IceCube Instrumentation and Online Systems

The IceCube Collaboration

1 Introduction

1.1 IceCube Science

(1 page)

1.2 A Functional Description of the IceCube Instrument

In order to observe astrophysical neutrinos, the primary science goal of the experiment, IceCube exploits the fact that charged particles moving through the ice at super-luminal speed emit Cherenkov photons. An enormous detection volume is required since the cross-sections of neutrinos are small for producing secondary charged particles in interactions with ordinary matter. The glacial ice cap at the South Pole is about 3 km thick and therefore predestined as operation site since it is not only offering aplenty interaction material but also a medium with unmatched high quality. Cherenkov light is produced in cascades of neutrino-induced muons penetrating the deep, other high-energy particles as well as generated by by cosmic-ray muons that result from cosmic-ray interactions in the atmosphere above Antarctica. Due to a Cherenkov photon yield of $\mathcal{O}(10^5)$ visible photons per GeV of shower energy, the long optical attenuation length in South Pole ice and large-area photomultipliers (PMTs) it is possible to instrument cubic kilometers of ice with a rather wide spacing of detectors. The basic detection unit in IceCube in order to capture the Cherenkov light is the digital optical module or *DOM* which is covered in great detail in Sec. **??**. Encapsulated in a 1/2" thick glass pressure sphere to withstand the extreme pressure in the deep ice, the main components of a DOM are a 10" PMT, embedded high-voltage generation, a flasher calibration board, and a mainboard containing the analog and digital processing circuitry for PMT pulses. The digitized data is fed to a central computing facility at the surface via a unique cable system, see Sec. **??**. Aspects of detector deployment and ice drilling are covered in Sec. **??**. An overview of the data flow as well as its readout, processing and filtering are subjects of Sec. **??** where we also cover the data handling, monitoring and operational performance of the observatory. The IceCube instrument consists of three sub-detectors – IceCube, DeepCore and IceTop – using the same instrumentation design of embedded digital optical modules and associated surface readout. A schematic layout of the array is shown in Fig. 1

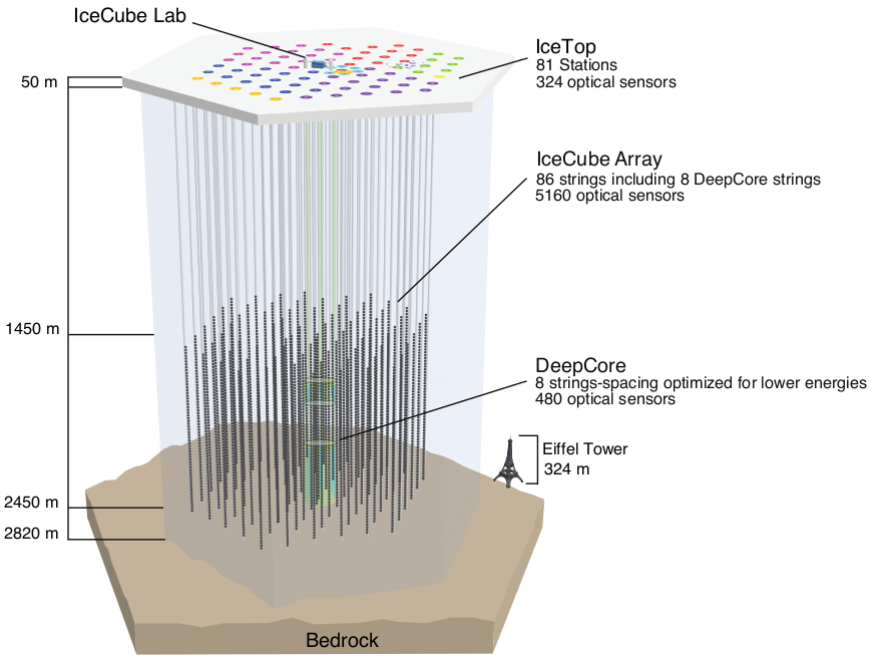

Figure 1: The IceCube Neutrino Observatory with its sub-array DeepCore and the air shower array IceTop.

