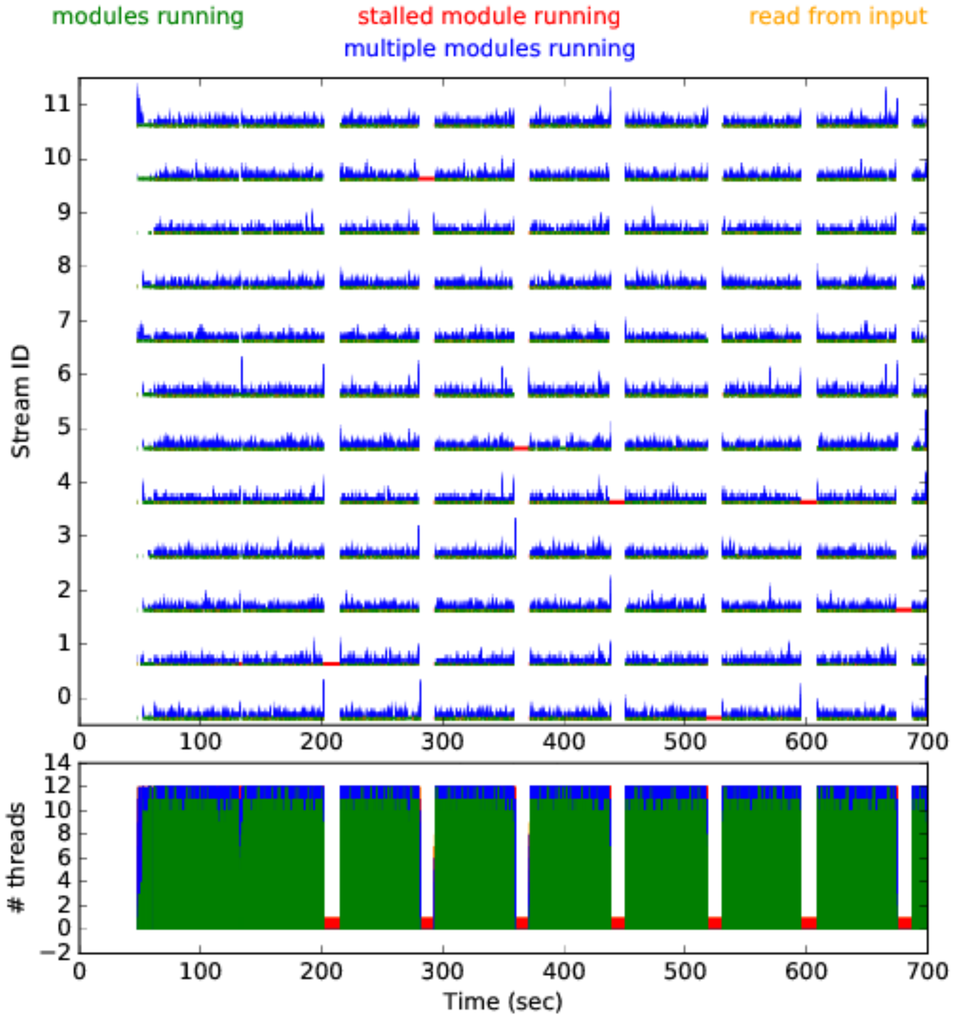
CMS ROOT I/O update

Dan Riley (Cornell)
ROOT I/O Workshop, 2017-10-11

ROOT I/O limits CMS scaling

CMS production jobs are multithreaded

- **Production jobs currently use 4 cores with 4 framework event streams**
- **Output is handled by "one" modules that can only be active on one thread at a time**
- **ROOT output is the dominant source of output stalls**
  - We lose efficiency with more than 4 cores, preventing us going to 8 cores
- **Compression is the principal bottleneck**
  - Especially for AOD and MINIAOD data compress with LZMA



Previously…

Presented some experiments at the 2017-06-12 ROOT IO workshop

- Prototype using TMemFiles as intermediate buffers (based on a concept from Philippe)

TBufferMerger

- Based on the same concept
- Conceptually nice interface that worked well for us
- Developed a new prototype using a new framework class
  - Some success!
  - Also some issues…
- Have been working with the developers to address the issues
  - Have not finished evaluating the latest changes

CMS Implementation

Refactored the CMS output module

- **Kept single-threaded (“one”) output module for cases that are IO bound**
- **Factored out common bookkeeping code**
- **Chris Jones implemented a new “limited” module type**
  - Normal “stream” and “global” modules have parallelism limited only by the thread count; “limited” modules have explicitly limited parallelism
  - Goal is to only have as many TBufferMerger buffers as necessary, not one for every thread
- **Parallel output module uses a tbb::concurrent_priority_queue to manage a pool of output buffers**
  - Priority is set so that the available TBufferMerge with the most entries is used, to prefer filling buffers quickly
  - Minimizes tail and synchronization effects (vs. FIFO/round-robin)

