Offline Data Compression in a Large Experiment

D. Lellouch and L. Levinson
Weizmann Institute, Rehovoth, Israel

Opal collaboration

Several million candidate events for physics have been recorded so far by the Opal detector at LEP and are stored on disks for analyis. Today this corresponds to $\sim$ 100 Gbytes of disk space and will keep on growing with the machine performance upgrades. We have developped a fast algorithm for data compression and expansion which reduces the disk space by a factor of 3.0 while keeping the full number range and precision for both integer and floating point numbers. This leads to a valuable apparent increase of the I/O bandwith in the analysis environment at a moderate CPU cost for data unpacking.

1 Motivation for data compression

The Opal Detector [1] at LEP can be viewed as a mass producer of candidate events for physics which, for convenient analyses, are stored on disk. The last stage of the 'online' and 'inline' system [2] is the writing of 'production' data-sets which contain both the compressed original raw data, mainly used for detector calibration and monitoring, and the physics oriented DST *(Data Summary Tape)*. At a later stage, typically one year after these data-sets were written, only the DST part is kept on disk, while the compressed raw data are dumped to tape.

This note adresses the further compression of the DST. The effort was triggered by the following considerations which hold for the central analysis complex at Cern as well as for an Institute environment. In both cases the typical configuration consists of several Unix processors accessing data either from directly attached SCSI disks or from a disk server over a high bandwidth network. Experience shows that:

- A physics analysis job running on the data cruncher to which the SCSI disks are attached is in most cases I/O bound rather than CPU bound.
- I/O bandwith in the case where data are read over the network is always the limiting factor.
- Although SCSI disk storage is cheap, CPU is cheaper.
- Disk reliability may become a problem, especially in multi disk environments, while CPU reliability is less a severe constraint.
- Anyhow there is a space shortage: a definite system can not handle more than a finite number of disks. This disk shortage is usually felt before the CPU power shortage.

2 Constraints and basic principle

The main constraints in designing the system were:

- The packing should be done with *no loss* of integer number range or floating number precision, and has to be robust against possible 'pathological' unexpected values.
- In order to be transparent to the user, unpacking should be performed on the fly, i.e. inside the analysis program. The time overhead for this operation should be kept as small as possible.
- All along the online and inline chain, memory management and data I/O is based on the Zebra data management package [3]. The compressed data have to be in a Zebra format in order to preserve all the already exisiting tools.

The first and natural idea is to try a standard Unix packer, based on the classical Lempel-Ziv-Welch algorithm. This yields a modest compression ratio of ≤ 1.5 because the DST consists of very different quantities (e.g. the integer number of tracks found in the central detector and the floating point number representing a vertex fitting chi-square). On the contrary, a program source, for example, exhibits recurrent patterns and can therefore be packed efficiently.

The principle of our compression is the following: the DST consists of several matrices, one per detector type (central detector chambers, electromagnetic calorimeters, hadronic calorimeters, ...). In each matrix, the columns are the quantities of interest (E , θ, ...) while the rows are the individual tracks/clusters. If one *transposes* it, one gets rows of similar numbers that can be efficiently packed. The method is then simple (though cumbersome): compress each quantity according to its own optimized packing scheme. These schemes are determined by an evaluation program which is part of the system.

3 Compression of integer numbers

For each DST integer quantity, the best compression method is determined in two steps:

- Find the number of bits needed to code the quantity (with proper treatments of negative numbers). Example: according to our evaluation sample, 7 bits are sufficient to code the number of tracks in the central detector (0 to 127). The coding will use *eight* bits, the extra one saying: 'quantity is normal (or not)'. If for some pathological event the number of tracks is 1234, the extra bit will say 'abnormal quantity' and 1234 will be written on 32 bits. *This extra bit is the price for data integrity.*
- Build the dictionary of the most frequent values taken by the quantity and find the optimal length for dictionary encoding. In the above example, this optimal is reached for 32 entries in the dictionary, i.e. one bit is used to say 'one of the 32 most frequents values (or not)', then either 5 bits to tell which value is coded or 7 bits representing the not coded value.

- Real life is actually more complex: sophisticated algorithms even use dictionaries with words of *different* length. We restrict the dictionary tree depth to 2 , i.e. in some cases we spend one bit to say : 'this is *the* most frequent entry (or not)'. The reason is that there remains very little to be gained when one has already done that.

Such a simple algorithm compresses the integer numbers in the DST with a factor of *more than 8*.

4 Compression of floating point numbers

In spite of what the current wisdom claims ('nothing can be done with random floating point numbers'), we have reached a factor of 1.9 with the following simple method:

- Build the dictionary of the most frequent values: for example the distribution of ϕ for electromagnetic clusters exhibits 160 spikes corresponding to single block clusters in the barrel electromagnetic calorimeter which has this 160 fold symmetry. 8 bits instead of 32 are then needed to code the number between 1 and 160 instead of the corresponding floating point values.
- In IEEE floating point number representations, 9 bits are used to represent the sign, the exponent and the exponent sign while 23 bits represent the mantissa. For a particular quantity, a very small subset of the exponent potential is used because the dynamical range of any measurement is limited by apparatus response and most DST quantities are directly related to a direct measurement (for example *dE/dX, Energy,...*). Applied to the exponent, the above method of dictionary coding saves several bits per quantity.

