

In the past year, the development of **ROOT I/O** has focused on improving the existing code and increasing the collaboration with the experiments' experts. Regular I/O workshops have been held to share and build upon the various experiences and points of view. The resulting improvements in **ROOT I/O** span many dimensions including reduction and more control over the memory usage, drastic reduction in CPU usage as well as optimization of the file size and the hardware I/O utilization.

Many of these enhancements came as a result of an increased collaboration with the experiments' development teams and thanks to their direct contributions both in code and to the quarterly **ROOT I/O** workshops.

Performance Enhancement

Many areas of the **ROOT I/O** and **TTree** packages have been made more efficient in either runtime or memory usage.

Significant improvement of the performance of **SetBranchAddress/SetAddress** (by a factor 3 to 10 depending on the length/complexity of the classname).

Prevented the unlimited growth of the **TBasket's** buffer even if the basket is reused and minimize the number of buffer reallocations.

Optimized the use of the **TTreeCache** memory by insuring that it is completely used when the clusters are small and does not grow without bound when the clusters are unusually large.

Extended the amount of statistics recorded by the **TTreePerfStats** class.

Multi-Thread

gDirectory/gFile are now thread-local. Completely removing the dependency requires backward incompatible changes in **TObject**.

Asynchronous Prefetching

When the asynchronous prefetching is enabled, a separate thread will fetch the **TTreeCache** blocks that are likely to be needed next and the main thread continues the normal data processing.

In order to enable prefetching the user must set the **rootrc** variable **TFile.AsyncPrefetching**.

```
gEnv->SetValue("TFile.AsyncPrefetching", 1)
```

In addition the result of the prefetch can be cached on local disk.

```
TString cachedir="file:/tmp/xcache/";
// or using xrootd on port 2000
// TString cachedir =
// "root://localhost:2000/tmp/xrdcache1/";
gEnv->SetValue("Cache.Directory", cachedir.Data());
```

Features

Object Merging

We introduced a new explicit interface for providing merging capability. If a class has a method with the name and signature:

```
Long64_t Merge(TCollection *input, TFileMergeInfo*);
```

it will be used by a **TFileMerger** (and thus by **PROOF**) to merge one or more other objects into the current object.

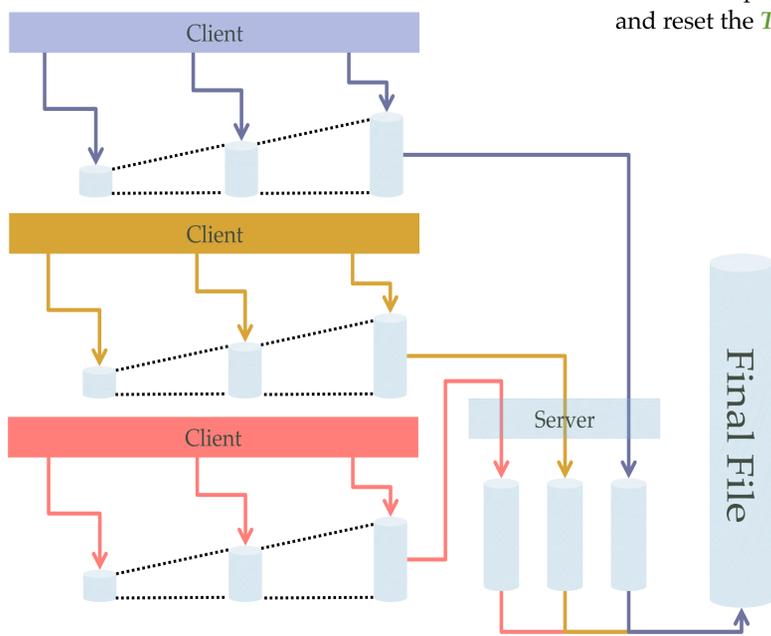
If this method does not exist, the **TFileMerger** will use a method with the name and signature:

```
Long64_t Merge(TCollection *input);
```

The object **TFileMergeInfo** can be used inside the **Merge** function to pass information between successive runs of the **Merge** operation.

Parallel Merging

Existing solution: Processing and storing the partial output on each local node and then **only** at the end of the process, upload to the server and only when all slaves are done, read all files on the server and write to a single output file.

