Cluster Arrays

Brett Viren

July 23, 2024

Contents

1 Low-level array representation **1**

2 Graph connection schema **2**

3 Array schema **2**

4 Nodes **4**
- 4.1 Common . . . 4
- 4.2 Activity . . . 4
- 4.3 Wire . . . 4
- 4.4 Blob . . . 5
- 4.5 Slice . . . 5
- 4.6 Measure . . . 6

5 Edges **6**

6 Persistence **6**
- 6.1 Cluster graph files . . . 7
- 6.2 Cluster array files . . . 7
- 6.3 Cluster tensor files . . . 8
- 6.4 Implementation . . . 8

Provides array representations of `ICluster`.

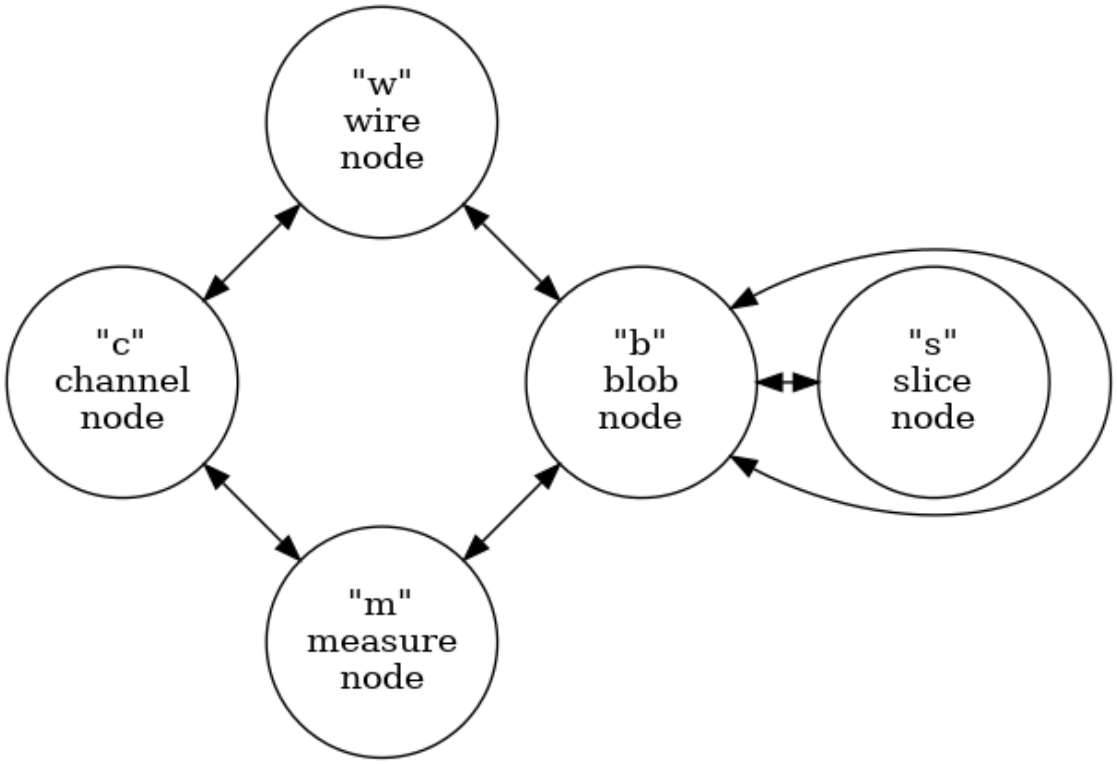
The `ICluster` graph represents identity, attributes and associations of five types of objects related to WCT imaging. In addition to the associations represented internally by the graph some vertex objects (nodes) carry additional information and references to objects that are external to the graph. The `ClusterArrays` class provides methods to "flatten" the internal and external structure to a set of arrays. This document describes the `ClusterArrays` interface and provides guidance on how to use the arrays it produces.

1 Low-level array representation

The API provides arrays in the form of Boost.MultiArray types.

2 Graph connection schema

Production of an `ICluster` graph is described in detail in the Ray Grid document. Its structure is summarized by the *type graph* from that document. This graph illustrates the five types of nodes and the allowed types of edges between node types.



3 Array schema

The array schema closely matches that provided by the python-geometric `HeteroData` interface with the following simplification:

- All node attributes are coerced to double precision floating point scalar values.
- No graph nor edge attributes. This results in a graph being represented by a set of **node arrays** and a set of **edge arrays**.

