

SAM File Merging - Version 3.5

H. Schellman

June 14, 2000

Introduction

We need to be able to merge input files in order to maintain reasonable files sizes in the D0 data store. Merging files themselves is reasonably easy, the D0 data model has separate read and write modules which can open and close files at different rates. Merging metadata is somewhat more difficult as input files may not have exactly the same metadata. In this note I consider the various metadata considerations.

At the end I have a plan which actually has several stages.

1. Make it possible to store files with several parents by making run/event tags (but not number of events) some null value and relying on lum blocks and parentage to keep the information. We need a minimal functionality for this before we can write merged rootuples.
2. Explore the possibility to have intermediate files in sam which are used in the merging process but never written into the robot.
3. At some future date, make it possible for users to put in some more complicated information (such as run range for input/output).

1 Merging

1.1 What merging means

By file merging, I mean any process which takes a set of input files and produces a set of output files which have a many input to one output file relationship.

This can be

- a special step in processing where the only operation performed is to create a bigger file from smaller ones, or it can be
- a natural part of other processing, for example a job which reads in 10 input files in DST format and produces 1 output file in Root format.
- It can also be a selection, where 10 input files in DST format have 1/10 events selected to produce a single input file in DST format.

In all these cases, what sam needs to know is what files were merged and what did the merging.

In this memo, I concentrate on the database structure and the sam store description files, not the framework interface. In the end almost all of this should be automated via the framework interface as that will prevent user error.

D0 file i/o supports concepts like closing the output file every N file boundaries and can ensure that output files have a unique set of parents. This is good as file lineages are much easier to trace if a child file cannot be partly from one and partly from another parent.

1.2 What merge does not mean

D0 will in fact combine files in two ways, which I will call **parallel** and **serial**. A **parallel** combination is done when events from two different streams (MC and 0-bias) are combined event by event into a single event stream. I do not discuss this case in this memo except to suggest that 'merge-type' information is probably needed in the file database to distinguish this case from the more common **serial** merge in which events are read in sequentially from files and written out to a single file in the order they were read in.

2 Splitting

Splitting can also have two meanings.

selection A particular set of events within a file is 'selected' and written to a given selection stream. Generally the whole input file is read and one output per selected stream is written. The luminosity info for the

input file is the same as for the output files if they are closed on input file boundaries.

division An input files is too large, it is subdivided into N smaller chunks each of which corresponds to a chunk of the input file. In this case the luminosity information is not the same and needs to be subdivided among the divisions.

In HEP experiments splitting normally means the **selection** of subsamples, not the division of a large file into smaller pieces.

We need to have methods for doing both of these. For this memo, selection is considered and we assume that output files are closed on (possibly multiple) input file boundaries. In this case an input file can have multiple children but each child has a unique selection stream. An output file can have multiple input files as parents.

Do we have a concept of selection stream? Any given file can have multiple such selections applied sequentially and we need to keep track of this. It can be done through the code versions but it may be much clearer if it is stored as a DB entry on its own.

3 Life of a file

Let's look at W triggers.

3.1 Data logging

These will be written to the exclusive **electron+MET** and some **electron+MET+other** streams by the data logger. The **electron+MET** stream is around 1% of all events and the **electron+MET+other** streams are 2%. The W sample thus comes from several streams which will be logging data at ~ 1 Hz or 250kB/sec each. For 1GB files this means an output file per stream every hour or so.

Each W file will span around 60 luminosity blocks and will be closed and opened by the data logger on lum block boundaries.

A data taking run will probably last several hours so one can expect that a stream that writes 1% of the data will produce several files per run.

Assuming that we run 50% of the time during a year, there will be around 2,500 **electron+MET** and 5,000 **electron+MET+other** files per year.

3.2 The farm

On the farm, once a run ends a W processing job will be fired up and will run on N processors, each getting W files from a sam consumer shared between the N processors. The processing reads in one raw file and produces one or more output files. These output files can be:

DBG full reco output - 250kB

DST most useful reco output - 120kB

XXX special events - streams for hot samples

TMB thumbnail output - 10kB

Rtpl ntuple output - 5 kB

each of these has the input file as a parent but some are a lot smaller than others and will need to be merged into larger files before they go in the robot.

In principle some of these files could be 'merged' by the reco process itself, by having reco read 2 and write one file. Given that we may wish to have reco do splitting of data via a Level 4 trigger, and that the files on any given worker node are almost certain to be widely separated in luminosity block and perhaps from different runs, this cannot be the only form of merging.

On the completion of processing for each input file, the output files and their metadata are written to one of the 72 GB disk buffers on the farms for merging. Each different data tier goes to a different directory.

3.3 The merge area

We wish to merge several input files and write them back into sam as a single file. It is desirable that those files be as close to in time order as possible. This minimizes confusion for users and decreases the number of DB accesses needed to get constants. D0 in run I and CDF in run II imposed absolute ordering on files when they were merged. I consider such ordering to be a very desirable feature and do not see any real problems in achieving it.