1.2.1 IceCube

In order to detect the Cherenkov photons emitted by charged particles traversing the ice, 5160 DOMs are deployed between 1450 m and 2450 m below the glacial surface on 86 vertical strings each holding 60 DOMs deployed along a copper cable. The main *deep-ice* array consists of 78 strings with a vertical separation of the DOMs on each string of 17 m. These strings are deployed on a hexagonal grid with 125 m horizontal spacing and spanning a volume of one cubic kilometer of ice. *Talk about energy range covered by this instrumentation and primary science goal*

1.2.2 DeepCore

The remaining subset of in-ice DOMs is deployed in the deep ice below a depth of 1750 m forming a denser instrumented volume. This sub-array, the DeepCore [1] consists of eight specialized and closely spaced strings of sensors located around the central IceCube string. Its inter-string spacings of 75 m and inter-DOM spacings of 7 m are optimized for the detection of atmospheric neutrinos with energies typically in the range from 100 GeV to 400 TeV [3].

1.2.3 IceTop

The air shower array IceTop [4] consists of 81 Cherenkov tanks filled with clear ice that are arranged in pairs on the same, approximately 125 m, triangular grid onw hich the in-ice array is deployed. The two tanks at each surface station are separated from each other by 10 m. Each tank contains two standard IceCube DOMs. Air showers initiated in the atmosphere by cosmic rays are typically spread over a number of stations. The light generated in the tanks by the shower particles (electrons, photons, muons and hadrons) is a measure of the energy deposit of these particles in the tanks.

2 The Digital Optical Module

(Chris Wendt; 10 pages)

2.1 A Functional Description of the DOM

2.2 Mainboard

Cite the DAQ article[5].

2.3 The Photomultiplier

Cite the PMT article[2].

2.4 High Voltage

2.5 Flasher Board

2.6 Pressure Housing and Optical Gel

2.7 Mu-metal Magnetic Shield

2.8 Cable Penetrator

2.9 Mechanical Mounting - Harness Assembly

2.10 Production and Testing

2.11 Calibration

2.11.1 DOMCal

2.11.2 Flasher Calibrations

2.12 Performance and Reliability

We have over N DOM years in ice. What can be said about the reliability? This section could be quite important.

3 The Cable Systems

3.1 Surface Cables

3.2 Surface-to-DOM Cables

3.3 Surface Junction Boxes

4 Drilling and Deployment

5 Online Systems

(John K; 12-15 pages)

The IceCube online systems comprise both the software and hardware at the detector site responsible for data acquisition, event selection, monitoring, and data storage and movement. As one of the goals of IceCube operations is to maximize the fraction of time the detector is sensitive to neutrino interactions ("uptime"), the online systems are modular so that failures in one particular component do not necessarily prevent the continuation of basic data acquisition. Additionally, all systems are monitored with a combination of custom-designed and industry-standard tools so that detector operators can be alerted in case of abnormal conditions.

5.1 Data Flow Overview

The online data flow consists of a number of steps of data reduction and selection in the progression from photon detection in the glacial ice to candidate neutrino event selection, along with associated secondary data streams and monitoring. An overview of the data flow is shown in Fig. **??**.

Since the majority of photons detected by the DOMs are dark noise, a first-level *local coincidence* (LC) is formed between neighboring DOMs deployed along the same cable, using dedicated wire pairs within the in-ice cable. DOM-level triggers, or *hits*, with corresponding neighbor hits are flagged with the LC condition, while hits without the condition are compressed more aggressively. The LC time window as well as the span of neighbor DOMs up and down the cable can both be configured, with standard settings of a $\pm 1 \mu s$ coincidence window and neighbor span of 2 DOMs.

All DOM hits are read out to dedicated computers on the surface by the data acquisition system (DAQ). The next level of data selection is the formation of *triggers* by the DAQ system. LC-flagged hits across the detector are examined for temporal and in some cases spatial patterns that suggest a common causal relationship. A number of different trigger algorithms run in parallel, described in Sect. **??**. All hits (both LC-flagged and non-LC hits) within a window around the trigger are combined into *events*, the fundamental output of the DAQ, and written to disk. The event rate is approximately 2.5 kHz but varies with the seasonal atmopheric muon flux, and the total DAQ data rate is approximately 1TB/day.

The DAQ also produces *secondary streams* that include time calibration, monitoring, and DOM scaler data. The scaler data, which is monitoring the noise rate of each DOM in 1.6 ms bins, is used in the supernova data acquisition system [**?**] to detect a global rise from many $O(10)$ MeV neutrino interactions occuring in the ice from a Galactic core-collapse supernova. The time calibration and monitoring streams are used to monitor the health and quality of the data-taking runs.