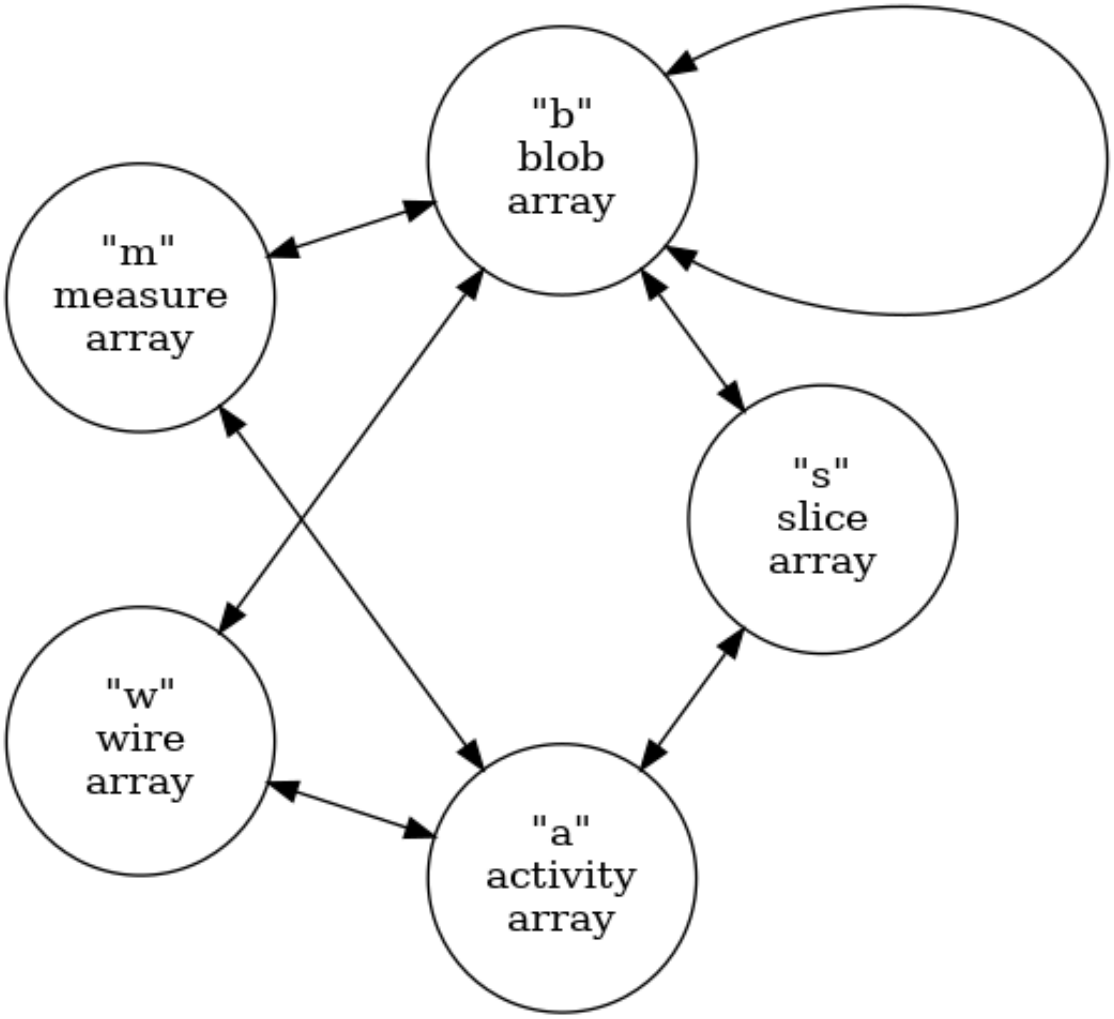
Each **node array** is of a given type. An array type is defined by a tuple that labels the interpretation of the columns of an array. The tuple elements map to node attributes. The rows of an array map to node instances of a given type.

All **edge arrays** have two columns. The first provides indices of rows in a "tail" array and the second in a "head" array". An edge array is mapped to these two node arrays by an array naming convention based on **array codes** and **edge codes**.

Before listing these codes, one structural change must be understood. The `ICluster` graph connection schema described above treats *channel* (and *wire*) nodes as representing physical entities. A given channel node may be reached through multiple `s-b-m-c` paths beginning from multiple slices. The `ICluster` also holds internal to *slice* nodes an `ISlice::activity()`. This provides

a the amount of signal in a channel that contributes to the slice. Furthermore, this *activity map* information is a superset of `s-c` relationships that may be found by following `s-b-m-c` paths. This is because some slice activity may not have contributed to forming blobs in the slice. This activity map can not be coerced into an slice array column (it would be a "ragged array").

