IceCube Photonics – An Overview for Computing Systems

by
Steve Barnet and Paolo Desiati – UW IceCube Research Center
June 18, 2009

Overview of the IceCube detector

IceCube is a neutrino detector under construction at the South Pole. The detector comprises optical modules buried deep in the ice. This ice is dark and very clear making it an ideal medium for detecting the Cerenkov radiation generated as a by-product of a neutrino interaction. The target geometry of the IceCube Neutrino Observatory consists of a regularly spaced array of 4800 photomultiplier tubes buried between 1450 to 2450 meters below the surface of the South Pole ice and a surface air shower detector, IceTop, Consisting of four photomultiplier tubes in two surface tanks at each In-Ice string location. An additional six strings with modules closer together (called the DeepCore) will provide sensitivity to lower energy events.

Photonics Tables

As mentioned, the ice itself is the medium of the detector, which will instrument approximately one cubic kilometer of ice when completed. For simulation, we must model the physics events, the propagation of light through the ice, and the detector response. The photonics tables are look-up tables we use in simulation to determine the amount of light the IceCube detector sensors see as a result of a physics event.

The total size of those tables is 14 GB which does not fit conveniently into main memory on contemporary computing systems. Given the large table size, in simulation production we split the tables into subsets which can fit reasonably in memory. The tables are split according to the physical event zenith angle; specifically, we divide the tables into 10 degree bins. Since the 14 GB tables cover 180 degrees, this means that we split the tables into 18 subsets of 780 MB each.

We process each subset in series, where in each step we load the 780 MB of the corresponding subset into memory and we proceed with simulation. This means that each job will eventually read the entire set of tables into memory during the course of its processing. Additional memory is then used by the subsequent detector simulation, that might easily take up to 2 GB of main memory (or more for high energy event simulation).

Since each job will eventually read the entire 14 GB table, we currently require the tables to be stored locally on each worker node or on a storage unit that can be reached by the execution nodes. Our experience suggests that local storage on the worker node is most effective as even a relatively small number of jobs, say O(100), can badly overload a shared storage system. Some of our testing has indicated that cluster filesystems (eg Lustre) may be able to handle higher loads (with some loss of per job performance) depending on configuration of the system.

Work in progress

This situation is problematic when we consider the use of grid computing. While the table size is not particularly large, requiring permanent local storage on the worker node or a high capacity central storage system represents additional effort and expense for sites. In addition, determining the availability of tables at sites or on worker nodes adds complexity in scheduling and troubleshooting.

We are working on a flexible solution where we submit jobs using the Condor Dag tool. Using the Dag we process each subset in parallel (instead of serially) and we can proceed in three possible different ways :

1. We keep the full 14 GB tables in each node (or on a storage unit with fast connection) and each Dag uses a given subset of those tables.

2. We keep the different table subsets in specified classes of nodes and we submit jobs by matching the specific Dag to the corresponding class of node

3. We ship the 780 MB of tables to the execution node for any given Dag and we store the tables in the given execution node until automatically removed.

In particular, option three shows promise for operating in a grid environment. In principal, the first job to process a bin would download the appropriate bin to local storage on the worker node; the bin would be left on the node. Subsequent jobs which needed to process that bin would be scheduled on that node and would reuse the file rather than copying it to the node again. Note that there could actually be many nodes processing the same bin, but each node would need to download the bin only one time for a certain number of jobs (until the bin was automatically removed, perhaps).

There may be other grid based mechanisms which would address our needs. We are exploring many options which would enable us to expand the pool of resources usable for our simulation and will consider alternative approaches to addressing our issues if it will improve our ability to use grid resources.