Multi-Threaded Checksum Computation for the ATLAS High-Performance Storage Software

Fabrice Le Goff, Giuseppe Avolio (CERN) - ACAT 2019

The workload of the storage application is then determined by the classification made by the High-Level Trigger. Here is the stream distribution for a typical ATLAS run in 2018. With this workload and one stream clearly dominating the throughput, the application is CPU-bound, limited by the checksum computation capacity for a single CPU core. Therefore the application uses at most ~3.2 of the 12 available CPU cores. The limitation really comes from single core performance, the necessary serialization of events in each stream, and the real-time nature of the workflow.

**Principle of Multi-Threaded Checksum Computation**

Checksum computation is essentially a sequential process as it requires applying recursively the same elementary operation on the data seen as a stream of bytes. OurChecksum computation is essentially a sequential process as it requires applying recursively the same elementary operation on the data seen as a stream of bytes. Our implementation uses a well-known checksum algorithm: adler32 provided by the Zlib library.

Adler32 also provides a very fast (O(1)) way of combining two checksums of consecutive chunks of data that will result in the same checksum as if computed for the whole data at once.

**Implementaiton**

The implementation is such that, if not used (due to configuration or operation conditions), this feature has virtually no impact on performance.

**Computation overhead** (elements in dark gray) comes from: data splitting, thread synchronization for distributing the work and retrieving the sub-results, and combining the results into the original checksum.

To minimize overhead, threads are pre-allocated, mutexes and condition variables are used as synchronization mechanism to distribute the work, and busy-wait loop to retrieve the sub-results (results are very likely to be already available).

**Results**

As compared to a single thread implementation, the performance improvement depends on the number of threads available to checksum computation: the maximum gain is obtained when all available physical CPU cores can be used for checksum computation (~40%).

SMT cores have no impact on performance.