

1. Database Luminosity Tracking

1.1. Database Luminosity Tracking Concept

Vicky's stuff goes here.

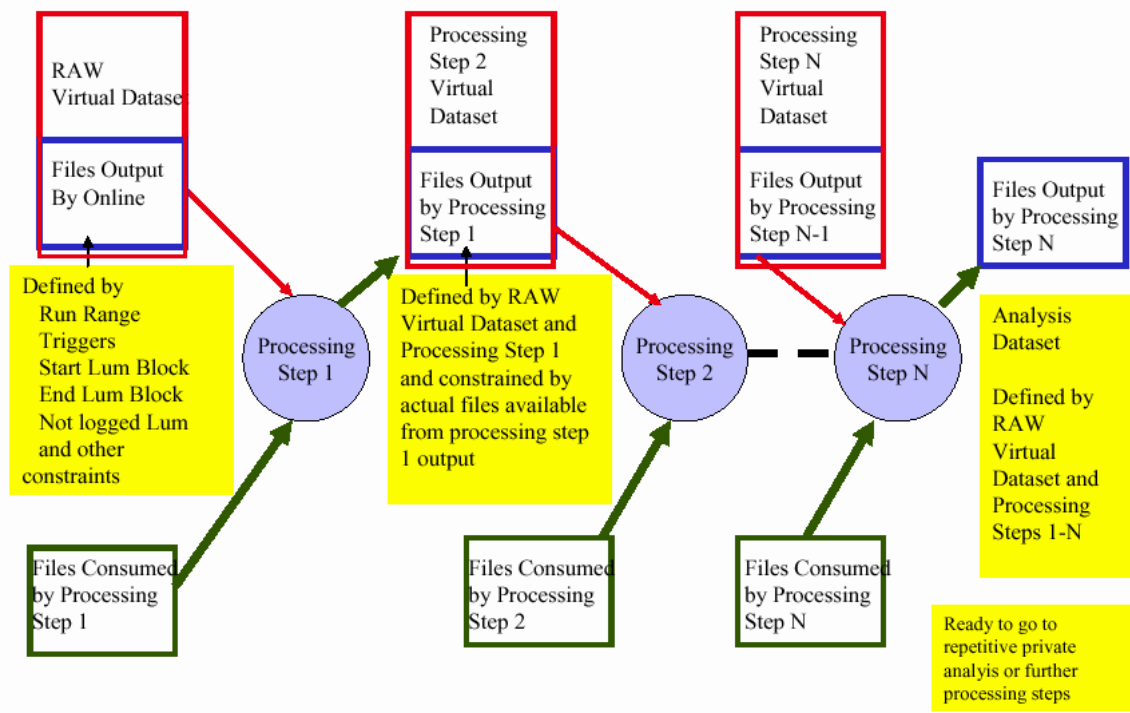


Figure 1-1: Conceptual outline of the database luminosity tracking algorithm.

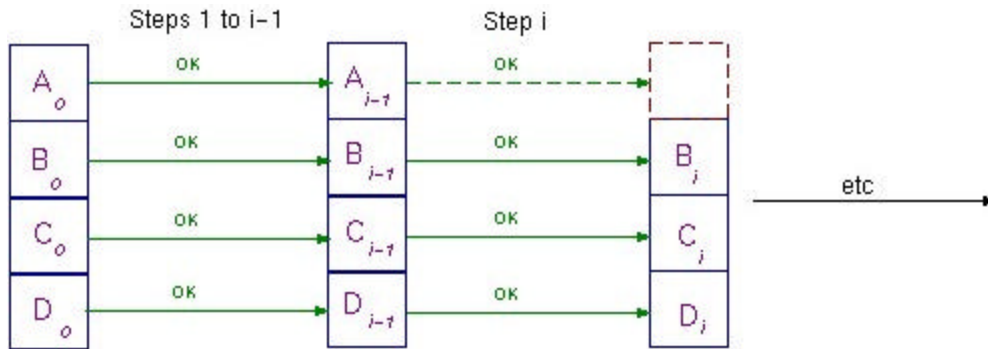
1.2. Database Luminosity Tracking Use Cases

This section describes several “plain English” use cases for the outlined database luminosity tracking concept, which illustrate use of the proposed algorithm.