In order to faithfully preserve the *activity map* in the array format, the `ClusterArrays` schema **does not follow the cluster graph** schema shown above. Instead, the cluster array type graph is:



Comparing these two type graph representations:

- *channel* nodes are replaced by *activity* nodes
- *activity* represents a channel in a slice (not just a physical channel)
- *activity* holds a signal value and uncertainty
- a *channel-slice* edge is added

The remaining vertex types and allowed edges are the same other as in the original cluster graph schema. In particular, the *wire* vertex type still represents a physical wire which is not specific to any particular slice. All other vertices are per-slice.

4 Nodes

We define *node array codes* as the ASCII value of the lower case initial letter of the node array names: *activity (a), blob (b), measure (m), slice (s)* and *wire (w)*. For example, an activity array will have a label `anodes`. The *edge array codes* are the combination of two node array codes in alphabetical order. For example, the edges between *slice* and *activity* vertices are represented in an array with a name that includes the label `asedges`.

The remainder of this section describes the columns that make up each type of array.

4.1 Common

The arrays are either of node type or edge type. All node type arrays have common definitions for their first **two** columns:

0. *vdesc*, (int) the vertex descriptor counts the node in the graph
1. *ident*, (int) the value from the WCT object's `ident()` method

The additional columns that make each array schema unique are given in the sections below and there these two columns are not repeated.

4.2 Activity

An *activity* represents an amount of signal and its uncertainty collected from a channel over the duration of a time slice.

2. *val*, the central value of the signal
3. *unc*, the uncertainty in the value
4. *index*, (int) the channel index
5. *wpid*, (int) the wire plane id

4.3 Wire

The wire array represents the "geometric" information about physical wire segments.

2. *wip*, (int) the wire-in-plane (WIP) index.
3. *segment*, (int) the number of segments between this wire segment and the channel input.
4. *channel*, (int) the channel ID (not row index).
5. *plane*, (int) the plane ID (not necessarily a plane index).
6. *tailx*, the x coordinate of the tail endpoint of the wire.
7. *taily*, the y coordinate of the tail endpoint of the wire.
8. *tailz*, the z coordinate of the tail endpoint of the wire.
9. *headx*, the x coordinate of the head endpoint of the wire.
10. *heady*, the y coordinate of the head endpoint of the wire.
11. *headz*, the z coordinate of the head endpoint of the wire.

4.4 Blob

A blob describes a volume in space bounded in the longitudinal direction by the duration of a time slice and in the transverse directions by pairs of wires from each plane and it includes an associated amount of signal contained by the volume.

2. *val*, the central value of the signal
3. *unc*, the uncertainty in the value
4. *faceid*, (int) the face identifier (see note)
5. *sliceid*, (int) the slice ident
6. *start*, the start time of the blob
7. *span*, the time span of the blob
8. *min1*, (int) the WIP providing lower bound of the blob in plane 1.
9. *max1*, (int) the WIP providing upper bound of the blob in plane 1.
10. *min2*, (int) the WIP providing lower bound of the blob in plane 2.
11. *max2*, (int) the WIP providing upper bound of the blob in plane 2.
12. *min3*, (int) the WIP providing lower bound of the blob in plane 3.
13. *max3*, (int) the WIP providing upper bound of the blob in plane 3.
14. *ncorners*, (int) the number of corners
15. 24 columns holding *corners as (y,z) pairs*, 12 pairs, of which *ncorners* are valid.

> *The* faceid *is not (necessarily) the* `IAnodeFace::ident()` *but is* `WirePlaneId afid(kUnknownLayer, iface->which(), iface->anode()).ident()`.

4.5 Slice

A slice represents a duration in drift/readout time.

2. *val*, the central value of the signal.
3. *unc*, the uncertainty in the value.
4. *frameid*, (int) the frame ident number
5. *start*, the start time of the slice.
6. *span*, the duration time of the slice.

4.6 Measure

A measure represents the collection of channels in a given plane connected to a set of wires that span one or more blobs overlapping in one wire plane. Its includes an associated signal representing the sum of signals from the participating channels.

2. *val*, the central value of the signal.
3. *unc*, the uncertainty in the value.
4. *wpid*, the wire plane ID

5 Edges

An edge array is of a type constructed as the concatenation of two node types, in alphabetical order. Eg, a `bw` edge array holds edges between `b` and `w` nodes. Each row holds tail and head indices into the corresponding tail and head arrays. An edge array also holds an edge descriptor giving the edge order in the original graph. Note, `boost::graph` edges do not provide this integer representation directly and thus it is expected to be formed as a simple count by the code that writes cluster array schema. Code that reads cluster array schema should add edges in order of this edge descriptor.

