Enhanced Data Caching for multi-process workflows

David Clark, Peter Van Gemmeren

Argonne National Lab

Outline

- Motivation
- Preloading and Retaining Clusters
- MaxVirtualSize Usage
- Implementation Details

Motivation

- With the change to Athena MT, ATLAS will be transitioning to a multi-threading workflow
- Athena MT will have multiple threads running different calculations on the same data and running several events in-flight
- Due to the random nature of the events, they can take different amounts of time to process
- Since Athena reads data on-demand, this can lead to out of sequence reads

Motivation *cont.*

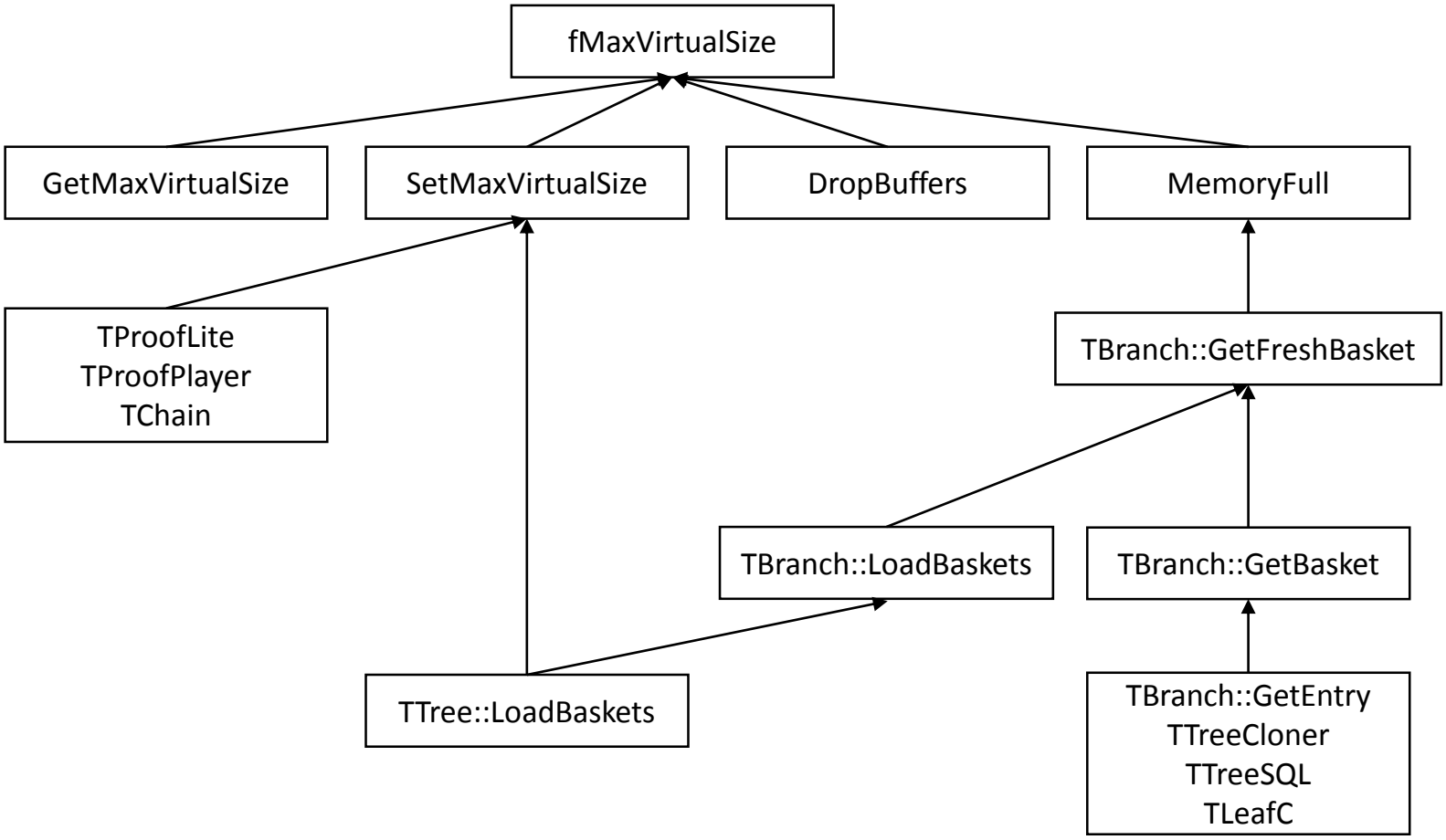
- Cache clearing
  - Any thread reading a branch from the next cluster will cause the tree cache to be cleared, even as other threads are still processing that data
  - All branches that will read previous entries, that are not yet loaded into memory, will cause disk reads.
    - And if they call LoadTree(), this will cause cache thrashing. Without LoadTree this will result in a small disk read.
- Single branch backward reads
  - Branches keep the current basket in memory. But, if a branch tries to read an entry in a previous basket, it is forced to reload that basket to memory
    - If the MaxVirutalSize of the tree is set, it will keep that many bytes in memory. But once the tree reaches the max size, DropBuffers is called and all of the previous baskets are removed from memory and there is a chance other baskets will be needed again
- Remainder branch reads in trailing baskets
  - Clusters may have multiple baskets and often times the second basket is small
  - If one processor triggers a cache flush and loads the next cluster, some of the trailing baskets may not have been read into memory yet, and will result in a small disk read.