In such a standalone merge step after a separate processing step we can do one of two things.

1. Declare the reconstructed files to sam but not store them in encp. Make a project consisting of convenient subsets of those files. Then run a normal framework job which uses sam to access the reconstructed files in the normal fashion and merges them. The merge would then consider the reconstructed files as the 'parents' and the merged file which actually goes into sam is the 'child'. The parentage is then:

raw -> reconstructed -> merged

and there would be separate processing information for the reconstruction and merging phases. This method can make full use of the sam framework and process tracking machinery and is the one I prefer.

2. Construct a list of reco files in the merge area, use the framework to do a file merge.

If one has the metadata files produced by the reco jobs on the worker nodes, one can write a simple merge script which takes the metadata for each file and produces metadata for the merged file.

One can then store back into sam.

The parentage is:

raw -> merged

This method saves a dummy file save in SAM but makes it harder to track the merge code version numbers as the 'application/version' info are for reco, not for the merge. It also prevents the merge step from using the sam project and framework tools.

One now has a set of reconstructed files of the right size stored on disk with correct parentage. These merged reco files are then stored into sam via a sam store command.

For 1 GB merged files

DBG full reco output - 1-1

DST most useful reco output - 2-1

XXX special events - streams for hot samples 10-1

TMB thumbnail output - 25-1

RTPL ntuple output - 50-1

If we avoid crossing run boundaries, the TMB and RTPL files will probably be smaller than 1 GB.

3.4 Later selection

The W ID group will process the DST files using the freight train on d0mino with a selection which requires the official W trigger and a PT cut on the electron at 15 GeV to produce the 'official' W sample. This official sample may be 20% of the reco sample and probably involve reading 5 files and writing 1. At this point one has ~ 500 files for the `electron+MET` stream and ~ 1000 for the `electron+MET+other` stream. This is a dataset of tier 'DST' but with a documented selection applied to it.

These output files are part of a selection stream 'Wofficial', a subset of these events might go into a tighter selection stream 'Wtight'.

Because these subsets only have selections applied to them, if the output files are closed on inputfile boundaries, Co the parentage and luminosity book-keeping should be straightforward.

3.5 Ntuple creation

The W group will then process the 'official' W sample to get an official W ntuple in ROOT format consisting of 5kB of information per event. This reduces to perhaps 25 `electron+MET` and 50 `electron+MET+other` files, it could be less but one may wish to have each file correspond to a fixed trigger or streaming list.

3.6 Ntuple selection

The W group might want to make a smaller ntuple with, for example, the W mass analysis selections. This would be done via another selection into stream 'Wmass'.

This sample then has tier Ntuple and 2 selections 'Wofficial' and 'Wmass' applied to it.

4 Determining the luminosity for the W analysis

In the end one needs to find the luminosity for the W analysis.

- One takes the **RTPL** files and finds the complete list of their **W-select DST** parents,
- one then takes the **W-select DST** parents and finds their **W-stream DST** parents,
- one then takes the **W-stream DST** files and finds their ghostly **reco** parents
- one then takes the **reco** parents and finds their **RAW** parents

The luminosity for the W sample is the sum of the luminosities for all of the **RAW** parents.

If the **electron+MET** and **electron+MET+other** streams come up with different lists of raw luminosity blocks due to processing losses, only the lum blocks which are in both lists are valid - the user will have to take that list of valid lum blocks and remove any events which are from bad ones from the data sample.

One can assume that people will be doing such queries on a reasonably regular basis, perhaps as an overnight job.

If at any step in this chain a selection or merge of a set of files yields 0 events, a file with 0 events but with parents must be entered into the DB and included in any further family tree - this is probably not a real problem for the high rate W sample but it will be for rarer events, which do not show up in every file.

The file tracking needs to be able to include such ghost files and include them in the parentage of children. If one uses SAM, I think this is almost guaranteed. If one does not - the odds are very high that one will get burned.

5 Metadata relevant to merging

The goal of file merge metadata is to allow one to find the parents of any merged file and hence the processing history and luminosity associated with

that file. Some pieces of metadata are very useful here while others are not and different merging models may wish to require or ignore consistency in the important metadata.

I propose that the user be able to tell sam whether consistency is required or not.

I list the metadata objects most likely to be useful in using a merged file. How they should be treated depends on the application and one should be able to specify a treatment option depending on what one is doing. One can constrain, ignore, list or save a range when confronted with metadata from different parent files.

One might wish to require that the parent files have the same trigger list or reco version. Each merged file might have the lum block range of the parent files. It is also useful to see if any given event can possibly be in a file by checking the event/run number range for the file itself.

Here is a partial list of the metadata which is relevant in file merging and the reasonable behaviour for each type.

`lum_min, lum_max` Should be min and max for the input files.