Status

Works well for “MINIAOD” data tier

Issues with “AOD” data tier with our default flush size

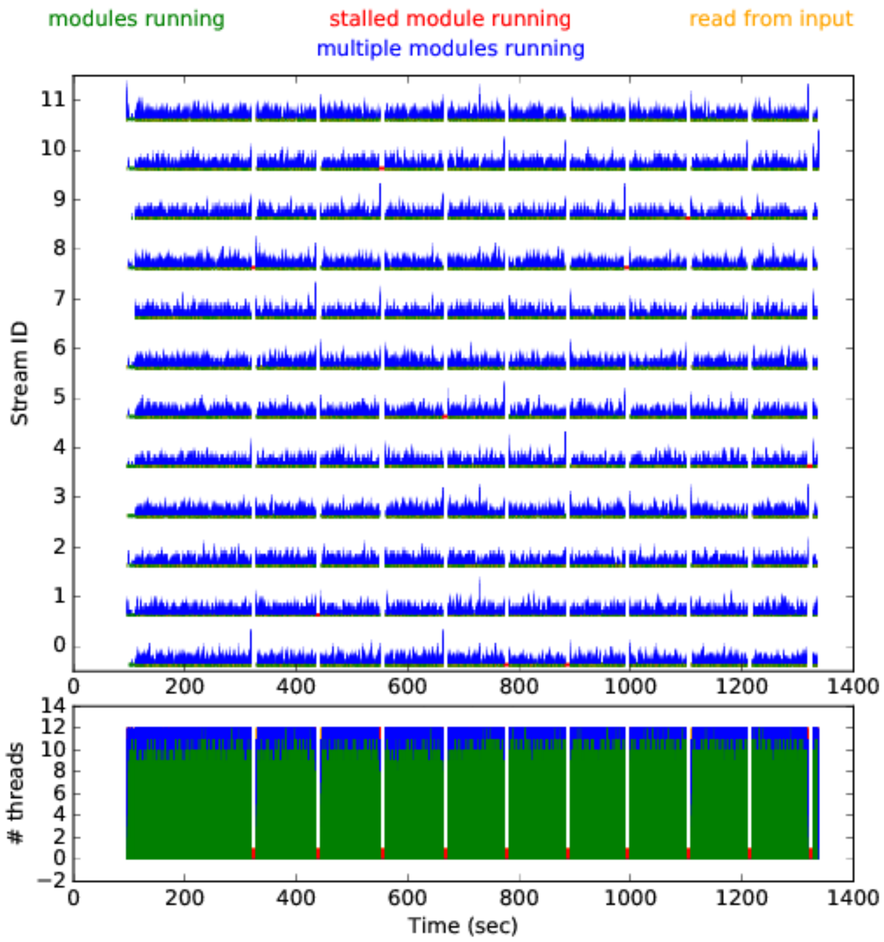
- **TBufferMerger thread ends up doing too much work compressing metadata**
  - Eventually can’t keep up, building up a queue of buffers waiting to be processed
  - Issue 1: amount of work done
  - Issue 2: queue can grow without bound with no feedback to the client
  - Issue 3: TBufferMerger only merged one TBufferMergerFile at a time
  - Issue 4: gROOTMutex scope?
- **ROOT responses (not yet evaluated by CMS)**
  - Callback at merge completion and new function to access queue size
  - SetAutoSave() can set the TBufferMerger to delaying merging
  - Have not evaluated tradeoff between setting the autosave size vs. increasing the flush size
  - Should the merger empty its queue on every merge?