The raw DAQ event data is then processed further with a number of *filters* in order to select a subset of events (less than 10%) to transfer over satellite to the Northern Hemi-

sphere (see Sect. **??**). Each filter, typically designed to select events useful for a particular physics analysis, is run over all events using a computing cluster in the IceCube Lab. Because of limitations both on total computing power and bounds on the processing time of each event, only fast directional and energy reconstructions are used. The processing and filtering system is also responsible for applying up-to-date calibrations to the DAQ data; processed events, even those not selected by the online filters, are stored locally for archival.

A dedicated system for data movement handles the local archival storage to tape or disk, as well as the handoff of satellite data (see Sect. **??**). This includes not only primary data streams but also monitoring data, calibration runs, and other data streams.

[Add experiment control paragraph]

Figure: data flow diagram indicating major subsystems to be described in this section: DAQ (incl. SNDAQ), PnF, I3Live, and SPADE/JADE.

5.2 SPS and SPTS

South Pole System: breakdown of computing hardware used at the pole between hubs, DAQ, PnF, other machines, and infrastructure. Internal network bandwidth. Redundancy, system monitoring (Nagios), and paging system.

Brief mention of SPTS as northern test and validation system. Replay capabilities.

5.3 Data Readout and Timing

5.3.1 Communications and Cable Bandwidth

Description of communications protocol and messaging strategy. Reference to RAPCal and how it fits in. Event compression.

5.3.2 Master Clock System

The GPS clock and time string fanout tree, from master clock (and hot spare), to Tier I and Tier II fanouts, the DSB card, and into the DOR card.

5.3.3 DOR Card and Driver

DOR card description: comms / readout, power control and measurement, RAPCal initiation, and clock string readout. Clock modes (internal / external). DOMs per card and cards per hub.

Brief description of driver. Proc file interface. Data transfer over PCI bus via DMA.

Figure: Combined clock fanout tree hierarchy and hub diagram (DSB and DOR cards, power distribution).

5.4 Processing at the Surface

(Dave G; 2-3 pages)

IceCube's data acquisition system (DAQ) is a set of components running on dedicated servers in the IceCube Lab. As described in **??**, physics data is read from the DOMs by the StringHub component and a minimal representation of each HLC hit is forwarded to the "local" Trigger components. The "local" Trigger components run these minimal hits through a configurable set of algorithms and form windows around interesting temporal and/or spatial patterns. These time windows are collected by the "global" Trigger and used to form non-overlapping trigger requests. These trigger requests are used by the Event Builder component as templates to gather the complete hit data from each StringHub and assemble the final events.

5.4.1 DOMHub and Hit Spooling

The *StringHub* is responsible for periodically reading all available data from each of its connected DOMs and passing that data onto the downstream consumers. It also saves all hits to a local "hit spool" disk cache, as well as queuing them in an in-memory cache to service future requests from the *Event Builder* for full waveform data.

The StringHub component is divided into two logical pieces. The front-end *Omicron* controls all of the connected DOMs, forwarding any non-physics data to its downstream consumers and sorting the hits using an HKN1 tree before passing them to the back-end *Sender*. The Sender caches SLC and HLC hits in memory, then forwards a few fields from each HLC hit to the local Triggers.

After the Trigger components have determined interesting time intervals, the Event Builder will send each interval to the Sender, and it will return a list of all hits within the interval, pruning the in-memory hit cache of all older hits after each interval.

In order to avoid saturating the 2006-era network, the StringHub sends only simplified HLC hit data to the local Triggers. Fields extracted from the full HLC are the string and DOM, the time of the hit, and a few DOM trigger fields (type, mode, configuration ID), for a total of 38 bytes per hit.

Dave G doesn't know the details of this

One core assumption of the DAQ is that each component operates on a time-ordered stream of data. IceCube's DAQ uses its *Splicer* to accomplish this. The Splicer is a common object which gathers all input streams during the setup phase; no inputs can be added once it's started. Each stream pushes new data onto a "tail" and the Splicer uses the HKN1 algorithm to collate the data from all streams into a single output stream. When a stream is closed, it pushes an end-of-stream marker onto its tail which causes the Splicer to ignore all further data.

