount of data generated from High Energy Physics (HEP) experiments, CERN uses the ROOT framework.
- Previous versions of ROOT used the TTtree data format, however, it will be soon replaced in v7 by RNTuple, an efficient columnar storage format developed by CERN.
- RNTuple also provides a set of metrics for the analysis of data ingestion performance, users can also add custom metrics, as desired.

```cpp
ntuple->EnableMetrics();

RNTupleMetrics inner("inner");
auto ctr = inner.MakeCounter<RNTuplePlainCounter *>("plain", "s", "example 1");
```
Contextualization

Metrics simplicity

Despite its capability to provide the user with useful insights, current RNTuple metrics constraint to counter aggregate type metrics, which are too simple in many scenarios (e.g., when we need to analyse the distribution of a certain metric).
Challenges

- Current RNTuple metrics are too simple to provide viable information about the distribution of data.
- Which makes the following questions hard to answer:
  - What is the distribution of the size of read requests to load an entire ntuple cluster?
  - How can we know if our ntuple metrics are unevenly distributed?
  - How can we detect the existence of outliers in our metrics?
- Possible solutions need to be efficient and be able to construct histograms on-the-go.
Solution #1

User-Provided Set of Intervals

Cons:
• Requires knowledge of underlying data
• Unable to detect outliers
• Error-prone
Solution #2

Log Scale

Cons:
- Amplitude of intervals is exponentially large
- Hard to interpret the meaning of histogram output
- Able to detect some outliers, depending on scale
Solution #3
Active Learning Phase

Cons:
- Heavily dependant on the distribution of samples in the LP
- Occurrence of outliers in LP deeply affects efficiency of histogram
- Can’t effectively separate outliers from real distribution
Solution #4

Fixed Width Bins

width = 100

#1: 70
#2: 170
#3: 270
#4: 370
Solution #4

Calculating the bin key (Fill Algorithm)

- If a new value, N, is greater or equal to the offset, then:
  - \( \text{Key} = (N - \text{offset}) / \text{width} + \#\{\text{below offset bins}\} + 1 \)
- Else:
  - \( \text{Key} = \#\{\text{below offset bins}\} - (\text{offset} - N) / \text{width} \)
- Examples, width=100, offset=170:
  - \( N = 178 \Leftrightarrow \text{key} = (178 - 170) / 100 + 2 + 1 = 3 \)
  - \( N = 384 \Leftrightarrow \text{key} = (384 - 170) / 100 + 2 + 1 = 5 \)
  - \( N = 105 \Leftrightarrow \text{key} = 2 - (170 - 105) / 100 = 2 \)
  - \( N = 69 \Leftrightarrow \text{key} = 2 - (170 - 69) / 100 = 1 \)
After the desired analysis, the histogram content can be dumped as a CSV and fed to external plotting utilities for visual analysis.
Sample #2

Convert LHC 1 run open data from TTree to RNTuple
Conclusion

• Performance visualization can easily allow a detailed analysis of the underlying metrics.
• Histogram output format can be easily ingested by external plotting utilities.
• More information can be found on the PR: [ntuple] Performance visualization improvements by ruipreis · Pull Request #8880 · root-project/root (github.com)
QUESTIONS?

ruipedronetoreis12@gmail.com