The luminosity blocks are unique and can be used to determine information like run and event ranges. If lum blocks are handled correctly the rest can be derived.

`run_number` If same, retain, if not set a general flag showing it is merged.

`run_type` Probably want to be able to constrain this.

`first_event, last_event` First and last events, if the run number is the same, the merged file should have the minimum of the first events and the maximum of the last. (Are these for read or written?). If the file contains several runs, us the merged flag.

`event_count` Should be the event count for the final merged file. Could be 0. File should still be declared to sam.

The lum information is most important and should contain the run/lum block range for the input files. The event info is less important and should perhaps refer to the output file.

Because one does have the file parentage and that is the real information about the files origins, all of this information is useful but not absolutely

necessary. However, it may be very useful in shortening the time of a file query or constants download. The parentage gives you the lum/run/event range for the INPUT files. For constants downloads, you only need the lum range for the file itself. Since lum ranges are unique and already exist, they should be updated for merged files to contain the lum range for the events actually present in that file (as calculated by the framework job) or if unknown, the lum range for the input files.

5.1 more information

Parent `application_name`, `application_version` Generally want to constrain to same or similar.

Parent `physicaldatastream` May be same, may be different

`logicaldatastream` What do we do with this?

`trigger_stream` ???

Parent `trigger_list` Is merging across trigger list boundaries wise? This should be an optional constraint.

Parent Monte Carlo generation variables (not on the DB picture I have but should be required to be consistent.

Parent Data Tier

Each of the above is information which may or may not be required to be consistent in a file merge. For consistent information, the child metadata inherits from the parent. For inconsistent information there are choices as to how to construct the metadata for the child file.

As with any sam process one also needs to store metadata about the process which did the merging. If the merge was run as a sam project, then the pid alone can point sam to the rest of the metadata. But if the merging is not done by a project, then things like the `ApplicationVersion`, `ApplicationName` etc. of the merge process need to be logged when the file goes into sam.

The bottom line is that the outputs of the merging process must have sufficient associated information to build a description file that sam can use to correctly associate them with their parents. This means a list of the parents,

information about the merge process itself and rules for combining the metadata of the parents to make metadata for the child files. This information should be passed to sam by the description file used in sam store.

6 Producing MetaData for a merged file

The metadata for a merged file are derived from the metadata for the parents + the metadata for the merging process itself. Most of the mechanisms for this are already in place, although the current 'sam store' description files do not yet accept a list of filenames as parents.

Here is a possible description for a merged file. One has the usual file metadata and then hints to sam on how to handle the metadata inherited from the parents.

```
TheFile = MergedFile(  
  name = "outputfilename",  
  parentnames = {"file1","file2","file3","file4"},  
  start_time = "...",  
  end_time = "...",  
  ApplicationName=mergorama, // tells you the merge process info  
  ApplicationVersion=pmc03.01.00 //  
  sizeK = 1345222,  
  MergeRunNumber = "Merged",  
  MergePhysical_Stream = "Constrain",  
  MergeApplicationName = "Constrain",  
  MergeApplicationVersion = "SQL:%preco3.07%"  
  MergeLumBlocks = "MinMax"  
  MergeLogicalStream = "Constrain"  
  MergeTriggerList = "Constrain"
```

Here there are several hints on how to construct the metadata which have been specified by the user.

First cases where the metadata are obvious once one has decided whether or not to make a constraint.

Constrain means require consistency and put the consistent value in for the child file.

Merged means general merge with metadata set to a null value.

Then cases where one needs to decide exactly what to store. These differ as the storage type in the database differs.

MinMax takes the minimum and maximum,

List stores a list of possible values

SQL: This is a bit advanced - means constrain to a certain subset, then store the constraint.

For a first pass, the **Contrain**, **Merged**, **MinMax**, and **List** behaviours are needed.

7 Empty files

In file merges it is often important to include an empty file in the merge records, because it still has associated luminosity data.

8 Merge/split

In principle a single process can both merge and split, leading to a file which has multiple parents and children. We may wish to disallow this and have 0-length intermediate files which keep the parentage 1-many, many-1.

9 Plan of attack

1. Identify any changes needed in the DB schema itself. Unless run ranges are added, I do not see any changes.
2. Check that sam can support the declaration of intermediate 'ghost' files in a parentage chain. What is exact command syntax for this?
3. Identify changes needed in the sam store description file.
 - NOW: add support for lists of parents
 - NOW: confirm that merge process information can be added properly even if the merge is not a sam 'project' run.
 - LATER: add support for constraints

- LATER: add support for controls in creating output file metadata in unconstrained case.

For a start, one could not implement constraints, use inheritance where possible, use min/max for lum as suggested above and flag any ambiguous metadata as 'go ask your father'.

4. Change the sam store description file interface to support merging
5. Test and document
6. Put into the framework SAM interface
7. Do a test from RAW \rightarrow ntuple
8. Test performance of 5 level parentage queries.