The HKN1 (Hansen-Krasberg-1) algorithm at the core of the Splicer uses a tree of *Nodes* to sort a constant number of time-ordered data streams. Each Node contains a

queue of pending data, a handle for a peer Node, and another handle for a sink Node (shared with the peer) to which sorted data is sent.

The tree is assembled by associating each input stream with a separate Node and adding the Node to a list. This list is then run through a loop which pulls off the first two Nodes, links them together as peers and creates a new Node which will act as the sink for both Nodes. The new Node is then added to the list. The loop exits when there is only one Node left.

When a new piece of data arrives, it is added to the associated Node's queue. That Node then goes into a loop where it compares the top item from its queue with the top item from the peer's queue and the older of the two items is passed onto the sink Node (which adds the item to its queue and starts its own comparison loop.) This loop stops as soon as the original Node or its peer has no more data.

Along with physics data, DOMs produce three additional streams of data. The scaler data, which is monitoring the noise rate of each DOM in 1.6 ms bins, is used in the supernova data acquisition system [**?**] to detect a global rise from many $O(10)$ MeV neutrino interactions occuring in the ice from a Galactic core-collapse supernova. The time calibration and monitoring streams are used to monitor the health and quality of the data-taking runs.

As hits move from the front end to the back end, they are written to the "hit spool", a disk-based cache of files containing hits. These files are written in a circular order so that the newest hits overwrite the oldest data, and the first hit time for each file is stored in an SQLite database.

Subsystems such as SnDAQ and HESE can send time interval requests to a HitSpool master daemon which then gathers the hits in that interval from each hub and forwards them to a final "sender" daemon which bundles them up and submits them to JADE for transfer to the North where the full waveforms can be used for further analysis.

One problem with the current DAQ design is that it only reads data when the DAQ is running, so the detector is essentially "off" during hardware failures or the periodic full restarts of the system. There is a plan to split the StringHub into several independent pieces to eliminate these blind spots. The front end (Omicron) piece will be moved to an always-on daemon which continuously writes data (including secondary, non-physics data and other metadata) to the disk cache. Part of the back end (Sender) piece will become a simple HitSpool client which reads data from the disk cache and sends it to the downstream consumers, while another simple component will listen for time intervals from the Event Builder and send back lists of hits taken from the hit spool.

5.4.2 Supernova System

SN secondary stream from DOMs and SNDAQ reference. Interface to hitspooling.

5.4.3 Triggers

General description of trigger architecture. Separation of trigger window and readout window. How trigger windows depend on geometry. Thorough escription of all different trigger algorithms. Trigger and readout window merging.

Table: standard settings for triggers

Figure: trigger windows and readout windows.

Figure: example bright multi-trigger event.

Figure: SLOP triplet geometry?

5.4.4 Event Building

Readout requests to StringHub components and packaging of waveforms into events. Spooling to disk and interface to PnF.

5.4.5 Configuration

Tree of XML configuration files for components, triggers, and DOM settings.

5.4.6 Distributed Network Control

CnC server description. XML-RPC.

5.5 Online Filtering

(Erik B.; 3-4 pages)

5.5.1 Overview

Big picture view of what PnF is and does (route data from DAQ output to files for pickup by data handling, applying calibration, recos and filters along the way, TFT content, etc)

5.5.2 System Design

Cover the overall system design and tech used (CORBA, I3Inlet/I3Outlets, etc) and how all pieces fit together and are controlled (mention pf2live). Words about historical operation in Plan B in early seasons, plan A.

5.5.3 Components

A brief description of each PnF component.

I3DAQDispatch

PnF Server

PnF Clients

DB Cache

Realtime Followup server and clients

5.5.4 Performance

Overall rates and known limits (server + I/O limits)

Average system latency Figure: plot of typical latency including run transition?

5.6 Data Handling

(P. Meade; 1 page)

Generation description of system architecture. Stream definitions, dropboxes, and data pickup. Archival vs. transfer to TDRSS system.

Data handling is provided by three servers named jade02, jade03, and jade04. The jade servers operate independently of one another and each of them are capable of handling the nominal data volume by itself. Having three servers allows for data handling to continue seamlessly in case of hardware failure or maintenance.