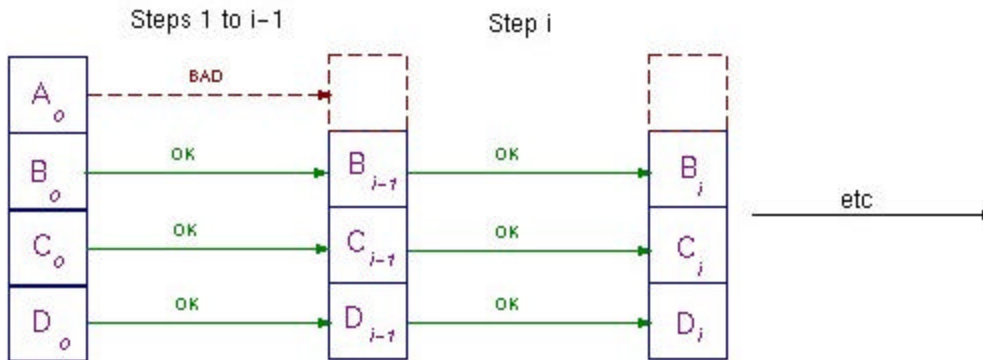
1.2.1. Filtering With the Zero-Length Output

Consider the following example. A stream, ELE_HIGH, is being filtered by the user to create a root-tuple from the events that contain both an electron and a muon. As a result of this filtering, one of the input partitions corresponds to a zero-length output, i.e. no output file is created for this input partition. How would the database approach handle “missing” luminosity blocks?

Let's assume that the filtering in question occurs on the processing step i , as shown in Fig. 1-1. The input file that was reduced to zero events as the result of filtering is present in the list of the input files for the processing step i (which is the same as the output list from the processing step $i-1$, see the blue part of the red box in Fig. 1-1). Since the file was not lost, it is also present in the list of files actually consumed by the processing step i (see black box in Fig. 1-1). However, since no output was produced, the corresponding "image" is not present in the list of output files for processing step i , or any subsequent steps. Since the luminosity calculation involves tracing the processing steps backward, all the way to the original list of RAW data files, the information of the "missing" luminosity blocks is immediately recovered as soon as one traces the data flow all the way back to the processing step $i-1$. This is illustrated in Fig. 1-2. Since the zero-event file has never been written out, it cannot be lost at any further processing steps, and therefore the database approach is fully operational in this use case.



- Figure 1-2. Tracing back to processing step i says: in=[A,B,C,D] of $i-1$ and out=[B,C,D] of i . So, the "missing" file line A is "recovered" from the input to the processing step i . All the luminosity information is recovered.



- Figure 1-3. Tracing back to step $i-n$ says: in = [A,B,C,D] of $i-n-1$, out = [B,C,D] of $i-n$, and A is not processed. We therefore know that the luminosity blocks of the missing file A have to be marked as bad. All the luminosity information is recovered. We now can generate the list of events corresponding to bad luminosity blocks to be removed from the user's output to provide accurate luminosity for the sample.

Finally, if the partition that would have given the zero event output is lost on one of the previous processing steps, this partition has been already added to the XLumFileList, and therefore all the corresponding luminosity blocks are already marked as "bad" by being added to the XLumBlockList. Therefore, these luminosity blocks won't be used for luminosity calculations, and corresponding events will be added to the XEventList and passed to the user to be excluded from the final analysis. This is illustrated in Fig. 1-3. If there was no merging between the streams involved in any of the processing steps, user would never find any of these events in his sample, and therefore does not have to do anything. If a cross-stream merging has occurred, certain events on XEventList might have to be

removed from the final data set, upon which the luminosity information will still be precise. Details of handling cross-stream merging are discussed in the last use case.