0. , (int) edge order
1. *tail*, (int) index of a row in a tail array
2. *head*, (int) index of a row in a head array

6 Persistence

WCT supports persisting clusters in files. A cluster file is an **stream archive file** format that is persisted through WCT iostreams support. It may be a Zip file (`.zip` or `.npz` file extension) or a Tar file with optional compression (`.tar`, `.tar.gz`, etc).

The stream itself consists of a sequence of files and is of a particular stream type. The type is defined by the internal schema and format of the files that comprise the stream. Three types of streams are supported.

cluster graph JSON one JSON file for each `ICluster`, file follows *cluster graph schema*.

cluster array Numpy a sequence of Numpy files for each `ICluster`, files follow *cluster array schema*

cluster tensor JSON+Numpy a sequence of JSON and Numpy files for each `ICluster`, files follow *cluster tensor schema*.

Each stream type is described in more detail in the following three sections followed by a summary of the implementation.

6.1 Cluster graph files

The *cluster graph file schema* is a close mapping of the in-memory `ICluster` object schema mapped to JSON data structures. The main difference relates translating the object pointers in the `ICluster` object schema to JSON plane-old-data support.

The *cluster graph schema* is formally defined by the moo schema file. Data may be validated against this file with the moo program. Similarly, a JSON Schema form of the moo schema file can be generated and 3rd party JSON Schema validation may be applied.

In summary, the *cluster graph schema* requires a JSON document to be evaluated to an JSON `object` with two attributes:

`nodes` an array of object, each element holding an object representing the attributes of one graph node

`edges` an array of object with `tail` and `head` attributes holding indices into the `nodes` array

The `INode` objects in the `ICluster` graph are thus converted to JSON objects. Any `IData::pointer` instances held internally by `INode` instances are referenced by their `ident` number. The `INode` instances can be restored to C++ objects but the internal pointers to `IData` require the program to supply pre-constructed instances. In particular, the `IAnodePlane` instances are required and an optional `IFrame` must be supplied to satisfy the reference held by `ISlice`.

The *cluster graph file schema* differs from *cluster array schema* in two important ways:

1. The c-nodes (physical channel) in `ICluster` graph are retained and not converted to a-nodes (activity in channel in slice).

2. The slice activity map in `ISlice` is directly expressed as an array of activity objects provided by the `signal` attribute of a `slice` node object and not through a-nodes and a-s edges.

The `ClusterFileSource` and `ClusterFileSink` WCT data flow graph components may be used to persist *cluster graph file* schema.

6.2 Cluster array files

The *cluster array file schema* is essentially a direct translation of the *cluster array schema* defined in this document to a persistent representation. It has also been explicitly designed to provide data to graph neural network training where pytorch-geometric `HeteroData` schema is expected.

Each `ICluster` is associated with a set of node array files and edge array files, all in Numpy format. The arrays in these files exactly follow *cluster array schema*. The sequence of array files may have names similar to:

```
cluster_6501_wnodes.npy
cluster_6501_snodes.npy
cluster_6501_bnodes.npy
cluster_6501_mnodes.npy
cluster_6501_anodes.npy
cluster_6501_bsedges.npy
cluster_6501_bwedges.npy
cluster_6501_asedges.npy
cluster_6501_bbedges.npy
```

```
cluster_6501_bmedges.npy
cluster_6501_awedges.npy
cluster_6501_amedges.npy
```

There are three identifiers encoded in the file names:

`prefix` The prefix `cluster` marks these files as following cluster array file schema.

`ident` The number (eg `6501`) provides the `ICluster::ident()` and allows for associating the set of files.

`<n>nodes` Marks the array as a nodes array for nodes of type `<n>`

`<th>edges` Marks the array as an edges array for edges between nodes of type `<t>` (tail) and `<h>` (head).

The `ClusterFileSource` and `ClusterFileSink` can persist *cluster array file* schema.