Each server runs a copy of the Java Archival and Data Exchange (JADE) software (stylized "jade"). As its name implies, the jade software is written in the Java programming language. It is a reimplementation and expansion of earlier prototype software called South Pole Archival and Data Exchange (SPADE), written by Cindy Mackenzie. The jade software has four primary tasks: consumption, archival, satellite transmission, and real-time transmission.

The jade software is configured with a number of data streams, which consist of a data server, a dropbox directory, and a filename pattern. The data stream dropbox directories are checked on a regular basis for new files. A file pairing scheme (binary and semaphore) prevents files from being consumed before they are finished being produced. For each file, a checksum calculated on the data server is compared to a checksum calculated on the jade server. This method ensures that the file was copied without error. After this, the original data file is removed from the data host.

After consumption, files are routed according to the configuration of their data stream. Files that are too large to send via the satellite link are archived to a configurable number of archival media copies. The prototype SPADE software archived to LTO tapes, while the later jade software archives to large (2+ TB) hard disk drives. All of the archival data is buffered on the jade server until the archival media is complete. In case of failure while

creating the archival media, all of the files can be immediately written to fresh archival media with a single command.

Files that are too large to send via the real-time link, but small enough to send via the satellite link are queued for satellite transmission. The jade software attempts to bundle multiple files together into 1 GB bundle archives to allow satellite link operators to manage the daily data transmission. Very large files (¿1 GB) are split apart into multiple 1 GB bundles for the same reason. The jade software will only transfer a configurable number of bundles to the satellite relay server. If satellite transmission is not possible, the jade software will buffer the excess bundles on the jade server, to avoid flooding the relay server unnecessarily.

Small files (¡50 KB) with high priority status information are sent via the real-time link. The real-time link is provided by the IceCube Messaging Service (I3MS). The jade software uses JeroMQ, a pure Java implementation of the ZeroMQ (ZMQ) protocol, to connect to I3MS. In cases where the real-time link is not available, I3MS will queue the messages to be sent when the link becomes available. All I3MS messages are also sent to jade to send via the satellite link to ensure delivery if the real-time link should be unavailable for an extended period of time.

5.7 IceCube-Live and Remote Monitoring

IceCube operations are controlled and monitored centrally by IceCube-Live, a suite of high-level software implemented mostly in the Python language. IceCube-Live consists of two major components: LiveControl, responsible for controlling data taking operations and collecting monitoring data, and the IceCube-Live website, responsible for processing and storing monitoring data as well as presenting this data in webpages and plots that characterize the state of the IceCube detector.

5.7.1 LiveControl

LiveControl is the central control component of IceCube operations. The program executes in the background as a daemon and accepts user input through XML-RPC. Operators typically enter commands and check the basic detector status using a command-line interface. LiveControl is responsible for controlling the state of DAQ and online-processing, starting and stopping data-taking runs, and recording the parameters of these runs. Stardard operation is to request a run start, supplying a configuration file specifying the DOMs to include in data taking. LiveControl then records the run number, configuration, start time, etc. and sends a request for DAQ to begin data taking. After data taking commences successfully, LiveControl waits a specified amount of time, generally eight hours, then stops the current run and automatically starts a new run using the same configuration. This cycle continues until stopped by a user request or a run fails. In case of failure, LiveControl attempts to restart data taking by starting a new run. Occasionally a hardware

failure occurs, and it is impossible to start a run with the supplied configuration because requested DOMs are unpowered or temporarily unable to communicate with the IceCube DAQ. In this case, LiveControl cycles through predefined partial-detector configurations in an attempt to exclude problematic DOMs. This results in taking data with half of the IceCube strings or fewer, but it greatly reduces the chance of a prolonged complete outage where no data at all is recorded.

A secondary function of LiveControl is the collection, processing, and forwarding of monitoring data from DAQ, online-processing, and other components. Monitoring data consists of a JSON dictionary with a well-definied format including a creation time, sender name, priority, data name, and either JSON data or a single integer or floating-point value. This data is forwarded to LiveControl using ZeroMQ and queued internally for processing. A few monitoring quantities indicate serious problems with the detector, e.g. the online-processing latency is too high. LiveControl provides a database of checked monitoring values indexed by service and data name and raises an alert if the value is out of the specified range or hasn't been received in a specified amount of time. The alert usually includes an email to parties responsible for the affected subsystem and, for serious problems, triggers an automated page to winterover operators. Several other types of monitoring data trigger a response by LiveControl. These include alerts generated internally by subsystems, and such alerts may trigger emails and pages from LiveControl. All monitoring data are forwarded to the IceCube-Live website for further processing and display.