1.2.2. Merging Several Partitions

Consider the following example. A stream, ELE_HIGH, is being filtered by the user to create a root-tuple from the events that contain both an electron and a muon. The resulting output is typically small, compared to the size of input, and therefore it is desirable to merge several of the output partitions. Imagine that one of these merged partitions is lost afterwards. How would the database approach account the lost data in calculating the total luminosity?

Let's assume that merging was done during processing step i , as shown in Fig. 1-1. Since the merged partition is then lost, the list of files consumed by the processing step $i+1$ will differ from the list of the files output by the processing step i by the lost partition. This file is therefore added to the XLumFileList, and the luminosity blocks corresponding to this file are traced all the way back to the original RAW files using the parentage information. The parentage information will recover all the luminosity blocks corresponding to the merged partition, i.e. the union of the luminosity blocks corresponding to each of the input partitions that were merged into the lost one. Therefore, all the luminosity blocks corresponding to the lost merged partition will be added to the XLumBlockList and the luminosity loss is calculated properly. No additional events have to be removed from the output data by the luminosity-conscious user in this case, so no XEventList file is generated.

1.2.3. Splitting a Partition

Consider the following example. A stream, ELE_HIGH, is being reconstructed by a RECO program from the RAW data files. Since the RECO output is generally bigger than the input RAW file, for some of the input partitions the output will exceed the maximum allowed file-size, and such a partition will be split in two or more. Imagine that one of the split partitions is further lost. How would the database approach account the lost data in calculating the total luminosity?

Let's assume that the splitting was done during processing step i , as shown in Fig. 1-1. Since one of the split partitions is then lost, the list of files consumed by the processing step $i+1$ will differ from the list of the files output by the processing step i by the lost partition. This file therefore has to be added to the XLumFileList. Since we do not know what luminosity blocks correspond to the lost split partition (even if the split is done on the boundary of the luminosity block, there is no way to guarantee that the events from the very next luminosity block will be in the subsequent output partition), all other daughters of the partition which has been split at the processing step i also have to be added to the XLumFileList, which is equivalent to adding the parent file to the XLumFileList, which is what actually happens. (If the parent in its turn is also a result of a split, than its siblings are collected as well, and the process ends up with the lowest unsplit partition in the processing tree shown in Fig. 1-1.) The parentage information is then used to obtain the list of luminosity blocks corresponding to this unsplit partition, and they are added to the XLumBlockList. Since some of the events from the siblings of the lost partition could be in the output passed to the user, the XEventList compiled from the XLumBlockList is passed to the user. Some of the events on this list might be in the user output, and they have to be removed if the precise knowledge of luminosity is required.

Again, the database approach is fully-operational in this case, so there is no reason not to do splitting in our current data flow.

1.2.4. A Trigger Spanning More Than One Stream

Consider the following example. A user would like to process an EM_MU trigger, which is split between the two streams, ELE_HIGH and MU_HIGH. One of the partitions corresponding to the ELE_HIGH stream is however lost, but the corresponding partition in the MU_HIGH stream is present. How is the luminosity handled in this case?

In this example, the input file list for the user process (step *i* in Fig. 1-1) will contain the partitions from both the ELE_HIGH and MU_HIGH streams. The list of files consumed by the process will have a partition missing from the ELE_HIGH stream. This partition is added to the XLumFile list, and the corresponding luminosity blocks are added to the XLumBlockList. The list of events, XEventList, corresponding to the luminosity blocks in the latter list is then generated and passed to the user. Some of the events on this list could be in the user output and have to be removed for precise luminosity accounting.

In the case when the lost partition was a result of a merge or a split, the information is traced back through the parentage in the way described in the previous two use case examples, and thus the database luminosity tracking is fully operational in this case as well.

This finishes the list of the use case examples and proves that the database luminosity tracking is a fully operational and elegant concept that should be endorsed by this group and recommended as a part of the conceptual streaming design.