To summarize, we take advantage that *measured* quantities often show recurrent values while *computed* quantities have a limited exponent range.

5 Performance

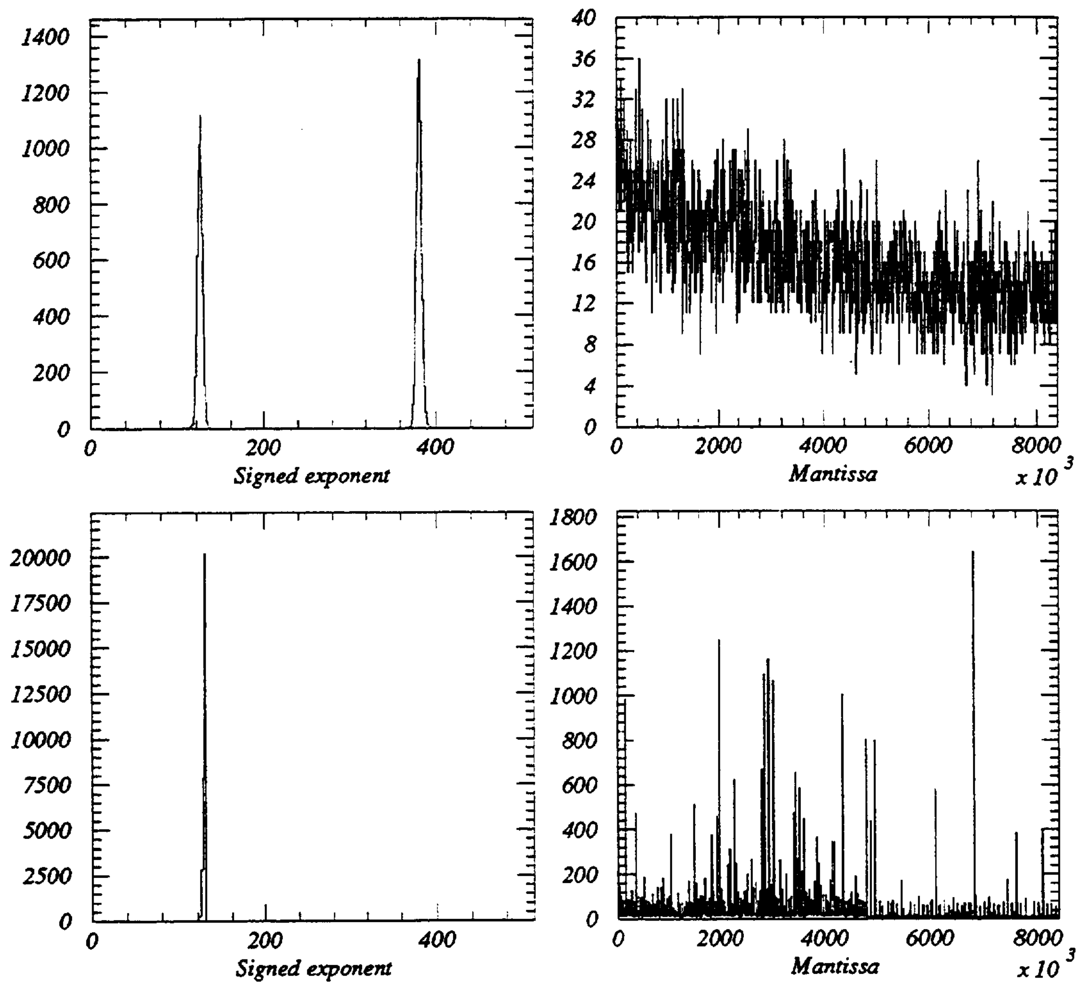
After some further 'technical' changes, such as simplification of the Zebra structure, the overall compression ratio is 3.0 . The cost in CPU time to unpack the data was found to be very moderate: if one considers the simplest analysis job, which does nothing but reading the DST banks from disk and relocating them into memory, the ratio of elapsed CPU times with and without unpacking is 1.9 .

References

[1] The Opal Detector at LEP. Nucl.Instrum.Meth. A305 (1991) 275-319.

[2] The Online system of the Opal Detector. Submitted to Nucl.Instrum.Meth

[3] ZEBRA. CERN Program Library Long Writeup Q100



Distribution of the floating point numbers representing two DST quantities. For each quantity 2 histograms are shown: the exponent distribution ranging from 0 to 511 and the mantissa distribution ranging from 0 to 2**23-1.

- Above: a *computed* quantity. The value of P_X for tracks in the central detector has a limited exponent range (first histogram); the two peaks correspond to positive and negative values. The mantissa do not exhibit a flat distribution (the original distribution of P_X is *not* flat), but can not be efficiently compressed.

- Below: a mainly *measured* quantity. The azimuthal angle of the center of gravity of electromagnetic calorimeter clusters is often the geometrical angle of one calorimeter elementary cell. This is the reason for the numerous peaks in the mantissa distribution. In our algorithm, the recurrent values are first dictionary coded; the underlying flat distribution corresponds to data which can't be compressed, except for their exponent part.