Preloading and Retaining Clusters

- Branches will load an entire clusters into memory
- Branches will keep the current and previous cluster in memory

Read Calls		**Tbaskets**				
GetEntry(0)	post-change MaxVirtualSize<0	0-59	60-99 preloaded	100-159	160-199	200-259
	pre-change					
GetEntry(60)		0-59 retained	60-99	100-159	160-199	200-259
GetEntry(59)		0-59 extra	60-99	100-159	160-199	200-259
GetEntry(61)		0-59	60-99 extra	100-159	160-199	200-259
GetEntry(100)		0-59	60-99	100-159	160-199	200-259

MaxVirtualSize Usage



MaxVirtualSize Usage *cont.*

- SetMaxVirtualSize
  - TProofLite & TProofPlayer sets maxvirtualsize to 0 then drops all the baskets from memory
  - TChain sets the maxvirtualsize of a fTree when it loading the tree
  - LoadBasket sets the MaxVirtualSize, but is for random access of entries
- DropBuffers
  - Called to remove buffers in memory until the total bytes in memory is less than fMaxVirtualSize.
  - fMaxVirtualSize being negative would be equivalent to it being set to 0
- GetBasket
  - Retains the same functionality

Preloading Entire Clusters

- In GetEntry, when a new basket is going to be loaded, the rest of the cluster is also loaded
- ClusterIterator allows for easy computation of the beginning of the next cluster

```
if (fTree->GetMaxVirtualSize() < 0) {
   TTree::TClusterIterator clusterIterator =
                              fTree>GetClusterIterator(entry);
   clusterIterator.Next();
   Int_t nextClusterEntry = clusterIterator.GetNextEntry();
   for (Int_t i = fReadBasket; i < fMaxBaskets && fBasketEntry[i] < nextClusterEntry; i++) {
      GetBasket(i);
   }
}
```

GetFreshCluster

- Added GetFreshCluster function to Tbranch
- Responsible for return new basket and cleaning up old clusters

```
+   TBasket *GetFreshCluster();

-   ClassDef(TBranch,12);  //Branch descriptor
+   ClassDef(TBranch,13);  //Branch descriptor
```

- Use new function to get fresh baskets in GetBasket

```
-   basket = GetFreshBasket();
+   basket = (fTree->GetMaxVirtualSize() < 0) ? GetFreshCluster() : GetFreshBasket();
```

Finding Entry

- Uses ClusterIterator to find cluster to be flushed from memory – this is the cluster that is two back from the current one

```
+   TTree::TClusterIterator iter =
                              fTree->GetClusterIterator(fBasketEntry[fReadBasket]);
+   if (iter.GetStartEntry() == 0) return fTree->CreateBasket(this);
+   TTree::TClusterIterator prevIter =
                              fTree->GetClusterIterator(iter.GetStartEntry() - 1);
+   if (prevIter.GetStartEntry() == 0) return fTree->CreateBasket(this);
+   Int_t entryToFlush =
                              fTree->GetClusterIterator(prevIter.GetStartEntry() - 1)
                              .GetStartEntry();
```

Finding Basket

- Once we know the entry index to start clearing from memory, we need the basket number
- Since the number of baskets per cluster should be small, just iterating backwards is efficient

```
+   Int_t basketToFlush = fReadBasket;
+   while (fBasketEntry[basketToFlush] != entryToFlush) {
+      basketToFlush--;
+      if (basketToFlush < 0) {
+         return fTree->CreateBasket(this);
+      }
+   }
```

Reusing Basket

- Reuses the first basket that needs to be flushed, if it exists, otherwise create a new one

```
+   TBasket *basket = (TBasket*)fBaskets.UncheckedAt(basketToFlush);
+   if (basket) {
+      fBaskets.AddAt(0,basketToFlush);
+      --fNBaskets;
+   } else {
+      basket = fTree->CreateBasket(this);
+   }
```

Clearing the Cluster

- Clears the rest of the baskets and recalculates the last basket in memory

```
+   while (fBasketEntry[basketToFlush] < prevIter.GetNextEntry()) {
+      fBaskets.AddAt(0, basketToFlush);
+      --fNBaskets;
+      ++basketToFlush;
+   }
+   fBaskets.SetLast(-1);
+   return basket;
```

- It would be ideal to reuse these baskets for other baskets in the new cluster
- This would require GetFreshCluster to go beyond its current scope
- For the cases where each cluster only contains a single basket, all the baskets will be reused

Results

- Reading the first 1000 (~1 GB) entries on test on dummy tree consisting of 2000 branches
- Every read has a 2.5% chance of reading 10 entries back from the current entry or a 2.5% chance of reading 10 entries forward

MaxVirtualSize	Bytes Read	Read Calls	Uncached Reads
0	1507824429	31102	31012
280MB	1394073580	24877	24787
600MB	1236320410	8905	8815
-1	1131937563	90	0