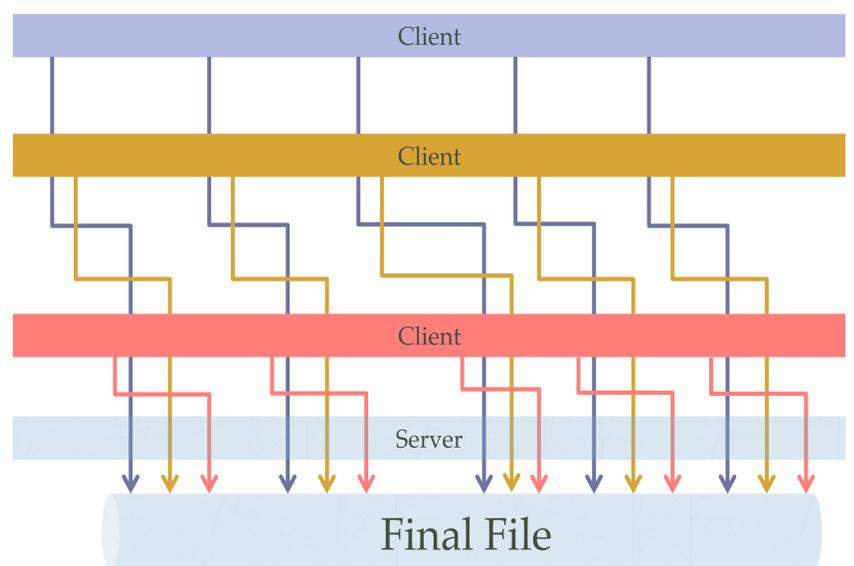


New TFile implementations:

- **TMemFile**: a completely in-memory version of **TFile**.
- **TParallelMergingFile**: a **TMemFile** that on a call to **Write** will upload its content and reset the **TTree** objects.

New solution: Increase parallelism by having the slaves start uploading the **TTree** clusters directly to the server which immediately starts saving them in the final output file.

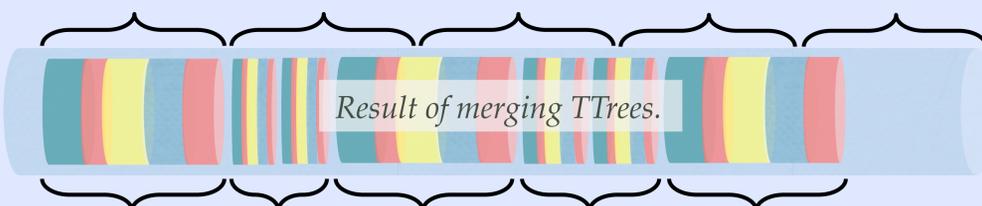
```
TFile::Open("mergedClient.root?pmerge=mergehost:1095", "RECREATE");
```



Variable TTree cluster size

A **TTree** cluster is a set of baskets containing all the data for an integral number of entries that will be read in a single I/O operation by the **TTreeCache**. Each **TTree** has a different cluster size. When merging files, the **TTree** now records the cluster size of each of the input **TTree**, resulting in higher I/O throughput when reading the merged file.

With fixed cluster size



With variable cluster size

Automatic support for more than one TTreeCache per file.

- **TTree::SetCacheSize(Long64_t)** no longer overrides nor deletes the existing cache
- Each cache is independent
- So the worst case scenario is the rare occurrence of two large **TTree** that are strongly intertwined in the file.

```
TTree *tree1, *tree2;

input.GetObject("tree1", tree1);
tree1->SetCacheSize(300*1024);

input.GetObject("tree2", tree2);
tree2->SetCacheSize(400*2048);

tree1->GetEntry(entry1);
tree2->GetEntry(entry2);
```

LZMA Compression

ROOT I/O now supports the **LZMA** algorithm to compress data in addition to the **ZLIB** compression algorithm. **LZMA** compression typically results in smaller files, but takes more CPU time to compress data.

Setting the Compression Level and Algorithm

There are three equivalent ways to set the compression level and algorithm supported by the classes **TFile**, **TBranch**, **TMessage**, **TSocket**, and **TBufferXML**.

For example, to set the compression to the **LZMA** algorithm and compression level 5.

```
TFile f(filename, option, title);
f.SetCompressionSettings(
    ROOT::CompressionSettings(ROOT::kLZMA, 5));
```

```
TFile f(filename, option, title,
    ROOT::CompressionSettings(ROOT::kLZMA, 5));
```

```
TFile f(filename, option, title);
f.SetCompressionAlgorithm(ROOT::kLZMA);
f.SetCompressionLevel(5);
```