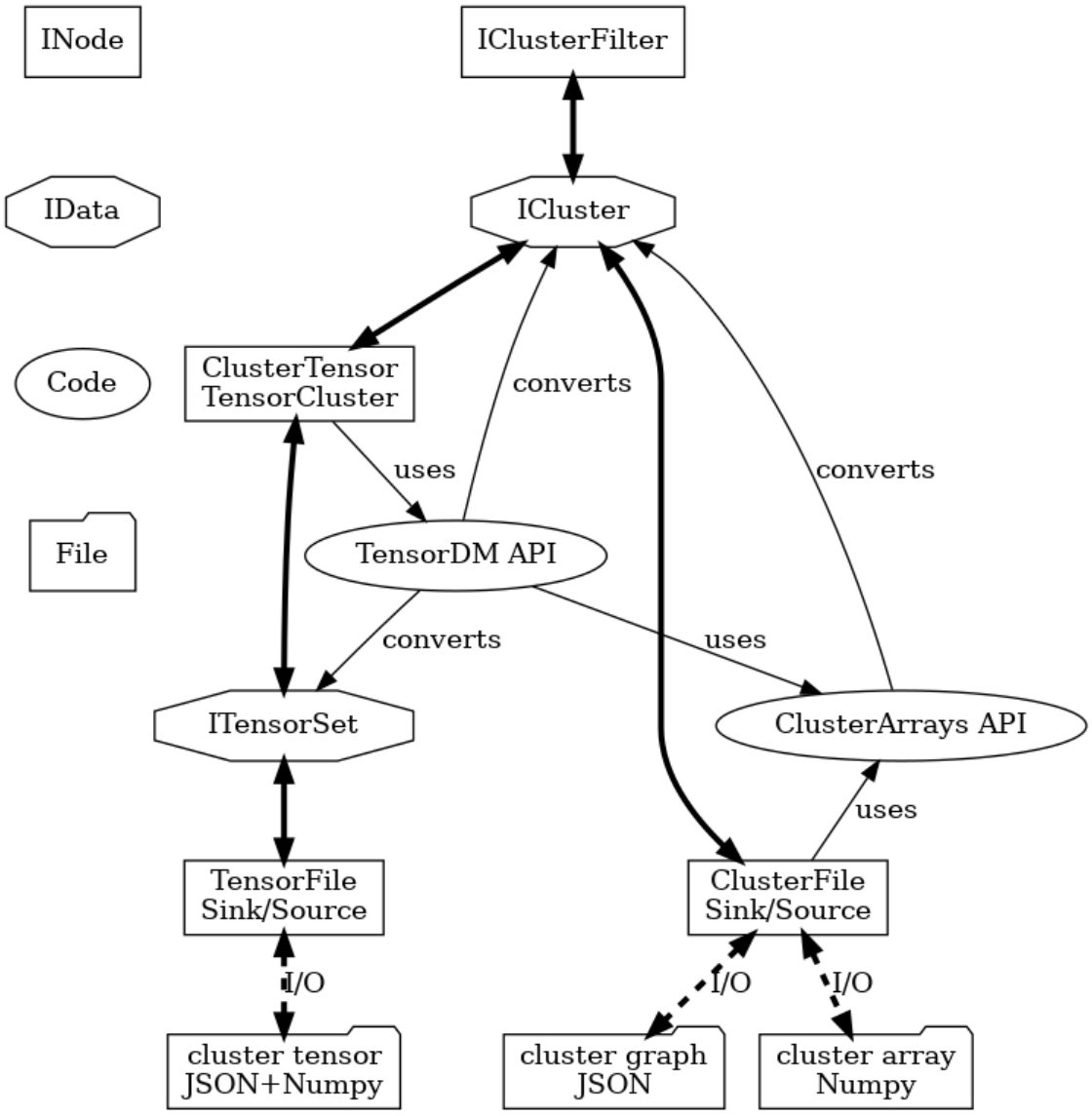
6.3 Cluster tensor files

The *cluster tensor file schema* is also essentially a direct translation of the *cluster array schema* to a persistent representation. The actual file representation is provided by the general purpose WCT tensor data model. Thus there is no specific WCT component that persists `ICluster` to cluster tensor files. Rather, converters from `ICluster` to `ITensor` are used and the `ITensor` representation is persisted.

With already two schema implemented, a third may seem excessive. The motivation is to unify persistence of all `IData` types in a single I/O model (the WCT tensor data model). Currently, this model is implemented by a WCT iostreams type consisting of JSON and Numpy files. However, the model intentionally mimics that of HDF5 and support for this file format is expected at some point in the future.

6.4 Implementation

The implementation landscape for persisting `ICluster` is illustrated:



Solid thick black edges represent exchange of data between WCT data flow graph nodes. Solid thin lines represent code dependency. Thick dashed edges represent file I/O. Note `ClusterFile{Source,Sink}` directly convert `ICluster` while `TensorFile{Source,Sink}` are general and the conversion to `ITensor` is performed by `TensorCluster` and `ClusterTensor`.