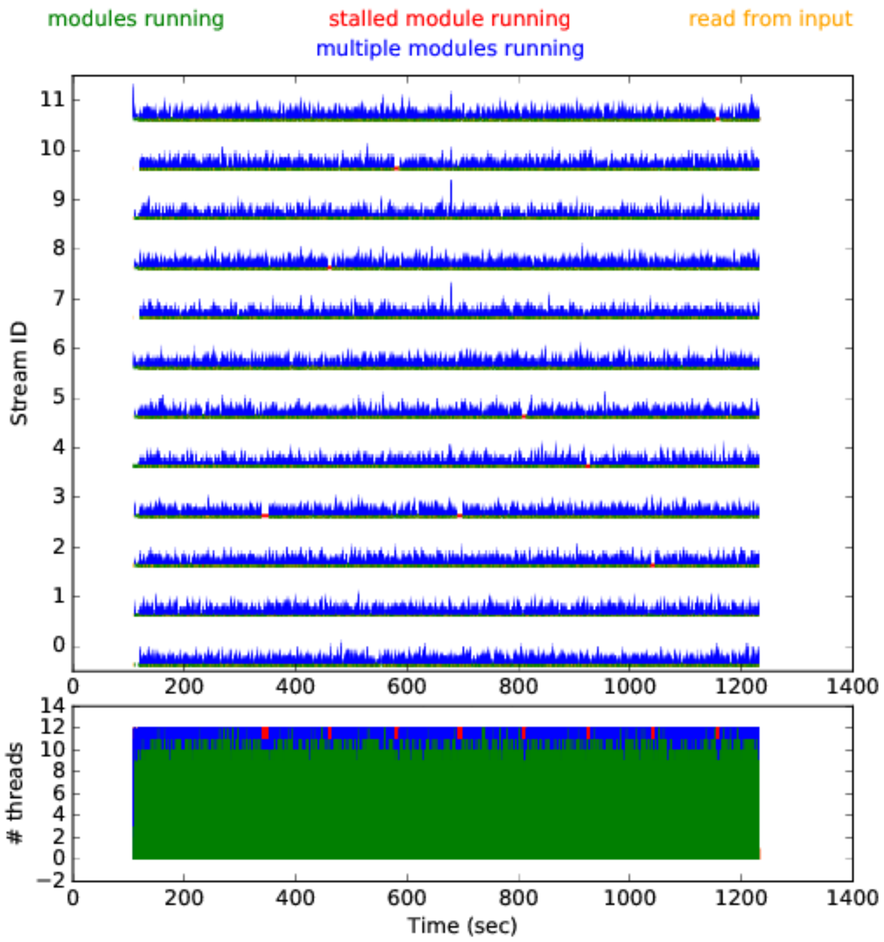
Some results

Results are for a full CMS reconstruction job writing only MINIAODSIM output

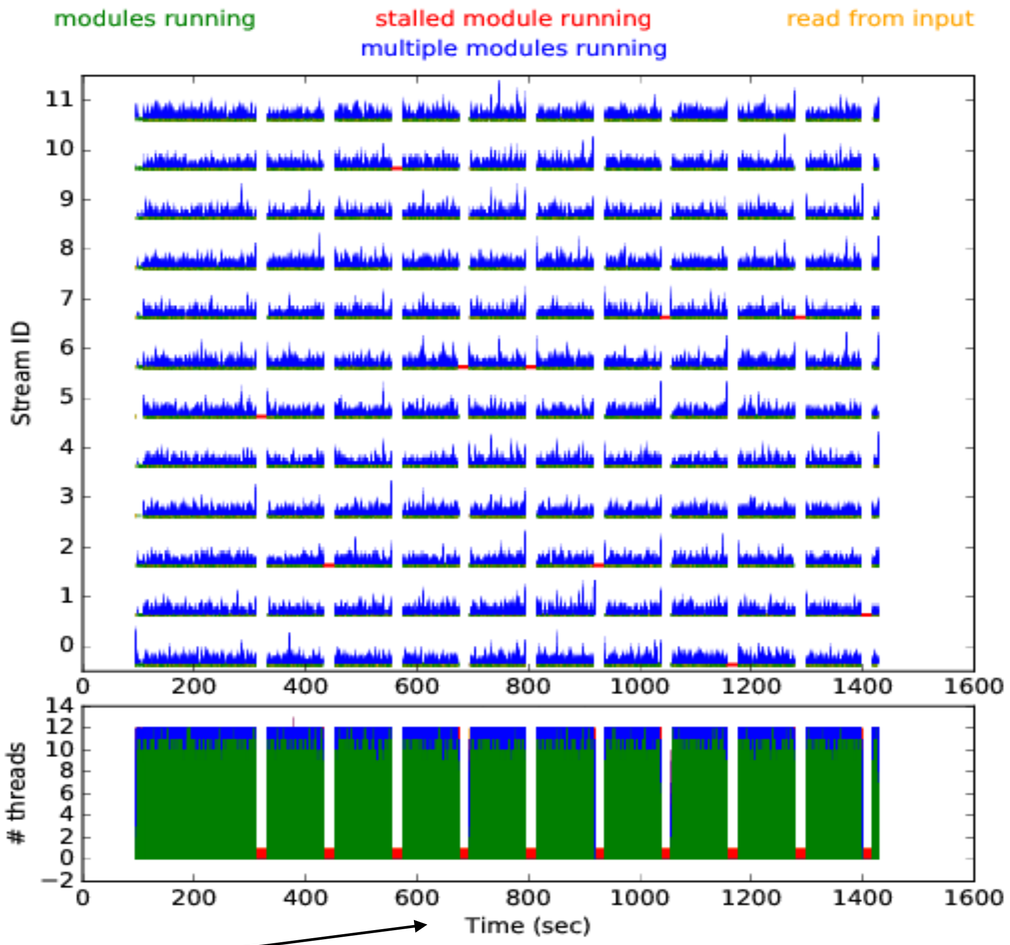
- **12 threads**
- **limited::OutputModule concurrencyLimit set to 4**
  - Could have reduced to 3 or 2, as the third and fourth buffers a barely used
- **10,000 events for stall graphs, 40,000 for statistics**