5.7.2 IceCube-Live Website

Priority	Transport System	Daily Messages	Daily Data	Typical Latency
1	ITS	10,000	1 MB	1 minute
2	I3MS	150,000	5 MB	1–5 minutes
3	JADE	300,000	100 MB	1 day

Table 1: Statistics for IceCube monitoring messages

Two operational copies of the IceCube-Live website exist: one inside the IceCube network at the South Pole, fed directly by LiveControl using ZeroMQ, and one in the Northern Hemisphere fed by several message transport systems depending on the priority of the particular monitoring data, summarized in table 1. IceCube generates approximately 300,000 monitoring messages daily. These compress to ∼100 MB, are sent over JADE, and are typically available in the Northern Hemisphere within 24 hours. Messages of priority level 2 or higher are additionally sent over the Iridium RUDICS link using I3MS, and messages of priority 1 and higher are sent over the Iridium ITS link.

Messages reaching the website are ingested by the DBServer daemon, processed, and inserted into one of several SQL database tables or a MongoDB database depending on

content. Messages also may contain directives requesting DBServer to send email, specifying recipients and content, or requesting that the monitoring message be published using ZeroMQ PUB-SUB so it can be ingested by an external process. The IceCube-Live website itself uses the Django framework and contains pages that create sophisticated views of monitoring data stored in either SQL or MongoDB. These pages include a front page that displays active alerts and plots of event rates and processing latencies from the previous few hours and a page for each run that displays start time, stop time, and other essential data. The run page contains low-level diagnostic data that includes e.g. charge histograms, digitizer baselines, and occupancy for each DOM, and is used to diagnose problems with detector components during the run and to determine if the run can be used in physics analysis.

Finally, the IceCube website transmits messages using ITS and I3MS to LiveControl running at the South Pole. This capability is used to connect the popular Slack chat service with the IceCube-Live website running at the South Pole, allowing the IceCube winterover operators to chat with experts in the Northern Hemisphere during periods with no geosynchronous satellite connectivity. Southbound messages are also used to trigger the HitSpool service by the request of experts in the Northern Hemisphere if an interesting event (e.g. a SNEWS alert) occurs.

5.8 Operational Performance

(John K.; 1 page)

Explanation of how design choices, system monitoring, and winterovers result in high uptime. Discussion of median downtime and various causes of downtime. Possible basic failure analysis of hardware components.

Figure: DAQ full uptime and clean uptime percentage.

Figure (optional): Downtime histogram.

6 Outlook

Discuss Gen2 here.

References

[1] R. Abbasi, Y. Abdou, T. Abu-Zayyad, M. Ackermann, J. Adams, J. Aguilar, M. Ahlers, M. Allen, D. Altmann, K. Andeen, et al. The design and performance of IceCube DeepCore. *Astroparticle physics*, 35(10):615–624, 2012.

[2] R. Abbasi, Y. Abdou, T. Abu-Zayyad, J. Adams, J. Aguilar, M. Ahlers, K. Andeen, J. Auffenberg, X. Bai, M. Baker, et al. Calibration and characterization of the IceCube photomultiplier tube. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 618(1):139–152, 2010.

[3] R. Abbasi, Y. Abdou, T. Abu-Zayyad, J. Adams, J. Aguilar, M. Ahlers, K. Andeen, J. Auffenberg, X. Bai, M. Baker, et al. Measurement of the atmospheric neutrino energy spectrum from 100 GeV to 400 TeV with IceCube. *Physical Review D*, 83(1):012001, 2011.

[4] R. Abbasi, Y. Abdou, M. Ackermann, J. Adams, J. Aguilar, M. Ahlers, D. Altmann, K. Andeen, J. Auffenberg, X. Bai, et al. IceTop: The surface component of IceCube. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 700:188–220, 2013.

[5] R. Abbasi, M. Ackermann, J. Adams, M. Ahlers, J. Ahrens, K. Andeen, J. Auffenberg, X. Bai, M. Baker, S. Barwick, et al. The IceCube data acquisition system: Signal capture, digitization, and timestamping. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 601(3):294–316, 2009.