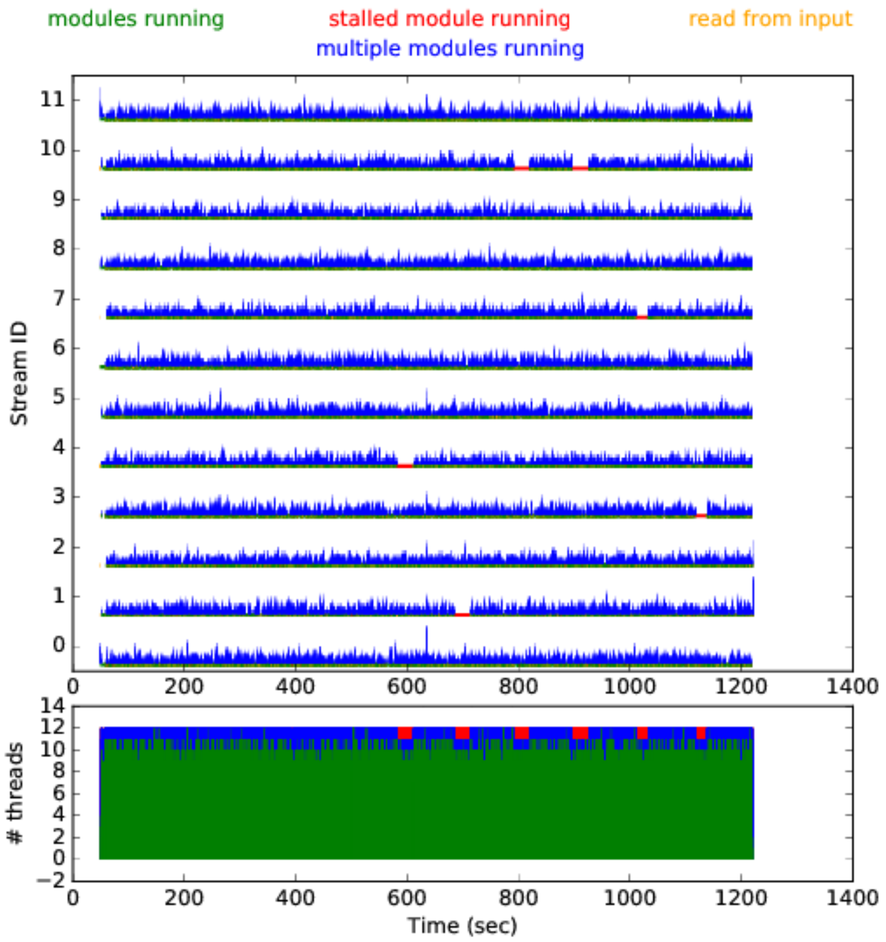
Stall Graph Comparison, LZMA 4





Stall Graph Comparison, LZMA 9





Note scale change

Statistics

40,000 events, MINIAOD LZMA 4, basic statistics:

- Single thread: 7.96 ev/sec, efficiency 0.907

```
StallMonitor>      Module label  # of stalls  Total stalled time  Max stalled time
StallMonitor>  ----------------  -----------  ------------------  ----------------
StallMonitor>  MINIAODSIMoutput         1030           4818.68 s          11.369 s
```

- FIFO queue: 8.62 ev/sec, efficiency 0.960

```
StallMonitor>      Module label  # of stalls  Total stalled time  Max stalled time
StallMonitor>  ----------------  -----------  ------------------  ----------------
StallMonitor>  MINIAODSIMoutput           62           158.863 s          10.649 s
```

- Priority queue: 8.76 ev/sec, efficiency 0.969

```
StallMonitor>      Module label  # of stalls  Total stalled time  Max stalled time
StallMonitor>  ----------------  -----------  ------------------  ----------------
StallMonitor>  MINIAODSIMoutput           39             5.513 s           0.299 s
```

Parallelization reduces # of stalls, “limited” module and priority queue strategy reduces duration of stalls

Next steps…

Use the TBufferMerger callback and queue interrogation

- **Monitor when the merge queue is growing**
  - Log a warning message
  - Defer scheduling writes to keep the queue from growing too large
- **Possibly use to tune the flush algorithm**

Evaluate the new autosave functionality

- **Increasing autosave vs. increasing buffer flush size?**